Tema 3. Eventos en jQuery

Alejandro Amat Reina y Pablo Matías Garramone Ramírez

ÍNDICE

1. Objetivos

En este tercer tema se pretenden conseguir los siguientes objetivos:

- Ser capaz de recorrer los elementos individuales de un conjunto de elementos seleccionados mediante un selector jQuery.
- Entender el funcionamiento de los eventos en jQuery.
- Trabajar con la posición y tamaño de los elementos de la página.
- Tratar de reutilizar funcionalidades ya implementadas para construir nuevas funcionalidades.

2. Recorrer los elementos seleccionados

Como hemos visto en temas anteriores utilizando selectores podemos obtener uno o más elementos del DOM agrupados en un conjunto. De esta forma, cuando llamamos a un método sobre un objeto jQuery, este método afectará a todos los objetos seleccionados (esto es así para la mayoría de los métodos, aunque existen algunos que están pensados para trabajar con un único elemento, por ejemplo el método `.attr()`, que sólo obtiene el valor del primer elemento del conjunto).

En algunos casos, nos puede interesar recorrer los elementos del conjunto seleccionado y realizar tareas sobre cada uno de ellos. Para este propósito tenemos el método `.each()`.

Cuando invocamos al método, este realiza una iteración sobre cada uno de los elementos que conforman el conjunto seleccionado. Para ello, tendremos que pasarle una función que se ejecutará una vez para cada elemento. Esta función recibirá como parámetro el índice del elemento actual comenzando por 0. Además, como la ejecución de la función se realiza en el contexto del elemento actual del conjunto, podemos acceder a él a través de `this`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>.each()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script  src="http://code.jquery.com/jquery-3.1.0.min.js"
             type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("div").each(function (i)
        {
          if (this.style.color != "blue")
            this.style.color = "blue";
        });
      });
    </script>
  </head>
<body>
  <div id="primera-entrada" style="color:red">
    Texto de la primera entrada
  </div>
  <div id="segunda-entrada" style="color:red">
    Texto de la segunda entrada
  </div>
  <div id="tercera-entrada" style="color:red">
    Texto de la tercera entrada
  </div>
</body>
```

```
</html>
```

Ejemplo 1: each.html

En este ejemplo manejamos `this` como elemento DOM. Estamos mostrando todos los textos de los `<div>` en azul en lugar de en rojo. Si quisiéramos obtener un objeto jQuery con el elemento que estamos recorriendo, sólo tendríamos que seleccionarlo. El código anterior podría utilizar objetos jQuery de la siguiente forma:

```
<!DOCTYPE html>
<html>
  <head>
    <title>.each() con objetos jQuery</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                    type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("div").each(function (i)
        {
          if ($(this).css("color") != "blue")
            $(this).css("color", "blue");
        });
      });
    </script>
  </head>
<body>
  <div id="primera-entrada" style="color:red">
    Texto de la primera entrada
  </div>
  <div id="segunda-entrada" style="color:red">
    Texto de la segunda entrada
  </div>
  <div id="tercera-entrada" style="color:red">
    Texto de la tercera entrada
  </div>
</body>
</html>
```

Ejemplo 2: eachConObjetosjQuery.html

3. Eventos

Los eventos son fundamentales para las aplicaciones Web, y JavaScript estaría muy limitado si no existieran. El problema que nos encontramos en este punto es que diferentes navegadores tratan los eventos de forma distinta, por lo que tenemos que adecuar nuestro código a cada uno de estos navegadores. De hecho, como se puede ver en la siguiente tabla, existen tres modelos de eventos diferentes.

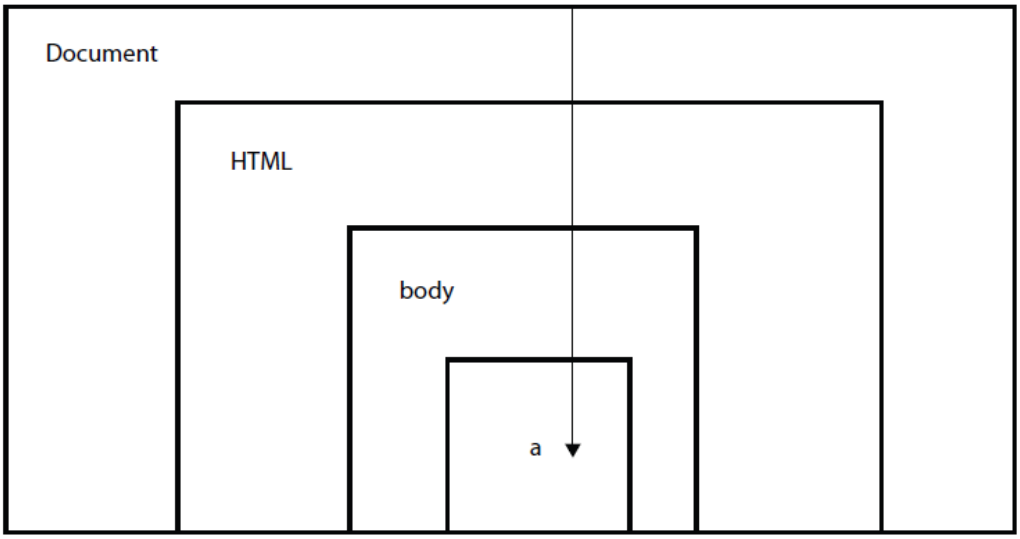
Nivel DOM	Descripción
0	Modelo de eventos original desarrollado por Netscape (todos los navegadores soportan este modelo)
2	Modelo de eventos estándar, más avanzado que el nivel 0, pero no soportado por IE.
IE Model	Un modelo de eventos separado del estándar, que sólo soporta IE.

Tabla 1: Modelos de eventos existentes

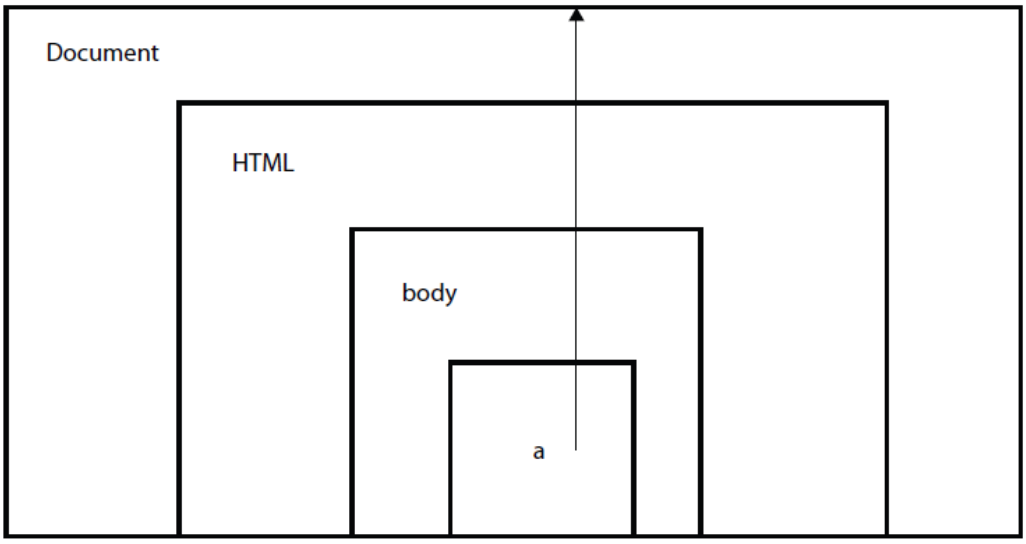
Las principales incompatibilidades que podemos encontrar entre estos modelos son las siguientes:

- La función manejadora del evento recibirá como parámetro un objeto `event` que contendrá todas las propiedades relacionadas con el evento. Este objeto será diferente para IE que para el resto de navegadores, por lo que tendremos que adecuar nuestro código JavaScript para que este objeto sea tratado igual en todos los casos.
- En ocasiones, nos interesa asociar más de un manejador para el mismo evento de un elemento. El nivel 0 del DOM no nos lo permite, si asociáramos dos eventos `onclick` al mismo elemento, el segundo sobrescribiría al primero. Para solucionar esto, en el nivel 2 del DOM aparece un método que nos permite asociar un manejador de evento a un elemento, de manera que, si ya existe un manejador anterior, se añade el nuevo sin eliminar el existente. El problema es que el modelo de eventos de IE utiliza un método distinto del que utilizan el resto de navegadores, por lo que, una vez más, tendremos que adecuar nuestro código JavaScript para que esta gestión se unifique.
- La propagación de los eventos es diferente en IE que en el resto. Cuando tenemos elementos anidados y se produce un evento (por ejemplo, un click del ratón) ¿dónde se debe manejar primero el evento, en el elemento contenedor o en el contenido? Originalmente, Netscape decidió que los eventos se manejaban de fuera hacia dentro, es decir, primero el elemento

contenedor y luego los contenidos (a este tipo de propagación se le llamo "`capturing`"):



Sin embargo, IE decidió que los eventos serían manejados de dentro hacia fuera, primero el contenido y luego el contenedor (a este tipo de propagación se le llamó "`bubbling`"):



De nuevo, si queremos cancelar la propagación del evento deberemos adecuar nuestro código JavaScript para que funcione correctamente en todos los navegadores.

Afortunadamente, jQuery nos abstrae de todo esto, de manera que podemos olvidarnos de estos problemas de compatibilidad entre navegadores.

Eventos en jQuery

En jQuery tenemos varias alternativas para asociar manejadores de eventos a los elementos del DOM: ~~`.bind()`, `.live()`~~ (aunque han sido eliminados en las nuevas versiones de la librería), uno de los muchos métodos de evento, tales como, `.click()`, `.mouseout()`, etc., o los nuevos métodos `.on()` `.off()` introducidos en la versión 1.7 de la librería. Elegiremos una forma u otra en función de nuestras necesidades.

Como en el nivel 2 del DOM, podemos asociar más de un manejador a un evento de un mismo elemento y el nuevo manejador respetará los anteriores. Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>método .bind()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                    type="text/javascript"></script>
    <script>
    $(function()
    {
      $("#aDiv").bind("click", function(event)
      {
        console.log("manejador 1");
      });
      $("#aDiv").bind("click", function(event)
      {
        console.log("manejador 2");
      });
    });
    </script>
</head>
<body>
  <div id="aDiv">Púlsame</div>
</body>
</html>
```

Ejemplo 3: metodoBind.html

Al pulsar sobre el div aparecerán en la consola los dos mensajes (“`manejador 1`” y “`manejador 2`”), ya que el método `.bind()` añadirá el segundo manejador de evento sin eliminar el primero. Como se puede observar, se elimina el prefijo “`on`” para nombrar el evento, en lugar de llamarse “`onclick`”, el evento se llama “`click`”. Este código funcionará exactamente igual para todos los navegadores, puesto que jQuery realizará por nosotros las gestiones necesarias.

El objeto jQuery.Event

El objeto `event` está normalizado. Es del tipo `jQuery.Event` y contendrá las siguientes propiedades, independientemente del navegador en que se ejecute:

- `target`: El elemento DOM que inició el evento.
- `relatedTarget`: El otro elemento DOM envuelto en el evento, si lo hubiera.
- `pageX`: La posición del ratón relativa al borde izquierdo del documento.
- `pageY`: La posición del ratón relativa al borde superior del documento.
- which: Para los eventos de tecla y de ratón, esta propiedad indica la tecla específica o el botón del ratón que fue pulsado.

La lista completa de propiedades que podemos encontrar es la siguiente (aunque algunas de ellas pueden contener el valor `undefined`, según el evento que se haya producido): `altKey, bubbles, button, buttons, cancelable, char, charCode, clientX, clientY, ctrlKey, currentTarget, data, detail, eventPhase, key, keyCode, metaKey, offsetX, offsetY, originalTarget, pageX, pageY, relatedTarget, screenX, screenY, shiftKey, target, toElement, view, which`.

Además, este objeto tiene una serie de métodos, de los cuales vamos a destacar dos:

- El método `event.preventDefault()`: Si llamamos a este método, la acción por defecto del evento no será realizada. Por ejemplo, si ponemos un manejador de evento para el evento click de un enlace, la acción por defecto del enlace al hacer click sobre él será cargar la página a la que apunta su atributo `href`. Si queremos anular esta acción tendremos que llamar al método `preventDefault` del enlace.
- El método `event.stopPropagation()`: Si llamamos a este método se detendrá la propagación del evento en el resto de elementos del árbol DOM, por lo que, si otros elementos tienen asociado un manejador de evento para este mismo evento, este no será ejecutado.

```
<!DOCTYPE html>
<html>
  <head>
    <title>event.preventDefault</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
        {
       $("a:first").click(function (evento)
```

```
        {
                    evento.preventDefault();
                    $(this).text("No voy a ningún sitio");
        });
      });
    </script>
  </head>
<body>
      <a href="http://www.google.es">Google con preventDefault</a><br />
      <a href="http://www.google.es">Google sin preventDefault</a>
</body>
</html>
```

Ejemplo 4: metodoPreventDefault.html

En el ejemplo anterior, al pulsar el primer enlace se cambia el texto del enlace, pero no se carga la página de Google, mientras que en el segundo que tiene el mismo valor en el atributo `href` sí que la carga. Esto es porque en el primer enlace se cancela la acción por defecto por medio del método `.preventDefault()`.

Asociando manejadores de evento con jQuery

Los métodos `.bind()` y `.unbind()`

Como hemos visto en el punto anterior con jQuery podemos utilizar el método `.bind()` para asociar un manejador de evento a un elemento html. Este método recibe tres parámetros: el tipo de evento, los datos que le pasamos al evento (si los hubiera, aunque no es muy habitual) y la función manejadora.

```
<!DOCTYPE html>
<html>
  <head>
    <title>rollover 1</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                    type="text/javascript"></script>
    <script>
    $(function()
    {
      $("#objId").bind("mouseover", function(event)
      {
        $(this).css({
                "text-decoration":"underline",
                "cursor":"pointer"
              });
      }).bind("mouseout", function(event)
      {
        $(this).css({
                "text-decoration":"none"
              });
      });
    });
    </script>
</head>
<body>
```

```
  <a id="objId">No voy a ningún lado</div>
</body>
</html>
```

Ejemplo 5: rollover1.html

En el ejemplo, se selecciona el elemento de id `#objid` y, utilizando encadenamiento, se le asocian funciones manejadoras anónimas a sus eventos `mouseover` y `mouseout`.

El método `.bind()` dispone de un atajo en jQuery. El ejemplo anterior, se puede sustituir por este código:

```
<!DOCTYPE html>
<html>
  <head>
    <title>rollover 2</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script>
    $(function()
    {
      $("#objId").mouseover(function(event)
      {
        $(this).css({
                "text-decoration":"underline",
                "cursor":"pointer"
              });
      }).mouseout(function(event)
      {
        $(this).css({
                "text-decoration":"none"
              });
      });
    });
    </script>
</head>
<body>
  <a id="objId">No voy a ningún lado</div>
</body>
</html>
```

Ejemplo 6: rollover2.html

Cómo se puede observar, se llama a los métodos `.mouseover()` y `.mouseout()` (el mismo nombre del evento), para asociarles un manejador de evento. Ya comentamos en el tema anterior que podemos utilizar métodos de evento para asociar manejadores a los eventos correspondientes.

Evidentemente, también podemos utilizar el método `.bind()` para asociar los distintos manejadores enumerados.

Para eliminar un manejador de evento asociado mediante el método `.bind()`, o uno de sus atajos, utilizamos el método `.unbind()`. Si utilizamos el ejemplo anterior

como referencia, para eliminar el manejador de evento, utilizaremos el siguiente código:

```
$("#objId").unbind("mouseover");//Eliminamos el manejador de evento mouseover
$("#objId").unbind("mouseout"); // Eliminamos el manejador de evento mouseout
```

Si el elemento `#objId` tuviera asociado más de un manejador para el evento `mouseover`, el código anterior los eliminaría todos. Si quisiéramos eliminar uno y mantener el resto, tendríamos que hacer uso de espacios de nombres.

Para asociar el manejador haríamos lo siguiente:

```
$("#objId").bind("mouseover.MisEventos", function(event)
{
  $(this).css("text-decoration", "underline");
}).bind("mouseout.MisEventos", function(event)
{
  $(this).css("text-decoration", "none");
});
```

Y para eliminarlo:

```
$("#objId").unbind("mouseover.MisEventos");
$("#objId").unbind("mouseout.MisEventos");
```

De esta forma, sólo se eliminará el manejador de eventos correspondiente al espacio de nombres `.MisEventos`. Esta técnica, nos permitirá trabajar con distintos manejadores de evento, para los mismos elementos, desde distintos módulos de la aplicación.

Si queremos eliminar todos los manejadores de eventos asociados con un elemento, llamaremos al método `.unbind()` sin parámetros:

```
$("#objId").unbind(); // Eliminamos los manejadores de todos los eventos
```

Se eliminarán los manejadores de los eventos `mouseover` y `mouseout`.

Los métodos ~~`.live()` y `.die()`~~[1] on()

En ocasiones, nos puede interesar asociar manejadores de evento a todos los elementos existentes de un mismo tipo. Por ejemplo a todos los elementos con un determinado estilo de clase. Podemos hacer esto de forma sencilla utilizando el método `.bind()` tal y como hemos visto en el punto anterior. Pero, ¿Qué ocurre si después de asociar el manejador de eventos a todos los elementos de la clase se crean nuevos elementos de esa misma clase? ¿Estos nuevos elementos tendrán el

[1] Los métodos .live() y .die() fueron sustituidos por .on() y .off() a partir de la versión 1.7 de la librería, por lo que, se consideran *deprecated*. En la versión 1.9 se han eliminado y ya no pueden utilizarse. Por este motivo, en el ejemplo que se muestra se utiliza la versión 1.8.3 de la librería.

manejador de eventos asociado o tendremos que asociárselo explícitamente? La respuesta es sencilla, si utilizamos el método `.bind()` el manejador de eventos se asociará con todos los elementos existentes, pero no con los futuros. Si queremos que los elementos nuevos también tengan asociado el manejador, deberemos utilizar el método `.live()`. A nivel de parámetros este método funciona exactamente igual que el anterior. Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>método .live()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.8.3.min.js"
                  type="text/javascript"></script>
    <script>
    $(function()
    {
      $(".enlace").live("click", function(event)
      {
        console.log("Tengo un manejador");
      });
      $("body").append("<a class='enlace'> Nuevo enlace </a>");
    });
    </script>
</head>
<body>
</body>
</html>
```

Ejemplo 7: metodoLive.html

En el ejemplo, aunque el nuevo enlace se crea después de asociar el manejador, al producirse el evento `click` sobre él, se ejecuta el código del mismo.

Al igual que el método `.bind()` dispone de `.unbind()` para eliminar el manejador, el método `.live()` dispone de `.die()` para el mismo cometido.

Los métodos ~~.delegate() y .undelegate()~~ on()

Una de las limitaciones del método `.live()`, es que no puede ser encadenado como el resto de métodos de jQuery. Cuando necesitemos utilizar encadenamiento, usaremos `.delegate()` en lugar de `.live()`, el cual, también asocia un manejador a elementos existentes y futuros.

```
<!DOCTYPE html>
<html>
  <head>
    <title>método .delegate()</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script>
    $(function()
```

```
    {
      $("body").delegate("p", "click", function()
      {
        console.log("AYYY!!!");
      }).css("color", "green");
    });
    </script>
  </head>
<body>
  <p>Púlsame!</p>
</body>
</html>
```

Ejemplo 8: metodoDelegate.html

Cómo vemos en el ejemplo, el primer parámetro del método `.delegate()` es el selector del elemento que recibe el manejador de evento correspondiente. La ventaja de este método, es que podemos encadenarlo con otros métodos jQuery.

De nuevo, al igual que .unbind() y .die(), .delegate() tiene su correspondiente .undelegate().

El método `.one()`

Puede darse el caso, en el que necesitemos que un manejador de evento, se ejecute una sola vez, y no se vuelva a ejecutar. Para esto disponemos del método `.one()`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo método .one()</title>
    <style type="text/css">
    .div1 { width : 100px; height : 100px; background-color: blue; }
      .div2 { width : 100px; height : 100px; background-color: red; }
    </style>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script>
    $(function()
      {
        $(".div1").one("click", function()
        {
            $("body").append("<p>clicked div 1 </p>");
        });
        $(".div2").bind("click", function()
        {
            $("body").append("<p>clicked div 2 </p>");
        });
    });
    </script>
</head>
<body>
  <div class="div1"></div>
  <div class="div2"></div>
</body>
```

```
</html>
```

Ejemplo 9: metodoOne.html

Si ejecutamos el ejemplo, veremos que al pulsar varias veces sobre el segundo `<div>` aparecerá un mensaje por consola para cada click que hagamos. Sin embargo, si hacemos varios clicks sobre el primer `<div>`, sólo aparecerá un mensaje, es decir, la función manejadora del evento sólo se ejecutará la primera vez.

Provocar la ejecución de los manejadores de evento

Con jQuery es muy sencillo ejecutar un manejador de evento que se ha asociado a un elemento. Para ello disponemos del método `.trigger()`. Este método recibe dos parámetros: un string indicando el tipo de evento a ejecutar y, opcionalmente, un array de parámetros adicionales para pasarle al método o un objeto de tipo `jQuery.Event`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>método .trigger()</title>
    <style type="text/css">
    div { padding : 10 10 10 10; width : 100; height : 100; }
    .div1 { background-color : blue; }
    .div2 { background-color : yellow; }
    .div3 { background-color : green; }
    </style>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                    type="text/javascript"></script>
    <script>
       $(function() {
         $("p").hide();
         $(".div1", this).mouseover(function() {
           $(this).find("p").show();
         }).mouseout(function() {
           $(this).find("p").hide();
         });
         $(".div2", this).mouseover(function() {
           $(this).find("p").show();
         }).mouseout(function() {
           $(this).find("p").hide();
         });
         $(".div3", this).mouseover(function() {
           $(this).find("p").show();
         }).mouseout(function() {
           $(this).find("p").hide();
         });
         $("input#tover").click(function() {
           $("div").trigger("mouseover");
         });
         $("input#tout").click(function() {
           $("div").trigger("mouseout");
         });
```

```
      });
    </script>
  </head>
<body>
  <input id="tover" type="button" value="trigger del mouseover"></input>
  <input id="tout" type="button" value="trigger del mouseout"></input>
  <div class="div1">
    <p>Aquí</p>
  </div>
  <div class="div2">
    <p>Aquí</p>
  </div>
  <div class="div3">
    <p>Aquí</p>
  </div>
</body>
</html>
```

Ejemplo 10: metodoTrigger.html

En el ejemplo, al pulsar sobre el primer input se ejecutará el manejador asociado al evento `mouseover` de los div, y al pulsar el sobre el segundo se ejecutará el manejador asociado al evento `mouseout`. Si en lugar de ejecutar todos los manejadores del conjunto de elementos seleccionados, quisiéramos ejecutar sólo el del primer elemento de la selección, podríamos utilizar el método `.triggerHandler()`.

Nueva API de la versión 1.7

A partir de la versión 1.7 jQuery ha hecho un esfuerzo por simplificar el API de gestión de eventos. Aunque podemos seguir utilizando todos los métodos vistos hasta ahora, también disponemos de dos nuevos métodos que unifican todas las posibilidades anteriores. Los nuevos métodos son `.on()` y `.off()`.

Lo bueno del método `.on()` es que reemplaza todas las funcionalidades vistas en este tema con un único método más flexible.

El siguiente ejemplo muestra como asociar dos manejadores de evento a un elemento del DOM ya existente. En este caso, el método `.on()` recibirá dos parámetros: el tipo de evento y la función manejadora. Por lo tanto, la funcionalidad será la misma que si utilizáramos el método `.bind()`.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo método .on()</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script>
    $(function()
    {
```

```
      $("#aDiv").on("click", function()
      {
        console.log("Manejador 1");
      });
      $("#aDiv").on("click", function()
      {
        console.log("Manejador 2");
      });
    });
    </script>
</head>
<body>
  <div id="aDiv">Púlsame!</div>
</body>
</html>
```

Ejemplo 11: metodoOn.html

Al probar el ejemplo anterior, podemos observar cómo se muestran los dos mensajes al hacer click sobre el `<div>`.

El método `.on()` también nos ofrece las mismas funcionalidades que los métodos `.live()` y `.delegate()`, y además lo hace utilizando la misma sintaxis. Para usar `.on()` como `.delegate()` o `.live()`, simplemente añadimos un segundo parámetro opcional en el que indicaremos un selector que representa el elemento al cual le estamos asociando el manejador de evento. En el siguiente ejemplo, `.on()` se usa para recibir los eventos de click en el documento y ejecutar la función indicada si el click se origina en un elemento con id `#enlace`. Como estamos asociando el manejador de evento a través de `document`, funcionará en cualquier lugar en el que aparezca el enlace. Además, funcionará tanto si el enlace existe en el momento de la asociación del manejador como si se crea en un futuro.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo método .on()</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script>
        $(function(){
          $(document).on("click", "#enlace", function(event){
              console.log("Tengo un manejador");
          });
          $("body").append("<a id='enlace'> No voy a ningún sitio </a>")
        });
    </script>
  </head>
<body>
</body>
</html>
```

Ejemplo 12: metodoOn2.html

Podemos reescribir el ejemplo del método `.delegate()` utilizando el método `.on()` de la siguiente forma:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo método .on()</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
    <script>
    $(function()
    {
      $("body").on("click", "p", function()
      {
        console.log('AYYY!!!');
      }).css("color", "green");
    });
    </script>
  </head>
<body>
  <p>Púlsame!</p>
</body>
</html>
```

Ejemplo 13: **metodoOn3.html**

Para eliminar los manejadores de eventos asociados mediante el método `.on()`, utilizamos el método `.off()`. Su funcionamiento es idéntico al de los métodos `.unbind()`, `.die()` o `.undelegate()`. Para eliminar los manejadores asociados en los ejemplos anteriores, utilizaremos el siguiente código:

```
$("#aDiv").off("click");
$(document).off("click", "#anchor");
$("body").off("click", "p");
```

Como se puede observar, el nuevo api de gestión de eventos es más limpio y sencillo, ya que utiliza un único método (variando los parámetros) para todas las posibilidades. Por lo tanto, desde la versión 1.7 de la librería se recomienda la utilización de este api en lugar de las anteriores. Todos los métodos anteriores se siguen manteniendo en la librería a excepción de los métodos `.live()` y `.die()`, que se han eliminado en la versión 1.9.

4. Posición y tamaño de los elementos

Tradicionalmente, en JavaScript obteníamos las propiedades de posición y tamaño de los elementos utilizando css:

```
var pos = $("div").css("left");
pos = parseInt(pos);
$("div").css("left", pos+10);
```

Sin embargo, con jQuery, podemos simplificar este código utilizando una serie de métodos que nos facilitan bastante las cosas. En la siguiente tabla, tenemos un resumen de los mismos.

Método	Funcionamiento
.offset([coordenadas[2]])	Sin parámetros obtiene las coordenadas del primer elemento de la selección. Si recibe un objeto coordenadas, colocará todos los elementos de la selección en esas coordenadas.
.position()	Obtiene las coordenadas del primer elemento de la selección relativas a la posición del padre.
.height([alto]) .innerHeight() .outterHeight([incluirMargen])	El método .height() sin parámetros obtiene el alto del primer elemento de la selección, sin incluir márgenes, padding ni bordes. Si se le pasa un alto todos los elementos de la selección tomarán ese valor en su propiedad height. El método .innerHeight() obtendrá el alto incluyendo el padding, pero no los bordes. Por último, el método .outerHeight() obtendrá el alto incluyendo padding, bordes y, opcionalmente, los márgenes en función del parámetro incluirMargen.
.width([ancho]) .innerWidth() .outerWidth([incluirMargen])	Estos métodos funcionan igual que los anteriores, pero en lugar de trabajar con los altos, lo hacen con los anchos.
.scrollLeft([desplazamiento])	Sin parámetros obtiene la posición del scroll horizontal para el primer elemento de la selección. Si recibe un desplazamiento, desplazará el scroll horizontal de todos los elementos seleccionados al valor recibido.
.scrollTop([desplazamiento])	Funciona igual que el anterior, pero para el scroll vertical.

Tabla 2: Métodos para trabajar con las posiciones y tamaños de los elementos

En el siguiente ejemplo se obtiene la posición de todos los párrafos de la página y se añade al final de los mismos:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo método offset</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <style>
        p { margin-left:10px; }
      </style>
      <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
      <script type="text/javascript">
       $(function()
       {
         $("p").each(function (i)
         {
            var pos = $(this).offset();
```

[2] El objeto coordenadas, deberá tener dos campos: uno llamado left y otro llamado top

```
          $(this).html($(this).html()+" left: "+pos.left+", top: "+pos.top);
        });
      });
      </script>
  </head>
<body>
  <p>Hola</p><p>2do Párrafo</p>
</body>
</html>
```

Ejemplo 14: metodoOffset.html

El anterior ejemplo obtiene la posición de un elemento utilizando el método `.offset()`. Este mismo método se puede emplear para cambiar la posición del elemento. Para ello, tenemos que pasarle un objeto que contenga dos propiedades llamadas `top` y `left`:

```
$("div#box").offset({top:100,left:100});
```

Otra opción sería cambiar la posición del elemento obteniendo primero la posición actual:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo método offset</title>
      <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
      <style>
        p { margin-left:10px; }
      </style>
      <script src="http://code.jquery.com/jquery-3.1.0.min.js"
                  type="text/javascript"></script>
      <script type="text/javascript">
       $(function()
       {
         $("p").click(function ()
         {
           var pos = $(this).offset();
           pos.left += 50;
           $(this).offset(pos);
           $(this).html($(this).text()+" left: "+pos.left+", top: "+pos.top);
         });
         $("p").each(function (i)
         {
           var pos = $(this).offset();
           $(this).html($(this).html()+" left: "+pos.left+", top: "+pos.top);
         });
      });
      </script>
  </head>
<body>
  <p>Hola</p><p>2do Párrafo</p>
</body>
</html>
```

Ejemplo 15: metodoOffset2.html

En el ejemplo anterior se le van sumando 50 píxeles a la posición horizontal de los párrafos y se va añadiendo al texto la posición actual cada vez que se pulsa sobre un párrafo.

Una cosa a tener en cuenta es que este método obtiene la posición del elemento relativa al documento independientemente de que la propiedad position sea abosolute, relative o cualquier otra.

5. Ejercicios

En este tema vamos a seguir ampliando la funcionalidad de nuestro carro de la compra. En concreto, vamos a hacer funcionar la barra de navegación del carrito con los botones de desplazamiento a izquierda, a derecha y el botón vaciar.

Ejercicio 1

En primer lugar, vamos a modificar la forma de introducir los manejadores de evento para el enlace `delete` de los artículos que se introducen en el carrito. Si en el anterior tema, utilizábamos el método `.click()` del enlace para asociar el manejador de evento, ahora vamos a hacer que se asocie el manejador al principio (en el `$(document).ready` simplificado), pero asociándolo para los elementos que se creen en el futuro con la clase `delete`. Esto lo haremos utilizando el método `.on()` visto en este tema.

Ejercicio 2

Otra cosa mejorable del tema anterior es la forma en que gestionábamos el stock disponible. Lo que estamos haciendo es controlar en el manejador del evento `dblclick` de los ítems que el stock disponible es mayor que cero, en cuyo caso no hacemos nada. Pero sería mejor, sobre todo de cara a futuras sesiones, que cuando el stock disponible llegara a cero se eliminara el manejador de evento del ítem en cuestión. Para esto disponemos del método `.unbind()`. Después, si se elimina algún ítem del carro de la compra, y el stock disponible vuelve a ser mayor que cero, volveremos a poner el manejador de evento para ese ítem.

Para hacer esto, lo más sencillo es que nos creemos una función que reciba como parámetro un objeto jQuery con los elementos a los que queremos asociar el manejador de evento. Después, desde nuestro `document.ready` llamaremos a la función de la siguiente forma:

```
establece_evento_dblclick_items($(".item"));
```

y desde la función que elimina el ítem del carro de la compra la llamaremos de la siguiente forma:

```
establece_evento_dblclick_items($(this));
```

A continuación, implementaremos la funcionalidad de los tres botones de nuestra barra de navegación: `btn_clear`, `btn_prev`, `btn_next`.

~~Ejercicio 3~~

~~Al pulsar sobre el botón `btn_clear` el carrito debe vaciarse, pero no sólo eso, también tienen que actualizarse los stock disponibles de los artículos, el número de artículos comprados y el precio total de la compra. Esto puede parecer engorroso, pero si aprovechamos las funcionalidades que ya tenemos implementadas, podemos hacerlo de una forma muy sencilla.~~

~~¿Cómo vaciaríamos el carrito si no existiera el botón vaciar y tuviéramos que hacerlo de forma manual? Tendríamos que pulsar con el ratón sobre el botón `delete` de todos los artículos del carrito ¿y esto actualizaría todos los elementos que hemos comentado? Sí. Pues ya está, lo que tenemos que hacer al pulsar el botón vaciar es lanzar por código un evento de click del ratón sobre el botón `delete` de todos los artículos del carrito. Simplemente, seleccionaremos los elementos `delete` y lanzaremos el evento correspondiente con el método `.trigger()` visto en este tema.~~

Ejercicio 4

Veamos ahora como implementaremos la funcionalidad de movernos por los artículos del carrito con los botones de desplazamiento:

- Lo primero que debemos tener en cuenta es que un artículo en el carrito tiene un ancho de unos 120 píxeles y que en el ancho inicial del carrito caben 4 artículos. Por lo tanto, cuando añadimos un producto al carrito, debemos comprobar si hay más de cuatro artículos añadidos, en cuyo caso, aumentaremos el ancho del contenedor de artículos del carrito (id `cart_items`) en 120 píxeles. Para saber cuántos artículos hay en el carrito podemos consultar la propiedad `length` de los objetos seleccionados de la siguiente forma:

  ```
  var numArticulosCarrito = $("#cart_items").children().length;
  ```

- Cuando pulsamos el botón anterior (id `btn_prev`), debemos desplazar el contenedor de artículos del carrito (id `cart_items`) 50 píxeles a la derecha (`pos.left += 50`).
- Cuando pulsamos el botón siguiente (id `btn_next`), debemos desplazar el contenedor de artículos del carrito (id `cart_items`) 50 píxeles a la izquierda (`pos.left -= 50`).

- Si el contenedor de artículos llega al borde del carrito, por la izquierda o por la derecha, no debemos seguir desplazándolo aunque se siga pulsando el botón de desplazamiento. La forma más sencilla de controlar esto, es guardando en dos variables globales la posición inicial y el ancho inicial del contenedor de artículos. Después, controlaremos que la posición actual después de desplazar el contenedor no sea nunca mayor que la posición inicial (borde izquierdo), y que la posición actual más el ancho actual no sea nunca menor que la posición inicial más el ancho inicial (borde derecho).
- Por último, al eliminar un artículo del carrito, tenemos que comprobar si el número de artículos restante es mayor de 4, y, si es así, disminuir el ancho del carrito 120 píxeles.

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

6. Índice de ejemplos y tablas

Ejemplos

Tablas