Eventos en JQuery

jQuery provee métodos para asociar controladores de eventos (en inglés *event handlers*) a selectores. Cuando un evento ocurre, la función provista es ejecutada. la sintaxis por defecto para un manejador de evento es de la siguiente forma `$.fn.nombreEvento.`

```
  <script>
   $(document).ready(function(){
    $( "#target" ).click(function() {
      alert( "El evento click ha sido disparado" );
    });
   });
  </script>
</head>
<body>
<div id="target">
  Click here
</div>
</body>
</html>
```

Dentro de la función, la palabra clave ***this*** hace referencia al elemento en que el evento ocurre.

```
<body>
<p>First Paragraph</p>
<p>Second Paragraph</p>
<p>Yet one more Paragraph</p>

<script>
$( "p" ).click(function() {
  alert($( this ).text());
});
</script>
</body>
</html>
```

Podemos enlazar eventos entre si para que se ejecuten cuando nos interese.

```
<body>
<form>
  <input id="target" type="text" value="Field 1">
  <input type="text" value="Field 2">
</form>
<div id="other">
  Trigger the handler
</div>
The event handler can be bound to the first input field:
<script>
$( "#target" ).blur(function() {
  alert( "Handler for .blur() called." );
});

$( "#other" ).click(function() {
  $( "#target" ).blur();
});
</script>
</body>
</html>
```

http://api.jquery.com/category/events/.

jQuery ofrece métodos para la mayoría de los eventos (entre ellos `$.fn.click`, `$.fn.focus`, `$.fn.blur`, …). Estos últimos son formas reducidas del método `$.fn.on` de jQuery (`$.fn.bind` en versiones anteriores a jQuery 1.7). El método `$.fn.on` es útil para vincular (en inglés *binding*) la misma función de controlador a múltiples eventos, para cuando se desea proveer información al controlador de evento, cuando se está trabajando con eventos personalizados o cuando se desea pasar un objeto a múltiples eventos y controladores.

```
$(elements).on(events [, selector] [, data], handler);
```

```
<script>
$( "p" ).on( "click", function() {
  alert( $( this ).text() );
});
</script>
```

```
<script>
$( "p" ).on( "click dbclick mouseover", function() {
  alert( $( this ).text() );
});
</script>
```

Se puede asociar cierta información cuando se ejecute el evento en concreto y además se aprecia cómo se ejecutan múltiples eventos asociados a la misma acción.

```
<script>
function greet( event ) {
  alert( "Hello " + event.data.name );
}
$( "#button" ).on( "click", {name: "Karl"}, greet ).on( "click", {name: "Addy"}, greet );
//$( "#button" ).on( "click", {name: "Addy"}, greet );
</script>
</body>
</html>
```

A veces puede necesitar que un controlador particular se ejecute solo una vez y después de eso, necesite que ninguno más se ejecute, o que se ejecute otro diferente. Para este propósito jQuery provee el método `$.fn.one`.

```
<script>
$('p').one('click', function() {
        console.log('Se clickeó al elemento por primera vez');
        $(this).click(function() { console.log('Se ha clickeado nuevamente'); });
    });
</script>
```

Para desvincular (en ingles *unbind*) un controlador de evento, puede utilizar el método `$.fn.off` pasándole el tipo de evento a desconectar. Si se pasó como adjunto al evento una función nombrada, es posible aislar la desconexión de dicha función pasándola como segundo argumento.

```
<script>
var foo = function() { console.log('foo'); };
    var bar = function() { console.log('bar'); };

    $('p').on('click', foo).on('click', bar);
    $('p').off('click', bar);
</script>
```

Muy a menudo, elementos en una aplicación estarán vinculados a múltiples eventos, cada uno con una función diferente. En estos casos, es posible pasar un objeto dentro de `$.fn.on` con uno o más pares de nombres claves/valores. Cada clave será el nombre del evento mientras que cada valor será la función a ejecutar cuando ocurra el evento.

```
<script>
$('p').on({
  'click': function() {
    console.log('clickeado');
  },
    'mouseover': function() {
    console.log('sobrepasado');
  }
});
</script>
```

```
<script>
$('p').on({
  'click': function() {console.log('clickeado');},
  'mouseover': function() {console.log('sobrepasado');}
});
</script>
```

El objeto event

Al igual que ocurre con javascript, la función controladora de eventos recibe un objeto del evento, el cual contiene varios métodos y propiedades, este elemento es el objeto *"event"* y es comúnmente utilizado para prevenir la acción predeterminada del evento a través del método *preventDefault* pero contiene muchas propiedades y metodos utiles.

```
<body>
<div id="button">

<a href="http://jquery.com">default click action is prevented</a>
<div id="log"></div>

<script>
$( "a" ).click(function( event ) {
  event.preventDefault();
  $( "<div>" )
    .append( "default " + event.type + " prevented" )
    .appendTo( "#log" );
});
</script>

</body>
```

http://api.jquery.com/category/events/event-object/

A través del método `$.fn.trigger`, jQuery provee una manera de disparar controladores de eventos sobre algún elemento sin requerir la acción del usuario. Si bien este método tiene sus usos, no debería ser utilizado para simplemente llamar a una función que pueda ser ejecutada con un click del usuario.

```
<script>
$( "div" ).on( "click", function( event, person ) {
  alert( "Hello, " + person.name );
  });
$( "div" ).trigger( "click", { name: "Jim" } );
</script>
```

En su lugar, debería guardar la función que se necesita llamar en una variable, y luego pasar el nombre de la variable cuando realiza el vinculo (*binding*). De esta forma, podrá llamar a la función cuando lo desee en lugar de ejecutar `$.fn.trigger`.

```
<script>
var foo = function(e) {
  if (e) {
    console.log(e);
  } else {
    console.log('esta ejecucción no provino desde un evento');
  }
};
$('p').click(foo);
foo();
</script>
```

Los eventos personalizados ofrecen una nueva forma de pensar la programación en JavaScript. En lugar de enfocarse en el elemento que ejecuta una acción, los eventos personalizados ponen la atención en el elemento en donde la acción va a ocurrir. Este concepto brinda varios beneficios:

```
<script>
$(document).on('myCustomEvent', { foo : 'bar' }, function(e, arg1, arg2) {
  console.log(e.data.foo); // 'bar'
  console.log(arg1); // 'bim'
  console.log(arg2); // 'baz'
});
$(document).trigger('myCustomEvent', [ 'bim', 'baz' ]);
</script>
```

- los comportamientos del elemento objetivo pueden ser ejecutados por diferentes elementos utilizando el mismo código;
- los comportamientos pueden ser ejecutados en múltiples, similares elementos objetivos a la vez;
- los comportamientos son asociados de forma más clara con el elemento objetivo, haciendo que el código sea más fácil de leer y mantener.

Un ejemplo es la mejor forma de explicar el asunto. Suponga que posee una lámpara incandescente en una habitación de una casa. La lámpara actualmente está encendida. La misma es controlada por dos interruptores de tres posiciones y un *clapper* (interruptor activado por aplausos):

```
<div class="room" id="kitchen">
    <div class="lightbulb on"></div>
    <div class="switch"></div>
    <div class="switch"></div>
    <div class="clapper"></div>
</div>
```

Sin la utilización de eventos personalizados, es posible escribir la rutina de la siguiente manera:

```
<script>
$('.switch, .clapper').click(function() {
  var $light = $(this).parent().find('.lightbulb');
  if ($light.hasClass('on')) {
    $light.removeClass('on').addClass('off');
  } else {
    $light.removeClass('off').addClass('on');
  }
});
</script>
```

Por otro lado, utilizando eventos personalizados, el código queda así:

```
<script>
$('.lightbulb').on('changeState', function(e) {
  var $light = $(this);
  if ($light.hasClass('on')) {
    $light.removeClass('on').addClass('off');
  } else {
    $light.removeClass('off').addClass('on');
  }
});
$('.switch, .clapper').click(function() {
  $(this).parent().find('.lightbulb').trigger('changeState');
});
</script>
```

Delegación de eventos

Cuando trabaje con jQuery, frecuentemente añadirá nuevos elementos a la página, y cuando lo haga, necesitará vincular eventos a dichos elementos. En lugar de repetir la tarea cada vez que se añade un elemento, es posible utilizar la delegación de eventos para hacerlo. Con ella, podrá enlazar un evento a un elemento contenedor, y luego, cuando el evento ocurra, podrá ver en que elemento sucede.

La delegación de eventos posee algunos beneficios, incluso si no se tiene pensando añadir más elementos a la página. El tiempo requerido para enlazar controladores de eventos a cientos de elementos no es un trabajo trivial. A partir de la versión 1.4.2, se introdujo `$.fn.delegate`, sin embargo a partir de la versión 1.7 es preferible utilizar el evento `$.fn.on` para la delegación de eventos.

```
<script>
$('#myUnorderedList').on('click', 'li', function(e) {
  var $myListItem = $(this);
});
</script>
```

Si necesita remover eventos delegados, no puede hacerlo simplemente desvinculándolos. Para eso, utilice el método `$.fn.off` para eventos conectados con `$.fn.on`, y `$.fn.undelegate` para eventos conectados con `$.fn.delegate`. Al igual que cuando se realiza un vinculo, opcionalmente, se puede pasar el nombre de una función vinculada.

```
<script>
$('#myUnorderedList').off('click', 'li');
</script>
```