ceviche.js

Capítulo 7: jQuery

Si bien ya sabemos cómo manejar el DOM, necesitamos que nuestro sitio web funcione bien en diferentes navegadores por igual. jQuery está diseñado desde sus inicios para dar soporte al manejo del DOM en todos los navegadores conocidos, simplificando drásticamente el desarrollo de un sitio web, resolviendo uno de los más grandes problemas en el desarrollo web: El código *cross-browser*. Hace muchos años, se tenía que crear dos versiones del mismo código: una para Netscape y otra para Internet Explorer. Cuando Netscape desapareció y apareció Firefox, se dio el mismo caso, una vez más con Internet Explorer del otro lado. Si a eso le sumamos otros navegadores, como Opera o Safari (para Mac OS), el código crece rápidamente.

jQuery ofrece una serie de métodos para manipular el DOM, manejar eventos y realizar llamadas asíncronas, de tal forma que todo funcione de la misma manera en todos los navegadores.

Para utilizar jQuery en un sitio web debemos ir a la sección *Download* y elegir una de las versiones que ofrece jQuery. Cabe resaltar que jQuery está dando soporte a dos versiones: la 1.x y la 2.x. La diferencia entre ambas es que la 2.x ya no tiene soporte para Internet Explorer 6, 7 y 8 (haciendo que la biblioteca pese bastante menos); así que elegir entre una y otra versión depende del soporte que quieras para tu sitio o aplicación.

En este caso, elegimos la versión 1.11.1 en su versión para desarrollo:

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
```

```
<body>
  <script src="http://code.jquery.com/jquery-1.11.1.js"></script>
</body>
</html>
```

Para cargar un archivo JavaScript utilizamos la etiqueta `<script>`, poniendo la dirección del archivo en el atributo `src`. En algunos casos la etiqueta `<script>` estará dentro de la etiqueta `<head>`, pero en otros casos puede estar al final de la etiqueta `<body>`. ¿Por qué pasa esto?

Los navegadores leen un documento HTML y lo muestran de forma visual en un proceso que es llamado **renderizado**. En este proceso, que puede variar un poco entre navegadores, el navegador lee el documento HTML y lo va interpretando progresivamente, lo cual quiere decir que, por cada parte que lee, verifica si existe algún recurso que debe ser cargado (puede ser una imagen, un iframe, una hoja de estilos o un archivo JavaScript).

Este proceso va de inicio a fin, por lo que, si las etiquetas `<script>` se ponen dentro de la etiqueta `<head>`, el navegador va a esperar a que terminen de cargar los archivos JavaScript para seguir leyendo el resto del documento. Esto puede ser contraproducente en la mayoría de casos, por lo que se recomienda poner las etiquetas `<script>` al final de la etiqueta `<body>`, de esta forma todo el documento cargará y se mostrará en la pantalla de una forma más rápida.

---

jQuery tiene una función del mismo nombre, pero se utiliza comúnmente un alias: `$`. Esta función acepta diferentes parámetros:

- Un selector (por ejemplo: `body`, o `#elemento_1`). Puede aceptar como segundo parámetro un nodo elemento de *contexto*, para limitar la búsqueda del selector.
- Una cadena conteniendo HTML (por ejemplo: `<p></p>`), para crear nodos elementos de una manera más rápida. Puede aceptar un segundo parámetro, el cual servirá como nodo documento, que es donde se agregará el o los elementos a crear.
- Un elemento, un arreglo o una lista de nodos elementos (por ejemplo:

`document.querySelectorAll('a')`).

Usando la función `$` con cualquiera de los 3 parámetros se devuelve un objeto instancia de jQuery. Esta instancia es parecida a un arreglo, y tiene diferentes métodos para manejar sus elementos.

Adicionalmente, `$` puede aceptar una función, la cual se ejecutará cuando todo el documento ha terminado de cargar.

Selectores

jQuery permite obtener los elementos del DOM mediante selectores, de la misma forma como lo hace el método `querySelectorAll`, con la diferencia que también acepta selectores propios:

Atributos

- `[name!="value"]`: Devuelve todos los elementos cuyo atributo de nombre `name` **no** tiene el valor `value`

Básico

- `:animated`: Devuelve los elementos que están siendo animados en ese instante.
- `:eq(index)`: Devuelve el elemento que se encuentra en el índice seleccionado, dentro de un conjunto de elementos.
- `:even`: Devuelve los elementos cuyos índices sean pares, teniendo en cuenta que el índice empieza en `0`, por lo que selecciona los elementos en los índices `0`, `2`, `4` y sucesivos.
- `:first`: Devuelve el primer elemento de un conjunto de elementos.
- `:gt(index)`: Devuelve los elementos cuyos índices sean mayores al índice seleccionado.
- `:header`: Devuelve todos los elementos que sean `h1`, `h2`, `h3` y similares.
- `:last`: Devuelve el último elemento de un conjunto de elementos.

- `:lt(index)`: Devuelve los elementos cuyos índices sean menores al índice seleccionado.
- `:odd`: Devuelve los elementos cuyos índices sean impares, teniendo en cuenta que el índice empieza en `0`, por lo que selecciona los elementos en los índices `1`, `3`, `5` y sucesivos.

Contenido

- `:has(selector)`: Devuelve todos los elementos que contienen los elementos definidos en el segundo selector.
- `:parent`: Devuelve todos los elementos que tienen al menos un nodo hijo (ya sea elemento o no).

Formularios

- `:button`: Devuelve los elementos que sean botones, ya sean elementos `<button>` o `<input type="button">`
- `:checkbox`: Devuelve todos los elementos que son `<input type="checkbox">`
- `:file`: Devuelve todos los elementos que son `<input type="file">`
- `:image`: Devuelve todos los elementos que son `<input type="image">`
- `:input`: Devuelve todos los elementos que son `<input>`, `<textarea>`, `<select>` y `<button>`
- `:password`: Devuelve todos los elementos que son `<input type="password">`
- `:radio`: Devuelve todos los elementos que son `<input type="radio">`
- `:reset`: Devuelve todos los elementos que son `<input type="reset">`
- `:selected`: Devuelve el elemento `<option>` seleccionado para un elemento `<select>`
- `:submit`: Devuelve todos los elementos que son `<input type="submit">`
- `:text`: Devuelve todos los elementos que son `<input type="text">`

Visibilidad

- `:hidden`: Devuelve todos los elementos ocultos, los cuales pueden ser: por tener `display: none` en sus estilos, ser elementos `<input type="hidden">`, tener `width` y `height` en 0, o si tiene algún elemento ancestro oculto.
- `:visible`: Devuelve todos los elementos que son visibles. En jQuery, un elemento es considerado visible si ocupa espacio en la pantalla, por lo que elementos con `visibility: hidden` u `opacity: 0` en sus estilos son considerados elementos visibles.

Si vemos el ejemplo usado en el capítulo 3, podremos cambiar el siguiente código:

```
var container = dom('#background');
container.delegate('transitionend', '.slide.current', function(e) {
  var current = dom(e.target),
      next = current.next();

  current.removeClass('current');

  if (current.isLastSibling()) {
    next = current.firstSibling();
  }

  next.addClass('current');
});
```

por:

```
var container = $('#background');
container.on('transitionend', '.slide.current', function(e) {
  var current = $(e.target),
      next = current.next();

  current.removeClass('current');
```

```
    if (current.is(':last-child')) {
      next = current.siblings().first();
    }

    next.addClass('current');
  });
```

Cuando diseñamos `dom.js` tuvimos en mente algunos métodos que maneja jQuery (como `next`, `addClass` y `removeClass`), por lo que el código es bastante similar. Sin embargo, en jQuery no tenemos `isLastSibling` ni `firstSibling`.

En el primer caso, cambiamos `isLastSibling()` por `is(':last-child')`. `is` es un método que permite comparar entre el *set* de elementos seleccionado y un selector (el cual puede ser de CSS o uno de los descrito al inicio del capítulo). En el segundo caso, reemplazamos `firstSibling()` por `siblings().first()`, donde `siblings` es un método que devuelve todos los nodos *hermanos* del nodo seleccionado (pero **no incluye al nodo seleccionado en el resultado**), y `first`, que devuelve el primer elemento de un *set* de nodos en jQuery.

Eventos

jQuery permite manejar eventos, tanto del navegador como propios, utilizando los métodos `on` y `off` (para agregar y eliminar *listeners*, respectivamente). Estos métodos funcionan de la misma manera para eventos del navegador y propios, e incluso se pueden lanzar (o *disparar*) manualmente utilizando el método `trigger`.

Cabe recordar que jQuery agrega *listeners* a los eventos en la *bubbling phase*, y no en la *capture phase*. Esto es importante a tener en cuenta, dada la diferencia que existe entre agregar un *listener* en cualquiera de las dos fases.

Por otro lado, jQuery utiliza *event delegation*, el cual permite definir eventos en elementos que aún no han sido creados, así como definir el mismo evento a un

conjunto de elementos, sin la necesidad de crear un *listener* por cada elemento.

Volvamos al ejemplo de la sección anterior:

```
var container = $('#background');
container.on('transitionend', '.slide.current', function(e) {
  var current = $(e.target),
      next = current.next();

  current.removeClass('current');

  if (current.is(':last-child')) {
    next = current.siblings().first();
  }

  next.addClass('current');
});
```

En este caso, seguimos usando *event delegation*. Sabemos que `$('#background')` devolverá un elemento, y que este, a su vez, tiene elementos hijo cuyas clases son `slide` y también serán `current`, y necesitamos agregar lanzar un evento `transitionend` para cada elemento `.slide.current` (es decir, el elemento `.slide` visible). En `dom.js` se llamaba `delegate`, pero en jQuery toma el nombre de `on`.

Ya sabiendo cómo usar jQuery, podemos terminar el sitio web de **La Buena Espina**. Agreguemos un evento `hashchange` a `window`:

```
$(window).on('hashchange', function(e) {
  $('.panel.current').removeClass('current');

  if (location.hash !== '') {
    $('.panel' + location.hash).addClass('current');
  }
```

```
});
```

De esta forma, cada vez que naveguemos por la barra de navegación, aparecerá el contenido correcto.

Sin embargo, ocurre un *bug* si recargamos la página y tenemos el hash `#historia` en la dirección: **El contenido de Historia no aparece**. Recordemos que al usar el evento `hashchange`, la función del *listener* solo se ejecutará cuando el *hash* cambie, así que necesitamos agregar un evento más a `window`. jQuery permite agregar un listener a más de un evento utilizando `on` una sola vez:

```
$(window).on('hashchange load', function(e) {
  $('.panel.current').removeClass('current');

  if (location.hash !== '') {
    $('.panel' + location.hash).addClass('current');
  }
});
```

De esta forma, nuestro código se ejecutará tanto al cambiar el *hash* en la barra de direcciones, como al cargar la ventana. Podemos ver el código funcionando en http://cevichejs.com/files/7-jquery/index.html

---

Ajax

Además de manejar operaciones en el DOM, jQuery es capaz de manejar operaciones asíncronas. jQuery utiliza `XMLHttpRequest` o `ActiveXObject`, según sea el caso (por ejemplo, en versiones de Internet Explorer donde existe `ActiveXObject`, se utiliza este).

Para poder realizar operaciones asíncronas, jQuery ofrece una serie de métodos, los cuales van desde el básico `$.ajax` hasta `$.get` o `$.post`.

En el capítulo anterior vimos cómo realizar llamadas asíncronas a un servidor.

Utilizamos `http://coffeemaker.herokuapp.com` para probar con el siguiente código:

```
var xhr = new XMLHttpRequest();

var url = 'http://coffeemaker.herokuapp.com
/twitter.json?q=ceviche';

xhr.open('GET', url, true);

xhr.addEventListener('error', function(e) {
  console.log('Un error ocurrió', e);
});

xhr.addEventListener('readystatechange', function() {
  console.log('xhr.readyState:', xhr.readyState);
});

xhr.send();
```

Luego, creamos `xhr.js`, que simplificaba todo el código anterior a:

```
var request = xhr({
  url: 'http://coffeemaker.herokuapp.com/twitter.json?q=ceviche',
  method: 'GET'
});
```

Una de las ventajas de jQuery es que permite tomar el código anterior y convertirlo a una sola línea:

```
var request = $.get('http://coffeemaker.herokuapp.com
/twitter.json?q=ceviche');
```

Actualmente, jQuery tiene soporte para promesas, por lo que podemos usarlo de la siguiente forma:

```
var request = $.get('http://coffeemaker.herokuapp.com
/twitter.json?q=ceviche');

request.then(function(data) {
  console.log(data.length + ' elementos');

  return data;
}).then(function(data) {
  var newPromiseValue = data[0];

  console.log('Primer elemento: ', newPromiseValue);

  return newPromiseValue;
}).then(function(data) {
  var newPromiseValue = data.id;
  console.log('ID del primer elemento: ', newPromiseValue);
});
```

Por otro lado, una buena práctica sería separar la dirección de la cadena de búsqueda, y pasar los parámetros de búsqueda como un objeto:

```
var request = $.get('http://coffeemaker.herokuapp.com
/twitter.json', { q: 'ceviche' });
```

jQuery se encargará de generar la URL antes de enviar la petición, pero ganamos flexibilidad si deseamos cambiar la dirección de la petición.

En la primera parte vimos cómo mostrar las diferentes secciones del sitio web de **La Buena Espina**, excepto una: el formulario de contacto.

Para poder hacer funcionar el formulario de contacto usaremos `$.post`:

```
var contactForm = $('#contact-form'),
    contactName = $('#contacto_nombre'),
    contactMessage = $('#contacto_mensaje');

contactForm.on('submit', function(e) {
  e.preventDefault();

  window.localStorage.setItem('contact-form',
contactMessage.val());

  var xhr = $.post('http://coffeemaker.herokuapp.com/form',
contactForm);

  xhr.then(function() {
    alert('¡Gracias por contactarnos!');
  });

  xhr.then(function() {
    contactMessage.val('');
    contactName.val('');
    window.localStorage.removeItem('contact-form');
  });
});
```

Una de las ventajas de `$.post` es que podemos pasarle un objeto jQuery, y este se **serializará** automáticamente, para obtener todos los valores de los elementos de formulario dentro del mismo objeto que tengan atributo `name`.

También utilizamos la API de *Web Storage* en este código. Recordemos el primer párrafo de esta API del navegador:

> *Empecemos con una API simple de usar pero que soluciona un problema común al trabajar con una aplicación web: El dueño de La Buena Espina quiere un formulario de contacto para que los comensales puedan dar sus impresiones sobre el servicio y la comida. Pero, ¿qué pasaría si luego de enviar el formulario se pierde la conexión, el usuario cierra su navegador o*

> *el servidor no responde? Los comentarios no llegarán al dueño y se pueden perder buenas críticas con respecto al restaurante.*

De esta forma, nos aseguramos que el mensaje del usuario no se pierda si es que existe un error al momento de enviar el mensaje.

Plugins

Una de las ventajas de jQuery es la comunidad que tiene detrás, creada en buena parte gracias a los *plugins* que permite crear. Un *plugin* en jQuery es, básicamente, un método agregado al *prototype* de la función `jQuery` al cual se puede acceder mediante la propiedad `jQuery.fn` (o `$.fn`).

De nuevo, en el ejemplo de las dos primeras secciones, tenemos:

```
var container = $('#background');
container.on('transitionend', '.slide.current', function(e) {
  var current = $(e.target),
      next = current.next();

  current.removeClass('current');

  if (current.is(':last-child')) {
    next = current.siblings().first();
  }

  next.addClass('current');
});
```

En `dom.js` teníamos `isLastSibling` y `firstSibling`, pero en jQuery no. Sin embargo, podemos extender el *prototype* de jQuery y agregar estos métodos:

```
$.fn.isLastSibling = function() {
  return $(this).is(':last-child');
```

```
}

$.fn.firstSibling = function() {
  return $(this).siblings().first();
}
```

Y, de esta forma, tendríamos el siguiente código, más entendible:

```
var container = $('#background');
container.on('transitionend', '.slide.current', function(e) {
  var current = $(e.target),
      next = current.next();

  current.removeClass('current');

  if (current.isLastSibling()) {
    next = current.firstSibling();
  }

  next.addClass('current');
});
```

← 6. Capítulo 6: Peticiones asíncronas

8. Capítulo 8: Mejorando el flujo de trabajo →

Acerca de | Disponible en Amazon.com