Desarrollador Frontend

clase 5 | jQuery

Librerías javascript

¿Qué es una librería* javascript?

- Es un conjunto de objetos/funciones/métodos pensados para poder ser reutilizados al vincularse a una aplicación
- Podemos pensarlo como una caja de herramientas
- Muchas librerías fueron pensadas para simplificar tareas repetitivas
- Por más que exponga determinada interfaz de programación (API), no deja de ser javascript
- Necesitan luego de un código que implemente la funcionalidad
- Podemos combinar múltiples librerías en nuestros proyectos

* si bien la traducción correcta del inglés de *library* sería biblioteca, se ha popularizado el término librería en castellano

Incluir una librería en mi proyecto

Como dijimos, una librería sólo es código javascript, por lo que lo único que necesitamos es un tag `<script>` e incorporarlo...

- "pegando" todo el código dentro del tag script *
- vincular el `<script>` a un archivo físico en mi servidor (esto es, bajando la librería, por ejemplo, jquery.min.js)
- vincular el `<script>` a un CDN o hosted library **

* una estrategia que ahorra un pedido al servidor, puede ser útil en contextos especiales, similar a lo que ocurre con los CSS en <style>

** CDN en wikipedia, Google hosted libraries. El beneficio de utilizar este tipo de scripts es ganar más velocidad en la entrega de contenidos y reutilizar el caché del cliente, ya que no se repite el llamado al mismo CDN cuando varios proyectos apuntan a la misma URL. Es recomendable tener una versión backup de la librería para probar código de forma local sin necesidad de depender de internet.

- jQuery es una librería javascript*
- Con el tiempo se convirtió en una de las librerías más populares
- **No** es un lenguaje en sí mismo
- En su origen apunta a codificar una vez para todos los navegadores
- También simplifica las tareas de manipulación DOM y animación
- Provee un manejo de eventos unificado
- De allí surge el slogan *write less, do more*
- Existen otros proyectos similares (como mootools)
- Existen desarrolladores que cuestionan el sobreuso de jQuery

* ¿Framework o librería? Pueden encontrarse numerosos debates sobre el límite de estos dos conceptos. En el sitio oficial se presenta al proyecto como una librería, acorde a esta idea de caja de herramientas. Un framework, aparte de incluir herramientas, impone una manera de trabajar y de estructurar los proyectos.

Incluir jQuery en mi proyecto

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="style.css">
    <title>jQuery</title>
  </head>

  <body>
    <h1>HTML</h1>

    <!-- primero incluímos las dependencias -->
    <script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
    <!-- luego la implementación -->
    <script>
      $('h1').html('jQuery está en la casa');
    </script>
  </body>
</html>
```

En la sección descargas de jquery.com podemos encontrar las diversas formas de incluir la librería en nuestro proyecto.

¿jQuery 1 o 2?
A partir de la versión 2.x la librería deja de dar soporte a IE8< para evitar sobrecarga de código en la compatibilidad. Los navegadores modernos pueden trabajar de la misma forma tanto con 2.x como con 1.x

Nota: En muchos ejemplos de CDN o hosted libraries encontrarán que las URLs carecen de protocolo (http: o https:). Es importante saber que si queremos probar el siguiente código sin utilizar un webserver, habrá que agregar el protocolo o el navegador no podrá encontrar el archivo.

El siguiente ejemplo manipulará al H1 cambiando su contenido (similar a innerHTML)

La función jQuery

jQuery y su alias $

En nuestro primer ejemplo utilizamos `$('h1')` para seleccionar al titular. Una vez incluída la librería, tenemos acceso en forma global (esto es, en **window**) a los objetos `jQuery` y `$`

El objeto $ es un *alias* de jQuery, lo que quiere decir que ambos refieren al mismo objeto en memoria. La siguiente expresión será evaluada como **true**:

```
$ === jQuery
```

Generalmente los ejemplos e implementaciones que encontramos utilizarán el signo $.

En caso de que $ sea tomado posteriormente por otra librería o declaración, siempre tenemos acceso a jQuery.

El objeto jQuery

Al utilizar la función jQuery o $, generalmente obtenemos como resultado un objeto jQuery.

Un objeto jQuery contiene una colección de elementos DOM que fueron creados a partir de un String u obtenidos de un documento.

Como generalmente se utiliza un selector CSS para acceder a los elementos, a esta colección se la suele llamar un grupo de “elementos seleccionados” o “coincidencias”.

El objeto jQuery se comporta de forma similar a un array; tiene una propiedad **length** y podemos acceder a sus elementos mediante índices. Debemos notar que **no es** un Array javascript, por lo que no tiene funciones como **join** por ejemplo.

jQuery vs API DOM

¿$ equivale a `document.querySelectorAll`?

Al ver un ejemplo de código, podemos pensar que es una versión abreviada de `document.querySelectorAll`, aunque esto es incorrecto.

Lo que devuelve el método `document.querySelectorAll` es un NodeList de javascript (un Array Like).

La función $ devuelve un objeto jQuery, que contiene dentro a los elementos DOM correspondientes **más** las funcionalidades de la librería.

Si queremos verificarlo, la siguiente expresión dará falso:

```
$('h1').toString() ===
document.querySelectorAll('h1').toString()
```

Formas de utilizar la función $

Seleccionar elementos utilizando un selector CSS:

```
var $elementos = $('p.importante strong');
```

Utilizar un selector CSS filtrado:

```
var contexto = document.querySelector('#hola');
var $elementos = $('p.importante', contexto);
```

Convertir a objeto jQuery a cualquier elemento DOM:

```
var seleccion = document.querySelectorAll('p');
var $elementos = $(seleccion);
```

Crear un objeto jQuery a través de un String HTML:

```
var $div = $('<div class="importante">Hola</div>');
```

Crear un objeto jQuery a través de un objeto javascript:

```
var $helper = $({id:12, precio: 2039});
```

Funciones básicas del objeto jQuery

Manipulación DOM de contenidos

- `html()`
  Devuelve el contenido HTML del primer elemento de la colección
- `html('Un poco de <em>markup</em>')`
  Asigna el nuevo HTML a **todos** los elementos de la colección
- `text()`
  Devuelve el contenido textual de **todos** los elementos de la colección concatenados
- `text('<em> se utiliza para dar enfasis')`
  Asigna el contenido textual a **todos** los elementos (utlizando entidades HTML para que el < se vea como caracter)
- `empty()`
  Equivale a html('')

Manipulación DOM de atributos

- `css('color')`
  Devuelve el valor de la propiedad CSS del primer elemento de la colección
- `css('color', 'red')`
  Asigna a **todos** los elementos el estilo *inline* color: red
- `css({})`
  Asigna a **todos** los elementos de la colección las propiedades de un objeto JSON como estilos inline
- `attr('src')`
  Devuelve el valor del atributo src del 1er elemento.
- `attr('src', 'http://dominio.com/imagen.jpg')`
  Asigna a **todos** los elementos de la colección un nuevo valor en el atributo src
- `prop()`
  En el caso de propiedades que no sean atributos HTML (por ejemplo la propiedad **nodeName** de todo elemento DOM, o una propiedad de un objeto) debemos usar prop

Funciones básicas del objeto jQuery

Manipulación DOM de clases CSS

- `addClass('importante especial')`
  Asigna las clases importante y especial a todos los elementos de la colección
- `removeClass('importante')`
  Elimina la clase de los elementos de la colección que la tengan asignada.
- `toggleClass('importante')`
  Los elementos que no tengan la clase importante asignada la tendrán, y será removida de los que la tengan.

Como podemos ver, es similar a la API DOM provista por `classList.add` en navegadores modernos, la diferencia es que jQuery 1.x implementará correctamente en cada navegador.

Lo mismo sucede con los demás métodos.

Manipulación DOM estructural

- `after()`
  Inyecta uno o varios elementos luego de **cada** elemento de la colección
- `append()`
  Cada elemento de la colección adopta al final de sus hijos al/los elemento/s
- `before()`
  Inyecta uno o varios elementos antes de **cada** elemento de la colección
- `prepend()`
  Cada elemento de la colección adopta previo a sus hijos al/los elemento/s
- `remove()`
  Elimina a los elementos de la colección del DOM
- `parent()`
  Obtiene el padre del primer elemento de la colección

Manejo de eventos

Eventos más utilizados
Contamos con métodos abreviados para la mayoría de los eventos más usados, algunos ejemplos:

- `click(callback)`
- `change(callback)`
- `blur(callback)`
- `focus(callback)`
- `submit(callback)`
- `hover(callback)`
- `hover(callbackIn, callbackOut)`

[Listado completo de eventos]

Eventos masivos
Al asignarse un evento a un objeto jQuery, automáticamente lo estamos registrando para **todos** los elementos de la colección.

Crear y disparar eventos
También podemos utilizar la forma larga (que permite suscribirse a eventos personalizados) y dispararlos programáticamente:

- `on('evento', callback)`
- `trigger('evento')`

Si queremos realizar eventos personalizados o dispararlos utilizando javascript puro, hay que tener en cuenta consideraciones de compatibilidad entre navegadores

Referencia a this dentro de los callbacks
Cualquier callback disparado por un evento jQuery dispone de una referencia al elemento que disparó el evento en la variable **this**

$(document).ready

El evento DOM ready
jQuery incluye una implementación para saber cuando el DOM fue interpretado por el navegador. Este evento **no** puede ser reproducido fácilmente en navegadores <= IE8.
La idea es la de disponer dentro del callback de todos los elementos del DOM.
Esto permite que podamos incluir el siguiente fragmento de código incluso en el <head>

```
$(document).ready(init);
function init(){
  //dentro del callback ya puedo acceder a todos los elementos que estén en el DOM
  $('#elemento').html('Funciona!');
}
```

Suscribirse cuando el DOM ya fue cargado
Si suscribimos una función al evento ready y el documento ya se había inicializado, el callback se ejecutará inmediatamente.

Versión abreviada
Si pasamos directamente una función como parámetro a la función jQuery, esta será ejecutada como callback del evento ready

```
$(init);
function init(){
  $('#elemento').html('Funciona!');
}
```

Evento javascript en navegadores modernos
Actualmente (IE9+) puede utilizarse el siguiente evento:
`document.addEventListener('DOMContentLoaded', callback)`

Diferencia con body.load y window.load
Los eventos load de window o body se disparan no sólo cuando fueron interpretados los elementos del DOM, si no cuando todas las imágenes y recursos fueron cargados.

Animaciones simples

Mostrar y ocultar elementos de un objeto jQuery
Con el conjunto de funciones básicas de jQuery disponemos de una serie de animaciones

- `show(milliseconds)`
- `hide(milliseconds)`
- `toggle(milliseconds)`
- `fadeIn(milliseconds)`
- `fadeOut(milliseconds)`
- `fadeToggle(milliseconds)`
- `slideUp(milliseconds)`
- `slideDown(milliseconds)`
- `slideToggle(milliseconds)`

Documentación completa sobre efectos

Esperar
Utilizando el método `delay(milliseconds)` podemos obligar al objeto a no hacer nada durante un tiempo.

Encadenamiento
Si a un conjunto de elementos le asignamos varias operaciones de animación, estas se ejecutan en secuencia, por ejemplo:

```
$('.caja').hide().delay(1000).fadeIn(500)
```

En este caso todos los elementos clase “caja” primero se ocultarán (al no pasar tiempo se realiza inmediatamente), esperarán un segundo y luego en 500ms se mostrarán.

Cancelar la secuencia
Utilizando el método `stop()` podemos detener cualquier animación en curso que tenga asignada el objeto.

Animar propiedades CSS
Utilizando el método `animate({}, ms)` podemos pasar un objeto JSON con todas las propiedades CSS que queremos transformar. Una buena práctica actual es recaer en animaciones CSS3 para esto, y utilizar jQuery sólo para asignar las clases correspondientes.

Soporte de XHR, JSONP y métodos abreviados

El método $.ajax
Este método es el más completo y nos permite realizar un pedido asincrónico (no siempre ajax, ya que también soporta JSONP) un ejemplo de su uso:

```
$.ajax({
  url: 'mi-archivo.html',
}).done(recibir).fail(manejarError);

function recibir(datos){
  $('#miDiv').html(datos);
}
```

El método ajax puede recibir numerosos parámetros en el objeto JSON que se pasa como parámetro

Actualmente se recomienda el uso de `done()` y `fail()` en vez de los parámetros `success` y `error` que pueden encontrarse en ejemplos online desactualizados.
Documentación completa sobre ajax

El método load()
El ejemplo anterior se puede reescribir de la siguiente manera

```
$('#miDiv').load('mi-archivo.html');
```

.load ejecutará un pedido GET y cargará el contenido del recurso dentro de los elementos que contenga el objeto jQuery.

Métodos abreviados get() y post()
Son versiones abreviadas para evitar construir el objeto de opciones JSON en $.ajax

getScript()
Este método utiliza la estrategia de creación dinámica de <script> para cargar contenido salteando el CORS.

getJSON()
jQuery validará el JSON cargado y lo pasará por JSON.parse antes de entregar el resultado al callback.

Documentación completa sobre métodos abreviados