Refinamiento de selecciones

$('div.foo').has('p'); // el elemento div.foo contiene elementos <p>

$('h1').not('.bar'); // el elemento h1 no posse la clase 'bar'

$('ul li').filter('.current'); // un item de una lista desordenada

// que posse la clase 'current'

$('ul li').first(); // el primer item de una lista desordenada

$('ul li').eq(5); // el sexto item de una lista desordenada

/\*\* seudo selectores \*\*/

:button

Selecciona elementos <button> y con el atributo type='button'

:checkbox

Selecciona elementos <input> con el atributo type='checkbox'

:checked

Selecciona elementos <input> del tipo checkbox seleccionados

:disabled

Selecciona elementos del formulario que están deshabitados

:enabled

Selecciona elementos del formulario que están habilitados

:file

Selecciona elementos <input> con el atributo type='file'

:image

Selecciona elementos <input> con el atributo type='image'

:input

Selecciona elementos <input>, <textarea> y <select>

:password

Selecciona elementos <input> con el atributo type='password'

:radio

Selecciona elementos <input> con el atributo type='radio'

:reset

Selecciona elementos <input> con el atributo type='reset'

:selected

Selecciona elementos <options> que están seleccionados

:submit

Selecciona elementos <input> con el atributo type='submit'

:text

Selecciona elementos <input> con el atributo type='text'

$('#myForm :input'); // obtiene todos los elementos inputs

// dentro del formulario #myForm

Encadenamiento

$('#content').find('h3').eq(2).html('nuevo texto para el tercer elemento h3');

Obtener propiedades CSS

$('h1').css('fontSize'); // devuelve una cadena de caracteres como "19px"

$('h1').css('font-size');

Establecer propiedades CSS

$('h1').css('fontSize', '100px'); // establece una propiedad individual CSS

$('h1').css({

'fontSize' : '100px',

'color' : 'red'

}); // establece múltiples propiedades CSS

Trabajar con clases

var $h1 = $('h1');

$h1.addClass('big');

$h1.removeClass('big');

$h1.toggleClass('big');

if ($h1.hasClass('big')) { ... }

Métodos básicos sobre Dimensiones

$('h1').width('50px'); // establece el ancho de todos los elementos H1

$('h1').width(); // obtiene el ancho del primer elemento H1

$('h1').height('50px'); // establece el alto de todos los elementos H1

$('h1').height(); // obtiene el alto del primer elemento H1

$('h1').position(); // devuelve un objeto conteniendo

// información sobre la posición

// del primer elemento relativo al

// "offset" (posición) de su elemento padre

Establecer atributos

$('a').attr('href', 'index.html');

$('a').attr({

'title' : 'all titles are the same',

'href' : 'index.html'

});

**Teoria sobre eventos y callbacks**

Las funciones en JS son objetos. También tienen propiedades y métodos.

Callbacks

La función recibe como parámetros otra función y ese es el callback

callbacks

son funciones anonimas que se utilizan mucho cuando

el parametro que tenemos que enviar es una funcion

eventos

$('Elemento').on('evento', function(){

$(this)

.closest('')

.addClass('');

});

Resp

Eventos

En vanillaJS seria:

Var button = document.getElementById(‘mybutton’);

Button.addEventListener(‘click’, function (event) {

alert(‘Me hicieron click! ‘);

con JQuery :

$(‘button’).click(function () {

Alert(‘me hicieron’);

})

Pero en realidad lo que se realiza por detras es algo así:

$(‘button’).on(‘click’, function () {

Alert(‘me hicieron click’)})

$(‘.product button.like’).click(function (ev) {

$(this)

.closest(‘.product’)

.addClass(‘liked’)

El this termina siendo cada uno de los productos a medida que se le da.  
Hacer binding significa añadir a un evento una funcion

JQuery nos permite hacer multiples eventos con le mismo handler

$(‘input’).on(

“click change”,

Function() {

Console.log(“Me hicieron click o me cambiaron el texto!”);

}};

Y para multiples eventos distinto callback: o handlers

$( “p” ).on({

“click”: function() ( console.log(“ me hicieron click”);},

“mouseover”: function () { console.log (“me pasaron el mouse por encima”);}

});

Métodos de Eventos

Browser events: .resize(), scroll()

Document loading: .ready();

Event handler attachment: .off(), JQuery.proxy(),.on(),.one(),.trigger(), .triggerHandler(), unbind(), .undelegate(),

Event Object: Es el evento que nos llega por parametron cuando se dispara uno de estos eventos:

Event.currentTarget, event.preventDefault(), event.stopPropagation(), event.target, event.type

Form Events: blur(), select(), change(), submit(), focus()

Keyboard Events: focusin(), focusout(), keydown(), keypress(), keyup()

Mouse Events:

Click(), dblclick(), focusin(), focusout(), hover(), mousedown(), mouseenter(), mouseleave(), mousemove(), mouseout(), mouseover(), mouseup(),

Sacando eventos:

// scar los handlers bindeados  
// saca todos los handlers bindeados  
// al evento click de los elementos <p> de la pagina  
$(‘p’).off(‘click’);

Var foo = function () { console.log (“foo”); };

Var bar = function () { console.log (“bar”);};

$(“p”).on(“click”, foo).on(“click”, bar);  
$(“p”).off(“click”, bar); pero todavia esta la otra

Forms

$(“form”).on(“submit”, function(event) {

// Prevenir que el form haga post  
event.preventDefault();

// Hacemos lo que queremos, como loguear el evento  
console.log(event);

Var action = $(this).attr(‘action’);

//Hacer el request  
$.ajax(action, { /\*…..\*/})

});

Propagacion de eventos

En una lista con productos podemos bindear al evento click de a:

$(“#list a”).on(“click”, function(event) {

Event.preventDefault();  
console.log($(this).text());  
});

Cada vez que hacemos un click en un element burbujea en todo el DOM,  
pasaría al a al li al ul y a todo cojeria todo

Se podría escribir el código de otra manera:

// Attachear un evento delegado

$(“#list”).on(“click”, “a”, function(event){

Event.preventDefault();  
console.log($(this).text());  
});

Con esto lo que conseguimos es bindear la seleccion al list y asi no importa si mas adelante se crean mas enlaces porque temenos bindeado el “padre” que no va a cambiar.

Trigger

Disparar eventos manualmente  
$(‘#myButton”).trigger(“click”)  
$(“#myButton”).click()

Uso de eventos onClick y AJAX

$('#app-body')

.find('form')

.submit(function (ev) {

ev.preventDefault();

console.log(this);

})

})

Con esto el this seria el elemento de html form, no seria un JQuery object para eso tendríamos que usar $(this)

Para por ejemplo tener en una variable el contenido de la búsqueda de un input para una búsqueda por ejemplo en una base de datos seria algo asi:

$('#app-body')

.find('form')

.submit(function (ev) {

ev.preventDefault();

var busqueda = $(this)

.find('input[type="text"]')

.val();

alert('Se ha buscado: ' + busqueda);

})

})

Todo encadenando los métodos de la variable donde esta el selector del input

Para buscar contenido dentro de una API tenemos que saber sus Endpoints y nosotros podamos acceder a ellos mediante una carga AJAX

Necesitamos hacer un request de tipo GET (coger los datos sin modificar)

Con este código conseguimos hacer un template por cada uno de los elementos de la API con el método forEach

var template = '<article class="tv-show">' +

'<div class="left img-container">' +

'<img src=":img:" alt=":img alt:">' +

'</div>' +

'<div class="right info">' +

'<h1>:name:</h1>' +

"<p>:summary:</p>" +

'</div>' +

'</article>'

Esta es la plantilla

Y aqui sustituimos cada uno de los elementos

$.ajax({

url: 'http://api.tvmaze.com/shows',

success: function(shows, textStatus, xhr){

// es un callback ejecuta la funcion por cada elemento de arreglo

var $tvShowsContainer = $('#app-body').find('.tv-shows')

shows.forEach(function (show) {

var article = template

.replace(':name:', show.name)

.replace(':img:', show.image.medium)

.replace(':summary:', show.summary)

.replace(':img alt:', show.name + " Logo")

$tvShowsContainer.append($(article))

})

}

})