JavaScript

Sesión 7 - **jQuery Avanzado**

Índice

7.1 Efectos

- Facilita el uso de animaciones y efectos
- Mostrar y ocultar elementos
- Efectos de aparecer y desvanecer elementos
- Mover elementos a través de la pantalla
- http://api.jquery.com/category/effects/

Mostrar y Ocultar

Método	Propósito
show()	Muestra cada elemento del conjunto de resultados, si estaban ocultos
show(*velocidad*[,*callback*])	Muestra todos los elementos del conjunto del resultados mediante una animación, y opcionalmente lanza un *callback* tras completar la animación
hide()	Oculta cada elemento del conjunto de resultados, si estaban visibles
hide(*velocidad*[,*callback*])	Oculta todos los elementos del conjunto del resultados mediante una animación, y opcionalmente lanza un *callback* tras completar la animación
toggle()	Cambia la visualización (visible u oculto, de manera contraria a su estado) para cada elemento del conjunto de resultados
toggle(*switch*)	Cambia la visualización para cada elemento del conjunto de resultados dependiendo del *switch* (verdadero muestra todos los elementos, falso para ocultarlos)
toggle(*velocidad*[,*callback*])	Cambia la visualización de todos los elementos del conjunto del resultados mediante una animación, y opcionalmente lanza un *callback* tras completar la animación

Velocidad de Animación

- Valor numérico → cantidad de milisegundos de duración del efecto
- Cadena → `slow` (lento - 600ms), `normal` (400ms) o `fast` (rápido - 200ms).

```
$(function() {
   $("#mostrar").click(function() {
      $("#laCapa").show("normal");
   });
   $("#ocultar").click(function() {
      $("#laCapa").hide(2000);  // ms
   });
   $("#cambiar").click(function() {
      $("#laCapa").toggle("slow");
   });
});
```

http://jsbin.com/zakaga/1/edit?html,css,js,output

Aparecer y Desvanecer

Método	Propósito
fadeIn(*velocidad*[,*callback*])	El contenido aparece a la *velocidad* indicada para cada elemento del conjunto de resultados, y opcionalmente lanza un *callback* tras completar la animación
fadeOut(*velocidad*[,*callback*])	El contenido se desvanece a la *velocidad* indicada para cada elemento del conjunto de resultados, y opcionalmente lanza un *callback* tras completar la animación
fadeTo(*velocidad*,*opacidad*[,*callback*])	El contenido cambia a la *opacidad* y *velocidad* indicadas para cada elemento del conjunto de resultados, y opcionalmente lanza un *callback* tras completar la animación

Ejemplo Aparecer y Desvanecer

```
$(function() {
   $("#aparecer").click(function() {
      $("#laCapa").fadeIn(300);
   });
   $("#desvanecer").click(function() {
      $("#laCapa").fadeOut("normal");
   });
   $("#fade03").click(function() {
      $("#laCapa").fadeTo("slow", 0.3); // opacidad 0,3
   });
   $("#fade10").click(function() {
      $("#laCapa").fadeTo("slow", 1.0);
   });
});
```

http://jsbin.com/holumo/1/edit?html,css,js,output

Uso de Callbacks

- Si a las animaciones le pasamos una función *callback*, se ejecutará al terminar la animación

```
$(this).fadeOut(1000, function() {
  console.log("He acabado");
});
```

```
$(this).fadeOut(500, function() {
  $(this).delay(2000).fadeIn(500);
});
```

- Si hubiésemos puesto la función fuera del parámetro del `fadeOut`, se hubiese ejecutado justamente después de iniciarse la animación, y no al finalizar la misma.

```
$(this).fadeOut(500).css("margin","50 px");
```

Enrollar y Desenrollar (persiana)

Método	Propósito
slideDown(*velocidad*[,*callback*])	El contenido se desenrolla a la *velocidad* indicada modificando la altura de cada elemento del conjunto de resultados , y opcionalmente lanza un *callback* tras completar la animación
slideUp(*velocidad*[,*callback*])	El contenido se enrolla a la *velocidad* indicada modificando la altura de cada elemento del conjunto de resultados , y opcionalmente lanza un *callback* tras completar la animación
slideToggle(*velocidad*[,*callback*])	Cambia la visualización del contenido enrollando o desenrollando el contenido a la *velocidad* indicada modificando la altura de cada elemento del conjunto de resultados, y opcionalmente lanza un *callback* tras completar la animación

Ejemplo Enrollar y Desenrollar

```
$(function() {
   $("#enrollar").click(function() {
      $("#laCapa").slideUp("normal");
   });
   $("#desenrollar").click(function() {
      $("#laCapa").slideDown(2000);
   });
   $("#cambiar").click(function() {
      $("#laCapa").slideToggle("slow");
   });
});
```

http://jsbin.com/hunoq/1/edit?html,css,js,output

Creando Animaciones

Método	Propósito
animate(*parámetros*, *duración*, *easing*, *callback*)	Crea una animación personalizada donde *parámetros* indica un objeto CSS con las propiedades a animar, con una *duración* y *easing* (linear o *swing*) determinados y lanza un *callback* tras completar la animación
animate(*parametros*, *opciones*)	Crea una animación personalizada donde *parametros* indica las propiedades a animar y las *opciones* de las animación (*complete*, *step*, *queue*)
stop()	Detiene todas las animaciones en marcha para todos los elementos

Ejemplo Animaciones

```
$(function() {
  $("#derecha").click(function() {
    $("#laCapa").animate({ width: "500px" }, 1000);
  });
  $("#texto").click(function() {
    $("#laCapa").animate({ fontSize: "24pt" }, 1000);
  });
  $("#mover").click(function() {
    $("#laCapa").animate({ left: "500" }, 1000, "swing");
  });
  $("#todo").click(function() {
    $("#laCapa").animate({ width: "500px", fontSize: "24pt", left: "500" },
                         1000, "swing");
  });
});
```

http://jsbin.com/wiyodo/2/edit?html,css,js,output

Carrusel

```
<div id="carrusel">
   <div class="actual"><img src="imagenes/hierba.jpg" width="400" height="400" class="galeria" /></div>
   <div><img src="imagenes/hoja.jpg"  width="400" height="400" class="galeria" /></div>
   <div><img src="imagenes/primavera.jpg" width="400" height="400" class="galeria" /></div>
   <div><img src="imagenes/agua.jpg" width="400" height="400" class="galeria" /></div>
</div>
```

```
$(function () {
   setInterval("carruselImagenes()", 2000);
});

function carruselImagenes() {
   var fotoActual = $('#carrusel div.actual');
   var fotoSig = fotoActual.next();
   if (fotoSig.length == 0) {
      fotoSig = $('#carrusel div:first');
   }
   fotoActual.removeClass('actual').addClass('anterior');
   fotoSig.css({ opacity: 0.0 }).addClass('actual')
      .animate({ opacity: 1.0 }, 1000,
         function () { fotoActual.removeClass('anterior'); });
}
```

Deshabilitando Efectos

- Si el sistema donde corre nuestra aplicación es poco potente, podemos deshabilitar todos los efectos y animaciones haciendo uso de la propiedad booleana `jQuery.fx.off`
- Los elementos aparecerán y desapareceran sin ningún tipo de animación.

```
$.fx.off = true;
// Volvemos a activar los efectos
$.fx.off = false;
```

7.2 AJAX

- http://api.jquery.com/category/ajax/
- `$.ajax()`

Método	Propósito
***selector*.load(url)**	Incrustra el contenido crudo de la *url* sobre el *selector*
$.get(url, callback, tipoDatos)	Realiza una petición GET a la *url*. Una vez recuperada la respuesta, se invocará el *callback* el cual recuperará los datos del *tipoDatos*
$.getJSON(url, callback)	Similar a `$.get()` pero recuperando datos en formato JSON
$.getScript(url, callback)	Carga un archivo *JavaScript* de la *url* mediante un petición GET, y lo ejecuta.
$.post(url, datos, callback)	Realiza una petición POST a la *url*, enviando los datos como parámetros de la petición

`$.ajax()`

- Recibe como parámetro un objeto

Propiedad	Propósito
url	URL a la que se realiza la petición
type	Tipo de petición (GET o POST)
dataType	Tipo de datos que devuelve la petición, ya sea texto o binario.
success	*Callback* que se invoca cuando la petición ha sido exitosa y que recibe los datos de respuesta como parámetro
error	*Callback* que se invoca cuando la petición ha fallado
complete	*Callback* que se invoca cuando la petición ha finalizado

Ejemplo `$.ajax()`

```
$.ajax({
   url: "fichero.txt",
   type: "GET",
   dataType: "text",
   success: todoOK,
   error: fallo,
   complete: function(xhr, estado) {
      console.log("Peticion finalizada " + estado);
   }
});

function todoOK(datos) {
   console.log("Todo ha ido bien " + datos);
}

function fallo(xhr, estado, msjErr) {
   console.log("Algo ha fallado " + msjErr);
}
```

```
function todoOK(datos) {
   $("#resultado").append(datos);
}
```

`selector.load(url)`

- Permite incrustar contenido vía una URL
- Similar a include estático JSP
- Podemos filtrar el contenido indicando un selector

```
$("body").load("contacto.html");
```

```
$("body").load("contacto.html .contenido");
```

```
<a href="contacto.html">Contacto</a>
<div id="contenedor"></div>

<script>
$('a').on('click', function(evt) {
   var href = $(this).attr('href');
   $('#contenedor').load(href + ' .contenido');
   evt.preventDefault();
});
</script>
```

$.get()

- Petición GET
- `$.get(url [,datos], callBack(datosRespuesta) [,tipoDatosRespuesta])`

```
$.get("fichero.txt", function(datos) {
   console.log(datos);
});
```

- Permite trabajar con XML

```
<heroe>
   <nombre>Batman</nombre>
   <email>batman@heroes.com</email>
</heroe>
```

```
$.get("heroes.xml", function(datos) {
   var nombre = datos.getElementsByTagName("nombre")[0];
   var email = datos.getElementsByTagName("email")[0];
   var val = nombre.firstChild.nodeValue + " " + email.firstChild.nodeValue;
   $("#resultado").append(val);
},"xml");
```

$.getJSON()

- Espera que la petición GET devuelva contenido JSON
- `$.getJSON(url [,datos], callBack(datosRespuesta))`

```
var flickrAPI = "http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?";
$.getJSON( flickrAPI, {
   tags: "proyecto víbora ii",
   tagmode: "any",
   format: "json"
}, formateaImagenes);

function formateaImagenes(datos) {
   $.each(datos.items, function(i, elemento) {
      $("<img>").attr("src", elemento.media.m).appendTo("#contenido");
      if (i === 4) {
         return false;
      }
   });
}
```

http://jsbin.com/hunape/1/edit?html,js,output

`$.each()`

- Método auxiliar de jQuery para iterar sobre una colección.
  - un array mediante `$.each(colección, callback(índice, elemento))`
  - un conjunto de selectores con `$(selector).each(callback(índice, elemento))`
  - las propiedades de un objeto mediante `$.each(objeto, callback(clave, valor))`

- Se emplea mucho al trabajar con AJAX para recorrer los datos obtenidos.

- http://api.jquery.com/jquery.each/

`$.getScript()`

- `$.getScript(urlScript, callback)`
- Permite inyectar código *al vuelo*
- Recupera un archivo JavaScript y lo ejecuta a continuación.
- Una vez finalizada la ejecución del script, se invocará al callback.

```
// script.js
console.log("Ejecutado dentro del script");
$("#resultado").html("<strong>getScript</strong>");
```

```
$.getScript("script.js", function(datos, statusTxt) {
  console.log(statusTxt);
})
```

`$.post()`

- `$.post(url, datos, callback(datosRespuesta))`
- Modos de adjuntar los datos:
  - Creando un objeto cuyos valores obtenemos mediante `val()`.
  - Serializando el formulario mediante el método `serialize()`, el cual codifica los elementos del formulario mediante una cadena de texto

```
<form name="formCliente" id="frmClnt" action="#">
<fieldset id="infoPersonal">
   <legend>Datos Personales</legend>
   <p><label for="nombre">Nombre</label>
      <input type="text" name="nombre" id="idNombre" /></p>
   <p><label for="correo">Email</label>
      <input type="email" name="correo" id="idEmail" /></p>
</fieldset>
<p><button type="submit">Guardar</button></p>
</form>
```

```
$('form').on('submit', function(evt) {
   evt.preventDefault();
   // var nom = $(this).find('#inputName').val();
   var datos = $(this).serialize(); // nombre=asdf&email=asdf
   $.post("/GuardaFormServlet", datos, function (respuestaServidor) {
      console.log("Completado " + respuestaServidor);
   });
});
```

Tipos de Datos

- Fragmentos **HTML** → mediante `load()` podemos cargarlos sin necesidad de ejecutar ningún callback.
  - Los datos puede que no tengan ni la estructura ni el formato que necesitemos
  - Acopla nuestro contenido con el externo.
- Archivos **JSON** → permiten estructurar la información para su reutilización. Compactos y fáciles de usar, donde la información es auto-explicativa y se puede manejar mediante objetos mediante `JSON.parse()` y `JSON.stringify()`.
  - Hay que vigilar los errores en el contenido de los archivos ya que pueden provocar efectos colaterales.
- Archivos **JavaScript** → ofrecen flexibilidad pero no son realmente un mecanismo de almacenamiento, ya que no podemos usarlos desde sistemas heterogéneos.
  - La carga de scripts JavaScripts en caliente permite refactorizar el código en archivos externos, reduciendo el tamaño del código hasta que sea necesario.
- Archivos **XML** → han perdido mercado en favor de JSON
  - Se sigue utilizando para permitir que sistemas de terceros sin importar la tecnología de acceso puedan conectarse a nuestros sistemas.

Eventos AJAX

- Métodos globales para interactuar con los eventos que se lanzan al realizar una petición AJAX.
- No los llamamos dentro de la aplicación, sino que es el navegador el que realiza las llamadas.

Método	Propósito
ajaxComplete()	Registra un manejador que se invocará cuando la petición AJAX se complete
ajaxError()	Registra un manejador que se invocará cuando la petición AJAX se complete con un error
ajaxStart()	Registra un manejador que se invocará cuando la primera petición AJAX comience
ajaxStop()	Registra un manejador que se invocará cuando todas las peticiones AJAX hayan finalizado
ajaxSend()	Adjunta una función que se invocará antes de enviar la petición AJAX
ajaxSuccess()	Adjunta una función que se invocará cuando una petición AJAX finalice correctamente

Ejemplo Eventos

```
$("document").ready(function() {
   $(document).ajaxStart(function () {
      console.log("AJAX comenzando");
   });
   $(document).ajaxStop(function () {
      console.log("AJAX petición finalizada");
   });
   $(document).ajaxSend(function () {
      console.log("Antes de enviar la información...");
   });
   $(document).ajaxComplete(function () {
      console.log("Todo ha finalizado!");
   });
   $(document).ajaxError(function (evt, jqXHR, settings, err) {
      console.log("Houston, tenemos un problema: " + err);
   });
   $(document).ajaxSuccess(function () {
      console.log("Parece que ha funcionado todo!");
   });
   getDatos();
});
```

http://jsbin.com/loqace/5/edit?js,console

```
AJAX comenzando
Antes de enviar la información...
▶XHR finished loading: GET "http://localhost:63342/Pruebas/jquery-ajax/fichero.txt".
Datos recibidos y adjuntándolos a resultado
Parece que ha funcionado todo!
Todo ha finalizado!
AJAX peticion finalizada
```

```
function getDatos() {
   $.getJSON("http://api.openbeerdatabase.com/v1/beers.json?callback=?", todoOk);
}

function todoOk(datos) {
   console.log("Datos recibidos y adjuntándolos a resultado");
   $("#resultado").append(JSON.stringify(datos));
}
```

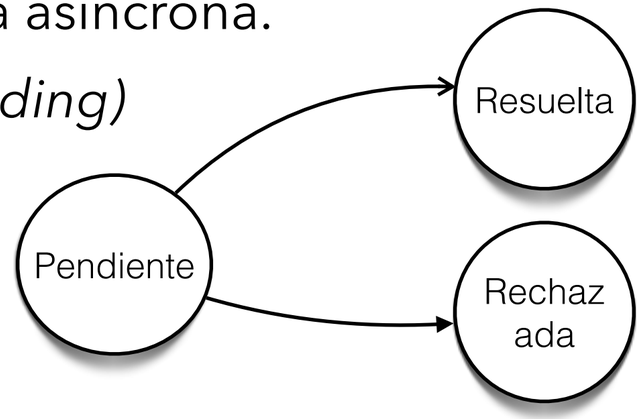
7.3 *Deferreds* / Promesas

- Implementación que hace *jQuery* de las promesas
  - Alternativa: Q (https://github.com/kriskowal/q)
- Facilitan la gestión los eventos asíncronos
  - Código más sencillo
  - Callbacks más cortos
  - Separar la lógica de la aplicación de alto nivel de de los comportamientos de bajo nivel.
- Permite usar callbacks en cualquier situación, y no solo con eventos.
- Ofrece un mecanismo estándar para indicar la finalización de tareas.

- *ECMAScript 6* da soporte a las promesas.
- *jQuery* ofrece los *Deferreds* como una implementación de las promesas independiente de la versión de ECMAScript que tenga nuestro navegador.

Promesa

- Objeto que representa un evento único, como resultado de una tarea asíncrona.
- Nada más comenzar, una promesa está en un estado **pendiente** *(pending)*
- Al finalizar, su estado será:
  - **resuelta** (*resolved*) → la tarea se ha realizado
  - o **rechazada** (*rejected*) → la tarea falló



- Una vez que una promesa se resuelve o se rechaza, se mantendrá en dicho estado para siempre, y sus callbacks nunca se volverán a disparar.
- Los *callbacks* que se adjuntan a una promesa, se dispararán una vez que la promesa se resuelva o rechace.
  - Podemos añadir más callback cuando queramos, incluso si la promesa ya ha sido resuelta o rechazada (en dicho caso, se ejecutarán inmediatamente).
- Podemos combinar varias promesas en una nueva.
  - Facilita escribir código de finalización de tareas paralelas, del tipo "Cuando todas estas cosas hayan finalizado, haz esta otra".

`$.Deferred()`

- Un *Deferred* es una promesa con métodos que permiten a su propietario resolverla o rechazarla
- Todas las promesas de otros propietarios son de sólo lectura.
- Operaciones:
  - `state()` → estado de la promesa
  - `resolve()` → resuelve la promesa
  - `reject()` → rechaza la promesa

```
var deferred = new $.Deferred();

deferred.state();  // pending
deferred.resolve();
deferred.state();  // resolved
deferred.reject();
```

- Si al método constructor le pasamos una función, ésta se ejecutará tan pronto como el objeto se cree, y la función recibe como parámetro el nuevo objeto `Deferred`.
  - Permite crear un envoltorio que realiza una tarea asíncrona y que dispare un *callback* cuando haya finalizado:

```
function realizarTarea() {
   return $.Deferred(function(def) {
      // tarea async que dispara un callback al acabar
   });
}
```

`promise()`

- Permite obtener una promesa pura.
- Similar a `Deferred`, excepto que faltan los métodos de `resolve()` y `reject()`.
- Se emplea para dar soporte a la encapsulación
  - Si una función devuelve un `Deferred`, puede ser resuelta o rechazada por el programa que la invoca.
  - Si sólo devolvemos la promesa pura correspondiente al *Deferred*, el programa que la invoca sólo puede leer su estado y añadir *callbacks*, no puede modificar su estado.
- Enfoque que sigue *jQuery* con `$.ajax()`

```
var obteniendoProductos = $.get("/products");

obteniendoProductos.state();  // "pending"
obteniendoProductos.resolve();  // undefined
```

Manejadores de Promesas

- Una vez tenemos una promesa, podemos adjuntarle tantos *callbacks* como queremos mediante los métodos:
  - `done()` → se lanza cuando la promesa se resuelve correctamente mediante `resolve()`
  - `fail()` → se lanza cuando la promesa se rechaza mediante `reject()`
  - `always()` → se lanza cuando se completa la promesa, independientemente que su estado sea resuelta o rechazada

```
promesa.done(function() {
   console.log("Se ejecutará cuando la promesa se resuelva.");
});

promesa.fail(function() {
   console.log("Se ejecutará cuando la promesa se rechace.");
});

promesa.always(function() {
   console.log("Se ejecutará en cualquier caso.");
});
```

Encadenando Promesas - `then()`

```
promesa.done(function() {
  console.log("Se ejecutará cuando la promesa se resuelva.");
}).fail(function() {
  console.log("Se ejecutará cuando la promesa se rechace.");
}).always(function() {
  console.log("Se ejecutará en cualquier caso.");
});
```

- `promesa.then(doneCallback, failCallback, alwaysCallback);`

```
promesa.then(function() {
  console.log("Se ejecutará cuando la promesa se resuelva.");
}, function() {
  console.log("Se ejecutará cuando la promesa se rechace.");
}, function() {
  console.log("Se ejecutará en cualquier caso.");
});
```

Orden de callbacks

- El orden en el que se adjuntan los callbacks definen su orden de ejecución.

```
var promesa = $.Deferred();
promesa.done(function() {
   console.log("Primer callback.");
}).done(function() {
   console.log("Segundo callback.");
}).done(function() {
   console.log("Tercer callback.");
});

promesa.fail(function() {
   console.log("Houston! Tenemos un problema");
});

promesa.always(function() {
   console.log("Dentro del always");
}).done(function() {
   console.log("Y un cuarto callback si todo ha ido bien");
});
```

```
"Primer callback."
"Segundo callback."
"Tercer callback."
"Dentro del always"
"Y un cuarto callback si todo ha ido bien"
```

http://jsbin.com/wanavo/1/edit?html,js,console,output

Prestando Promesas del Futuro

- Para separar la creación de una promesa del *callback* de lógica de aplicación → reenviar los eventos de `resolve`/`reject` desde la promesa POST a una promesa que se encuentre fuera de nuestro alcance.
- En vez de necesitar varias líneas con código anidado del tipo `promesa1.done(promesa2.resolve());` → Usar **`then()`**.

- `promesa.`**`then`**`(doneCallback, failCallback, alwaysCallback);`
  - Devuelve una nueva promesa que permite filtrar el estado y los valores de una promesa mediante una función
  - Es una ventana al futuro → permite adjuntar comportamiento a una promesa que todavía no existe.

Ejemplo `then()`

```
var enviandoObservaciones = new $.Deferred();
var guardandoObservaciones = enviandoObservaciones.then(function(input) {
  return $.post("/observaciones", input);
});
```

```
$("#observaciones").submit(function() {
  enviandoObservaciones.resolve($("textarea", this).val());
  return false;
});

enviandoObservaciones.done(function() {
  $("#contenido").append("<div class='spinner'>");
});
guardandoObservaciones.then(
  function() {  // done
    $("#contenido").append("<p>¡Gracias por las observaciones!</p>");
  }, function() { // fail
    $("#contenido").append("<p>Se ha producido un error al contactar con el servidor.</
p>");
  }, function() { // always
    $("#contenido").remove(".spinner");
});
```

Intersección de Promesas - `$.when()`

- Las promesas siguen un esquema binario
- Se pueden combinar como si fuesen *booleanos*

- `$.when(promesa1, promesa2, promesa3, …)` → intersección de promesas
- Devuelve una nueva promesa que cumple estas reglas:
  - Cuando todas las promesas recibidas se resuelven, la nueva promesa esta resuelta.
  - Cuando alguna de las promesas recibidas se rechaza, la nueva promesa se rechaza.

- Permite crear un punto de sincronización de promesas

Ejemplo `$.when()`

```
$("#contenido").append("<div class='spinner'>");
$.when( $.get(“/encryptedData"), $.get("/encryptionKey"))
   .then(function() { // done
      // ambas llamadas han funcionado
   }, function() { // fail
      // una de las llamadas ha fallado
   }, function() { // always
      $("#contenido").remove(".spinner");
});
```

AJAX mediante *Deferreds*

- El objeto `jxXHR` que se obtiene de los métodos AJAX como `$.ajax()` o `$.getJSON()` implementan el interfaz `Promise`
  - Vamos a poder utilizar los métodos `done`, `fail`, `then`, `always` y `when()`.

```
function getDatos() {
  var peticion = $.getJSON("http://api.openbeerdatabase.com/v1/beers.json?callback=?");
  peticion.done(todoOk).fail(function() {
    console.log("Algo ha fallado");
  });
  peticion.always(function() {
    console.log("Final, bien o mal");
  });
}

function todoOk(datos) {
  console.log("Datos recibidos y adjuntándolos a resultado");
  $("#resultado").append(JSON.stringify(datos));
}
```

http://jsbin.com/ponaca/2/edit?html,js,console,output

7.4 Utilidades - Comprobación de Tipos

- jQuery ofrece un conjunto de utilidades como funciones globales → `$.funcion()`

Función	Propósito
$.isArray(*array*)	Determina si *array* es un *Array*. Si es un objeto *array-like* devolverá falso
$.isFunction(*función*)	Determina si *función* es una Función
$.isEmptyObject(*objeto*)	Determina si *objeto* esta vacío
$.isPlainObject(*objeto*)	Determina si *objeto* es un objeto sencillo, creado como un objeto literal (mediante las llaves) o mediante `new objeto`.
$.isXmlDoc(*documento*)	Determina si el *documento* es un documento XML o un nodo XML
$.isNumeric(*objeto*)	Determina si *objeto* es un valor numérico escalar.
$.isWindow(*objeto*)	Determina si *objeto* representa una ventana de navegador
$.type(*objeto*)	Obtiene la clase *Javascript* del *objeto*. Los posibles valores son `boolean`, `number`, `string`, `function`, `array`, `date`, `regexp`, `object`, `undefined` o `null`

Manipulando Colecciones

Función	Propósito
$.makeArray(*objeto*)	Convierte el *objeto* en un *array*. Se utiliza cuando necesitamos llamar a funciones que sólo soportan los *arrays*, como `join` o `reverse`, o cuando necesitamos pasar un parametro a una función como *array*
$.inArray(*valor*, *array*)	Determina si el *array* contiene el *valor*. Devuelve `-1` o el índice que ocupa el *valor*. Un tercer parámetro opcional permite indicar el índice por el cual comienza la búsqueda.
$.unique(*array*)	Elimina cualquier elemento duplicado que se encuentre en el *array*
$.merge(*array1*, *array2*)	Combina los contenidos de *array1* y *array2*
$.map(*array*, *callback*)	Construye un nuevo array cuyo contenido es el resultado de llamar al *callback* para cada elemento
$.grep(*array*, *callback* [,invertido])	Filtra el *array* mediante el *callback*, de modo que añadirá los elementos que pasen la función, la cual recibe un objeto DOM como parámetro, y devuelve un *array JavaScript*

Copiando Objetos

- `$.extend([boolRecursivo,] destino, origen)`
- Permite copiar miembros de un objeto fuente en uno destino, sin realizar herencia, sólo clonado las propiedades.
- Si hay un conflicto, se sobreescribirán con las propiedades del objeto fuente, y si tenemos múltiples objetos fuentes, de izquierda a derecha.
- Si los objetos a clonar contienen objetos anidados → primer parámetro a `true`
- http://api.jquery.com/jquery.extend/

```
var animal = {
   comer: function() {
      console.log("Comiendo");
   }
}

var perro = {
   ladrar: function() {
      console.log("Ladrando");
   }
}

$.extend(perro, animal);
perro.comer();  // Comiendo
```

Ejemplo `$.extend()`

```
var perroCopia = {};

$.extend(perroCopia, perro);
perroCopia.acciones.ladrar(); // Ladrando

$.extend(true, perroCopia, animal);
perroCopia.acciones.comer();  // Comiendo
perroCopia.acciones.ladrar(); // Ladrando

$.extend(perro, animal);
perro.acciones.comer();   // Comiendo
perro.acciones.ladrar();  // error
```

```
var animal = {
   acciones: {
      comer: function() {
         console.log("Comiendo");
      },
      sentar: function() {
         console.log("Sentando");
      }
   }
};

var perro = {
   acciones: {
      ladrar: function() {
         console.log("Ladrando");
      },
      cavar: function() {
         console.log("Cavando");
      }
   }
};
```

7.5 Plugins

- *jQuery* soporta una arquitectura de plugins para extender la funcionalidad de la librería.
- Repositorio → http://plugins.jquery.com/,
  - Listado con demos, código de ejemplo y tutoriales para facilitar su uso.
- Sesión 8 → *jQueryUI*

- En ocasiones necesitamos escribir nuestro propio código que podemos empaquetar como un nuevo plugin.
  - Nombrado: `jquery.nombrePlugin.js` o `jquery.nombrePlugin-1.0.js`
  - En el fichero, poner un comentario con la versión

Creando un *plugin*

- A la hora de crear un plugin, se asume que *jQuery* ha cargado.
- No asumir que el alias `$` esté disponible.
- Dentro del plugin se recomienda utilizar el nombre completo `jQuery` o redefinir el `$`.
- Podemos hacer uso de una IIFE para poder usar el `$` dentro de nuestro plugin.

```
(function($) {
   // código del plugin
})(jQuery);
```

Funciones Globales

- Similares a `$.ajax()`
- No necesitan ningún objeto `jQuery` para funcionar
- Amplian el abanico de funciones de utilidades.
- Para añadir una función al espacio de nombre de *jQuery* → asignar la función como una propiedad del objeto `jQuery`:

```
(function($) {
   $.suma = function(array) {
      var total = 0;
      $.each(array, function (indice, valor) {
         valor = $.trim(valor);
         valor = parseFloat(valor) || 0;

         total += valor;
      });
      return total;
   };
})(jQuery);
```

```
var resultado = $.suma([1,2,3,4]); // 10
```

Espacio de Nombres

- Hay que evitar el conflicto de nombres.
- Asociar las funciones a un objeto.
- El objeto contiene como propiedades las funciones que queramos añadir al plugin.

```
(function($) {
   $.MathUtils = {
      suma : function(array) {
         // código de la función
      },
      media: function(array) {
         // código de la función
      }
   };
})(jQuery);
```

```
var resultado = $.MathUtils.suma([1,2,3,4]);
var resultado = $.MathUtils.media([1,2,3,4]);
```

Métodos de Objeto

- Para extender las funciones de *jQuery*, mediante prototipos podemos crear métodos nuevos que se apliquen al objeto `jQuery` activo.
- jQuery utiliza el alias **`fn`** en vez `prototype`.

```
$.fn.nombreNuevaFuncion = function() {
   // código nuevo
}
```

- Al no saber si la función trabajará sobre un sólo objeto o sobre una colección (un selector puede devolver cero, uno o múltiples elementos.), hay que plantear un escenario donde recibimos un array de datos.
- Solución → iterar sobre la colección mediante el método `each()`.

```
$.fn.nombreNuevaFuncion = function() {
   this.each(function() {
      // Hacemos algo con cada elemento
   });
};
```

Encadenar Funciones

- Al crear una función prototipo después de llamar a nuestra función, es posible que el desarrollador quiera seguir encadenando llamadas.
- La función tiene que devolver un objeto `jQuery` para permitir que continue el encadenamiento.
  - Normalmente es `this`.

```
(function($) {
  $.fn.cambiarClase = function(clase1, clase2) {
    return this.each(function() {
      var elem = $(this);
      if (elem.hasClass(clase1)) {
        elem.removeClass(clase1).addClass(clase2);
      } else if (elem.hasClass(clase2)) {
        elem.removeClass(clase2).addClass(clase1);
      }
    });
  };
})(jQuery);
```

Opciones

- Una buena práctica es permitir que el plugin reciba un objeto con las opciones de configuración.

```
$.fn.pluginConConf = function(opciones) {
  var confFabrica = {prop: "valorPorDefecto"};
  var conf = $.extend(confFabrica, opciones);

  return this.each(function() {
    // código que trabaja con conf.prop
  });
};
```

- Para permitir modificar los valores de fabrica → extraer a una propiedad del método

```
$.fn.pluginConConf = function(opciones) {
  var conf = $.extend({},$.fn.pluginConConf.confFabrica,opciones);

  return this.each(function() {
    // código que trabaja con conf.prop
  });
};
$.fn.pluginConConf.confFabrica = {prop: "valorPorDefecto"};
```

7.6 Rendimiento - Consejos

- Consejo Nº 0: A ser posible, **siempre es mejor utilizar *JavaScript*** "puro" a *jQuery*
  - *jQuery* envuelve a *JavaScript*, y al ofrecer *Cross-Browser* hay tareas sencillas que las hace más costosas computacionalmente

- Consejo Nº 1: Utilizar la **última versión de *jQuery***, ya que siempre contienen mejoras de rendimiento que repercutirán en nuestra aplicación.
- Leer la '*release note*' antes de modificar la librería en producción

Consejo N°2 - Cachear los selectores

- Evitar búsquedas innecesarias.

```
console.time("Sin cachear");
for (var i=0; i < 1000; i++) {
   var s = $("div");
}
console.timeEnd("Sin cachear");
```

8.990 ms

```
console.time("Cacheando");
var miCapa = $("div");
for (var i=0; i < 1000; i++) {
   var s = miCapa;
}
console.timeEnd("Cacheando");
```

0.045 ms

±200 veces menos

Consejo Nº 3 - Cachear otros elementos

- Cachear llamadas a métodos, propiedades
  - No usar `.length` en la condición de un `for` → Mejor al iniciar la variable o fuera del bucle
  - Cachear llamadas AJAX

```
function getDatos(id) {
 if (!api[id]) {
    var url = "http://api.openbeerdatabase.com/v1/beers/" + id + ".json?callback=?";
    console.log("Petición a " + url);
  api[id] = $.getJSON(url);
  }

  api[id].done(todoOk).fail(function() {
    $("#resultado").html("Error");
  });
}
function todoOk(datos) {
  console.log("Datos recibidos y adjuntándolos a resultado");
  $("#resultado").append(JSON.stringify(datos));
}
```

Consejo Nº 4 - Utilizar propiedades de elementos

- Se basa en el consejo Nº 0
- En vez de usar un método *jQuery*, acceder a la propiedad mediante *JavaScript*.
  - cuidado que al acceder mediante una propiedad el resultado deja de ser un objeto *jQuery*, con lo que no vamos a poder encadenar el resultado en una llamada posterior

```
var lento = miCapa.attr("id");
var rapido = miCapa[0].id;
```

¿Preguntas?