INTRODUCCION

JQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC. jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos.[1] jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Características

- Selección de elementos DOM.
- Interactividad y modificaciones del árbol DOM, incluyendo soporte para CSS 1-3 y un plugin básico de XPath.
- Eventos.
- Manipulación de la hoja de estilos CSS.
- Efectos y animaciones.
- Animaciones personalizadas.
- AJAX.
- Soporta extensiones.
- Utilidades varias como obtener información del navegador, operar con objetos y vectores, funciones para rutinas comunes, etc.
- Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.[4]

Es Importante saber que siempre tenemos que hacer el llamado al archivo núcleo de jQuery para que el código pueda ser interpretado, el mismo se encuentra en la siguiente ruta: http://code.jquery.com/jquery-1.8.3.min.js

CONTENIDO

1 Conceptos Básicos de jQuery

1.1 $(document).ready()

No es posible interactuar de forma segura con el contenido de una página hasta que el documento no se encuentre preparado para su manipulación. jQuery permite detectar dicho estado a través de la declaración $(document).ready() de forma tal que el bloque se ejecutará sólo una vez que la página este disponible.

El bloque $(document).ready()

```
$(document).ready(function() {
    console.log('el documento está preparado');
  });
```

Existe una forma abreviada para $(document).ready() la cual podrá encontrar algunas veces; sin embargo, es recomendable no utilizarla en caso que este escribiendo código para gente que no conoce jQuery.

Forma abreviada para $(document).ready()

```
$(function() {
    console.log('el documento está preparado');
```

```
});
```

Además es posible pasarle a $(document).ready() una función nombrada en lugar de una anónima:

Pasar una función nombrada en lugar de una función anónima

```
function readyFn() {
    // código a ejecutar cuando el documento este listo
}
$(document).ready(readyFn);
```

1.2 Selección de Elementos

El concepto más básico de jQuery es el de "seleccionar algunos elementos y realizar acciones con ellos". La biblioteca soporta gran parte de los selectores CSS3 y varios más no estandarizados. En http://api.jquery.com/category/selectors/ se puede encontrar una completa referencia sobre los selectores de la biblioteca.

A continuación se muestran algunas técnicas comunes para la selección de elementos:

Selección de elementos en base a su ID

```
$('#myId'); // notar que los IDs deben ser únicos por página
```

Selección de elementos en base al nombre de clase

```
$('div.myClass'); // si se especifica el tipo de elemento,
                  // se mejora el rendimiento de la selección
```

Selección de elementos por su atributo

```
$('input[name=first_name]'); // tenga cuidado, que puede ser muy lento
```

Selección de elementos en forma de selector CSS

```
$('#contents ul.people li');
```

Pseudo-selectores

```
$('a.external:first');  // selecciona el primer elemento <a>
                        // con la clase 'external'
$('tr:odd');            // selecciona todos los elementos <tr>
                        // impares de una tabla
$('#myForm :input');    // selecciona todos los elementos del tipo input
                        // dentro del formulario #myForm
$('div:visible');       // selecciona todos los divs visibles
$('div:gt(2)');         // selecciona todos los divs excepto los tres primeros
$('div:animated');      // selecciona todos los divs actualmente animados
```

Nota

Cuando se utilizan los pseudo-selectores :visible y :hidden, jQuery comprueba la visibilidad actual del elemento pero no si éste posee asignados los estilos CSS visibility o display — en otras palabras, verifica si *el alto y ancho físico del elemento* es mayor a cero. Sin embargo, esta comprobación no funciona con los elementos <tr>; en

este caso, jQuery comprueba si se está aplicando el estilo display y va a considerar al elemento como oculto si posee asignado el valor none. Además, los elementos que aún no fueron añadidos al DOM serán tratados como ocultos, incluso si tienen aplicados estilos indicando que deben ser visibles (En la sección Manipulación de este manual, se explica como crear y añadir elementos al DOM).

Como referencia, este es el fragmento de código que utiliza jQuery para determinar cuando un elemento es visible o no. Se incorporaron los comentarios para que quede más claro su entendimiento:

```
jQuery.expr.filters.hidden = function( elem ) {
    var width = elem.offsetWidth, height = elem.offsetHeight,
      skip = elem.nodeName.toLowerCase() === "tr";

    // ¿el elemento posee alto 0, ancho 0 y no es un <tr>?
    return width === 0 && height === 0 && !skip ?

      // entonces debe estar oculto (hidden)
      true :

      // pero si posee ancho y alto
      // y no es un <tr>
      width > 0 && height > 0 && !skip ?

        // entonces debe estar visible
        false :

        // si nos encontramos aquí, es porque el elemento posee ancho
        // y alto, pero además es un <tr>,
        // entonces se verifica el valor del estilo display
        // aplicado a través de CSS
        // para decidir si está oculto o no
        jQuery.curCSS(elem, "display") === "none";
};

jQuery.expr.filters.visible = function( elem ) {
    return !jQuery.expr.filters.hidden( elem );
};
```

Elección de Selectores

La elección de buenos selectores es un punto importante cuando se desea mejorar el rendimiento del código. Una pequeña especificidad — por ejemplo, incluir el tipo de elemento (como div) cuando se realiza una selección por el nombre de clase — puede ayudar bastante. Por eso, es recomendable darle algunas “pistas” a jQuery sobre en que lugar del documento puede encontrar lo que desea seleccionar. Por otro lado, demasiada especificidad puede ser perjudicial. Un selector como #miTabla thead tr th.especial es un exceso, lo mejor sería utilizar #miTabla th.especial.

jQuery ofrece muchos selectores basados en atributos, que permiten realizar selecciones basadas en el contenido de los atributos utilizando simplificaciones de expresiones regulares.

```
// encontrar todos los <a> cuyo atributo rel terminan en "thinger"
  $("a[rel$='thinger']");
```

Estos tipos de selectores pueden resultar útiles pero también ser muy lentos. Cuando sea posible, es recomendable realizar la selección utilizando IDs, nombres de clases y nombres de etiquetas.

Si desea conocer más sobre este asunto, Paul Irish realizó una gran presentación sobre mejoras de rendimiento en JavaScript (en ingles), la cual posee varias diapositivas centradas en selectores.

1.2.1 Comprobar Selecciones

Una vez realizada la selección de los elementos, querrá conocer si dicha selección entregó algún resultado. Para ello, pueda que escriba algo así:

```
if ($('div.foo')) { ... }
```

Sin embargo esta forma no funcionará. Cuando se realiza una selección utilizando $(), siempre es devuelto un objeto, y si se lo evalúa, éste siempre devolverá true. Incluso si la selección no contiene ningún elemento, el código dentro del bloque if se ejecutará.

En lugar de utilizar el código mostrado, lo que se debe hacer es preguntar por la cantidad de elementos que posee la selección que se ejecutó. Esto es posible realizarlo utilizando la propiedad JavaScript length. Si la respuesta es 0, la condición evaluará falso, caso contrario (más de 0 elementos), la condición será verdadera.

Evaluar si una selección posee elementos

```
if ($('div.foo').length) { ... }
```

1.2.2 Guardar Selecciones

Cada vez que se hace una selección, una gran cantidad de código es ejecutado. jQuery no guarda el resultado por si solo, por lo tanto, si va a realizar una selección que luego se hará de nuevo, deberá salvar la selección en una variable.

Guardar selecciones en una variable

```
var $divs = $('div');
```

Nota

En el ejemplo “Guardar selecciones en una variable”, la variable comienza con el signo de dólar. Contrariamente a otros lenguajes de programación, en JavaScript este signo no posee ningún significado especial — es solamente otro carácter. Sin embargo aquí se utilizará para indicar que dicha variable posee un objeto jQuery. Esta práctica — una especie de Notación Húngara — es solo una convención y no es obligatoria.

Una vez que la selección es guardada en la variable, se la puede utilizar en conjunto con los métodos de jQuery y el resultado será igual que utilizando la selección original.

Nota

La selección obtiene sólo los elementos que están en la página cuando se realizó dicha acción. Si luego se añaden elementos al documento, será necesario repetir la selección o añadir los elementos nuevos a la selección guardada en la variable. En otras palabras, las selecciones guardadas no se actualizan “mágicamente” cuando el DOM de modifica.

1.2.3 Refinamiento y Filtrado de Selecciones

A veces, puede obtener una selección que contiene más de lo que necesita; en este caso, es necesario refinar dicha selección. jQuery ofrece varios métodos para poder obtener exactamente lo que desea.

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
```

1.2.4 Selección de Elementos de un Formulario

jQuery ofrece varios pseudo-selectores que ayudan a encontrar elementos dentro de los formularios, éstos son especialmente útiles ya que dependiendo de los estados de cada elemento o su tipo, puede ser difícil distinguirlos utilizando selectores CSS estándar.

: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'

Utilizando pseudo-selectores en elementos de formularios

```
$('#myForm :input'); // obtiene todos los elementos inputs
                     // dentro del formulario #myForm
```

1.3 Trabajar con Selecciones

Una vez realizada la selección de los elementos, es posible utilizarlos en conjunto con diferentes métodos. éstos, generalmente, son de dos tipos: obtenedores (en inglés *getters*) y establecedores (en inglés *setters*). Los métodos obtenedores devuelven una propiedad del elemento seleccionado; mientras que los métodos establecedores fijan una propiedad a todos los elementos seleccionados.

1.3.1 Encadenamiento

Si en una selección se realiza una llamada a un método, y éste devuelve un objeto jQuery, es posible seguir un "encadenado" de métodos en el objeto.

Encadenamiento

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

Por otro lado, si se está escribiendo un encadenamiento de métodos que incluyen muchos pasos, es posible escribirlos línea por línea, haciendo que el código luzca más agradable para leer.

Formateo de código encadenado

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

Si desea volver a la selección original en el medio del encadenado, jQuery ofrece el método $.fn.end para poder hacerlo.

Restablecer la selección original utilizando el método $.fn.end

```
$('#content')
    .find('h3')
    .eq(2)
      .html('nuevo texto para el tercer elemento h3')
    .end() // reestablece la selección a todos los elementos h3 en #content
    .eq(0)
      .html('nuevo texto para el primer elemento h3');
```

Nota

El encadenamiento es muy poderoso y es una característica que muchas bibliotecas JavaScript han adoptado desde que jQuery se hizo popular. Sin embargo, debe ser utilizado con cuidado. Un encadenamiento de métodos extensivo pueden hacer un código extremadamente difícil de modificar y depurar. No existe una regla que indique que tan largo o corto debe ser el encadenado — pero es recomendable que tenga en cuenta este consejo.

1.3.2 Obtenedores (Getters) & Establecedores (Setters)

jQuery "sobrecarga" sus métodos, en otras palabras, el método para establecer un valor posee el mismo nombre que el método para obtener un valor. Cuando un método es utilizado para establecer un valor, es llamado método establecedor (en inglés *setter*). En cambio, cuando un método es utilizado para obtener (o leer) un valor, es llamado obtenedor (en inglés *getter*).

El método $.fn.html utilizado como establecedor

```
$('h1').html('hello world');
```

El método html utilizado como obtenedor

```
$('h1').html();
```

Los métodos establecedores devuelven un objeto jQuery, permitiendo continuar con la llamada de más métodos en la misma selección, mientras que los métodos obtenedores devuelven el valor por el cual se consultó, pero no permiten seguir llamando a más métodos en dicho valor.

1.4 CSS, Estilos, & Dimensiones

jQuery incluye una manera útil de obtener y establecer propiedades CSS a los elementos.

Nota

Las propiedades CSS que incluyen como separador un guión del medio, en JavaScript deben ser transformadas a su estilo *CamelCase*. Por ejemplo, cuando se la utiliza como propiedad de un método, el estilo CSS font-size deberá ser expresado como fontSize. Sin embargo, esta regla no es aplicada cuando se pasa el nombre de la propiedad CSS al método $.fn.css — en este caso, los dos formatos (en *CamelCase* o con el guión del medio) funcionarán.

Obtener propiedades CSS

```
$('h1').css('fontSize'); // devuelve una cadena de caracteres como "19px"
  $('h1').css('font-size'); // también funciona
```

Establecer propiedades CSS

```
$('h1').css('fontSize', '100px'); // establece una propiedad individual CSS
   $('h1').css({
      'fontSize' : '100px',
      'color' : 'red'
   }); // establece múltiples propiedades CSS
```

Notar que el estilo del argumento utilizado en la segunda línea del ejemplo — es un objeto que contiene múltiples propiedades. Esta es una forma común de pasar múltiples argumentos a una función, y muchos métodos establecedores de la biblioteca aceptan objetos para fijar varias propiedades de una sola vez.

A partir de la versión 1.6 de la biblioteca, utilizando $.fn.css también es posible establecer valores relativos en las propiedades CSS de un elemento determinado:

Establecer valores CSS relativos

```
$('h1').css({
      'fontSize' : '+=15px', // suma 15px al tamaño original del elemento
      'paddingTop' : '+=20px' // suma 20px al padding superior original del elemento
   });
```

1.4.1 Utilizar Clases para Aplicar Estilos CSS

Para obtener valores de los estilos aplicados a un elemento, el método $.fn.css es muy útil, sin embargo, su utilización como método establecedor se debe evitar (ya que, para aplicar estilos a un elemento, se puede hacer directamente desde CSS). En su lugar, lo ideal, es escribir reglas CSS que se apliquen a clases que describan los diferentes estados visuales de los elementos y luego cambiar la clase del elemento para aplicar el estilo que se desea mostrar.

Trabajar con clases

```
var $h1 = $('h1');

   $h1.addClass('big');
   $h1.removeClass('big');
   $h1.toggleClass('big');

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

Las clases también pueden ser útiles para guardar información del estado de un elemento, por ejemplo, para indicar que un elemento fue seleccionado.

1.4.2 Dimensiones

jQuery ofrece una variedad de métodos para obtener y modificar valores de dimensiones y posición de un elemento.

El código mostrado en el ejemplo “Métodos básicos sobre Dimensiones” es solo un breve resumen de las funcionalidades relaciones a dimensiones en jQuery; para un completo detalle puede consultar http://api.jquery.com/category/dimensions/.

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
```

1.5 Atributos

Los atributos de los elementos HTML que conforman una aplicación pueden contener información útil, por eso es importante poder establecer y obtener esa información.

El método $.fn.attr actúa tanto como método establecedor como obtenedor. Además, al igual que el método $.fn.css, cuando se lo utiliza como método establecedor, puede aceptar un conjunto de palabra clave-valor o un objeto conteniendo más conjuntos.

Establecer atributos

```
$('a').attr('href', 'allMyHrefsAreTheSameNow.html');
   $('a').attr({
      'title' : 'all titles are the same too',
      'href' : 'somethingNew.html'
   });
```

En el ejemplo, el objeto pasado como argumento está escrito en varias líneas. Como se explicó anteriormente, los espacios en blanco no importan en JavaScript, por lo cual, es libre de utilizarlos para hacer el código más legible. En entornos de producción, se pueden utilizar herramientas de minificación, los cuales quitan los espacios en blanco (entre otras cosas) y comprimen el archivo final.

Obtener atributos

```
$('a').attr('href');  // devuelve el atributo href perteneciente
                      // al primer elemento <a> del documento
```

1.6 Recorrer el DOM

Una vez obtenida la selección, es posible encontrar otros elementos utilizando a la misma selección.

En http://api.jquery.com/category/traversing/ puede encontrar una completa documentación sobre los métodos de recorrido de DOM (en inglés *traversing*) que posee jQuery.

Nota

Debe ser cuidadoso en recorrer largas distancias en un documento — recorridos complejos obligan que la estructura del documento sea siempre la misma, algo que es difícil de garantizar. Uno -o dos- pasos para el recorrido esta bien, pero generalmente hay que evitar atravesar desde un contenedor a otro.

Moverse a través del DOM utilizando métodos de recorrido

```
$('h1').next('p');            // seleccionar el inmediato y próximo
                              // elemento <p> con respecto a H1
   $('div:visible').parent();     // seleccionar el elemento contenedor
                              // a un div visible
```

```
$('input[name=first_name]').closest('form');  // seleccionar el elemento
                                  // <form> más cercano a un input
$('#myList').children();       // seleccionar todos los elementos
                        // hijos de #myList
$('li.selected').siblings();    // seleccionar todos los items
                        // hermanos del elemento <li>
```

También es posible interactuar con la selección utilizando el método $.fn.each. Dicho método interactúa con todos los elementos obtenidos en la selección y ejecuta una función por cada uno. La función recibe como argumento el índice del elemento actual y al mismo elemento. De forma predeterminada, dentro de la función, se puede hacer referencia al elemento DOM a través de la declaración this.

Interactuar en una selección

```
$('#myList li').each(function(idx, el) {
      console.log(
         'El elemento ' + idx +
         'contiene el siguiente HTML: ' +
         $(el).html()
      );
   });
```

1.7 Manipulación de Elementos

Una vez realizada la selección de los elementos que desea utilizar, “la diversión comienza”. Es posible cambiar, mover, remover y duplicar elementos. También crear nuevos a través de una sintaxis simple.

La documentación completa sobre los métodos de manipulación puede encontrarla en la sección Manipulation: http://api.jquery.com/category/manipulation/.

1.7.1 Obtener y Establecer Información en Elementos

Existen muchas formas por las cuales de puede modificar un elemento. Entre las tareas más comunes están las de cambiar el HTML interno o algún atributo del mismo. Para este tipo de tareas, jQuery ofrece métodos simples, funcionales en todos los navegadores modernos. Incluso es posible obtener información sobre los elementos utilizando los mismos métodos pero en su forma de método obtenedor.

Nota

Realizar cambios en los elementos, es un trabajo trivial, pero hay debe recordar que el cambio afectará a *todos* los elementos en la selección, por lo que, si desea modificar un sólo elemento, tiene que estar seguro de especificarlo en la selección antes de llamar al método establecedor.

Nota

Cuando los métodos actúan como obtenedores, por lo general, solamente trabajan con el primer elemento de la selección. Además no devuelven un objeto jQuery, por lo cual no es posible encadenar más métodos en el mismo. Una excepción es el método $.fn.text, el cual permite obtener el texto de los elementos de la selección.

$.fn.html

Obtiene o establece el contenido HTML de un elemento.

$.fn.text

Obtiene o establece el contenido en texto del elemento; en caso se pasarle como argumento código HTML, este es despojado.

$.fn.attr

Obtiene o establece el valor de un determinado atributo.

$.fn.width

Obtiene o establece el ancho en pixeles del primer elemento de la selección como un entero.

$.fn.height

Obtiene o establece el alto en pixeles del primer elemento de la selección como un entero.

$.fn.position

Obtiene un objeto con información sobre la posición del primer elemento de la selección, relativo al primer elemento padre posicionado. *Este método es solo obtenedor.*

$.fn.val

Obtiene o establece el valor (*value*) en elementos de formularios.

Cambiar el HTML de un elemento

```
$('#myDiv p:first')
    .html('Nuevo <strong>primer</strong> párrafo');
```

1.7.2 Mover, Copiar y Remover Elementos

Existen varias maneras para mover elementos a través del DOM; las cuales se pueden separar en dos enfoques:

- Querer colocar el/los elementos seleccionados de forma relativa a otro elemento
- Querer colocar un elemento relativo a el/los elementos seleccionados.

Por ejemplo, jQuery provee los métodos $.fn.insertAfter y $.fn.after. El método $.fn.insertAfter coloca a el/los elementos seleccionados después del elemento que se haya pasado como argumento; mientras que el método $.fn.after coloca al elemento pasado como argumento después del elemento seleccionado. Otros métodos también siguen este patrón: $.fn.insertBefore y $.fn.before; $.fn.appendTo y $.fn.append; y $.fn.prependTo y $.fn.prepend.

La utilización de uno u otro método dependerá de los elementos que tenga seleccionados y el tipo de referencia que se quiera guardar con respecto al elemento que se esta moviendo.

Mover elementos utilizando diferentes enfoques

```
// hacer que el primer item de la lista sea el último
  var $li = $('#myList li:first').appendTo('#myList');

  // otro enfoque para el mismo problema
```

```
$('#myList').append($('#myList li:first'));

// debe tener en cuenta que no hay forma de acceder a la
// lista de items que se ha movido, ya que devuelve
// la lista en sí
```

1.7.2.1 Clonar Elementos

Cuando se utiliza un método como $.fn.appendTo, lo que se está haciendo es mover al elemento; pero a veces en lugar de eso, se necesita mover un duplicado del mismo elemento. En este caso, es posible utilizar el método $.fn.clone.

Obtener una copia del elemento

```
// copiar el primer elemento de la lista y moverlo al final de la misma
$('#myList li:first').clone().appendTo('#myList');
```

Nota

Si se necesita copiar información y eventos relacionados al elemento, se debe pasar true como argumento de $.fn.clone.

1.7.2.2 Remover elementos

Existen dos formas de remover elementos de una página: Utilizando $.fn.remove o $.fn.detach. Cuando desee remover de forma permanente al elemento, utilize el método $.fn.remove. Por otro lado, el método $.fn.detach también remueve el elemento, pero mantiene la información y eventos asociados al mismo, siendo útil en el caso que necesite reinsertar el elemento en el documento.

Nota

El método $.fn.detach es muy útil cuando se esta manipulando de forma severa un elemento, ya que es posible eliminar al elemento, trabajarlo en el código y luego restaurarlo en la página nuevamente. Esta forma tiene como beneficio no tocar el DOM mientras se está modificando la información y eventos del elemento.

Por otro lado, si se desea mantener al elemento pero se necesita eliminar su contenido, es posible utiliza el método $.fn.empty, el cual “vaciará” el contenido HTML del elemento.

1.7.3 Crear Nuevos Elementos

jQuery provee una forma fácil y elegante para crear nuevos elementos a través del mismo método $() que se utiliza para realizar selecciones.

Crear nuevos elementos

```
$('<p>Un nuevo párrafo</p>');
$('<li class="new">nuevo item de la lista</li>');
```

Crear un nuevo elemento con atributos utilizando un objeto

```
$('<a/>', {
    html : 'Un <strong>nuevo</strong> enlace',
    'class' : 'new',
    href : 'foo.html'
```

```
});
```

Note que en el objeto que se pasa como argumento, la propiedad class está entre comillas, mientras que la propiedad href y html no lo están. Por lo general, los nombres de propiedades no deben estar entre comillas, excepto en el caso que se utilice como nombre una palabra reservada (como es el caso de class).

Cuando se crea un elemento, éste no es añadido inmediatamente a la página, sino que se debe hacerlo en conjunto con un método.

Crear un nuevo elemento en la página

```
var $myNewElement = $('<p>Nuevo elemento</p>');
  $myNewElement.appendTo('#content');

  $myNewElement.insertAfter('ul:last'); // eliminará al elemento <p>
                                        // existente en #content
  $('ul').last().after($myNewElement.clone());  // clonar al elemento <p>
                                                // para tener las dos versiones
```

Estrictamente hablando, no es necesario guardar al elemento creado en una variable — es posible llamar al método para añadir el elemento directamente después de $(). Sin embargo, la mayoría de las veces se deseará hacer referencia al elemento añadido, por lo cual, si se guarda en una variable no es necesario seleccionarlo después.

Crear y añadir al mismo tiempo un elemento a la página

```
$('ul').append('<li>item de la lista</li>');
```

Nota

La sintaxis para añadir nuevos elementos a la página es muy fácil de utilizar, pero es tentador olvidar que hay un costo enorme de rendimiento al agregar elementos al DOM de forma repetida. Si esta añadiendo muchos elementos al mismo contenedor, en lugar de añadir cada elemento uno por vez, lo mejor es concatenar todo el HTML en una única cadena de caracteres para luego anexarla al contenedor. Una posible solución es utilizar un vector que posea todos los elementos, luego reunirlos utilizando join y finalmente anexarla.

```
var myItems = [], $myList = $('#myList');

  for (var i=0; i<100; i++) {
    myItems.push('<li>item ' + i + '</li>');
  }

  $myList.append(myItems.join(''));
```

1.7.4 Manipulación de Atributos

Las capacidades para la manipulación de atributos que ofrece la biblioteca son extensos. La realización de cambios básicos son simples, sin embargo el método $.fn.attr permite manipulaciones más complejas.

Manipular un simple atributo

```
$('#myDiv a:first').attr('href', 'newDestination.html');
```

Manipular múltiples atributos

```
$('#myDiv a:first').attr({
      href : 'newDestination.html',
      rel : 'super-special'
   });
```

Utilizar una función para determinar el valor del nuevo atributo

```
$('#myDiv a:first').attr({
      rel : 'super-special',
      href : function(idx, href) {
         return '/new/' + href;
      }
   });

   $('#myDiv a:first').attr('href', function(idx, href) {
      return '/new/' + href;
   });
```

2 El núcleo de jQuery

2.1 $ vs $()

Hasta ahora, se ha tratado completamente con métodos que se llaman desde el objeto jQuery. Por ejemplo:

```
$('h1').remove();
```

Dichos métodos son parte del espacio de nombres (en inglés *namespace*) $.fn, o del prototipo (en inglés *prototype*) de jQuery, y son considerados como métodos del objeto jQuery.

Sin embargo, existen métodos que son parte del espacio de nombres de $ y se consideran como métodos del núcleo de jQuery.

Estas distinciones pueden ser bastantes confusas para usuarios nuevos. Para evitar la confusión, debe recordar estos dos puntos:

- Los métodos utilizados en selecciones se encuentran dentro del espacio de nombres $.fn, y automáticamente reciben y devuelven una selección en sí.
- Métodos en el espacio de nombres $ son generalmente métodos para diferentes utilidades, no trabajan con selecciones, no se les pasa ningún argumento y el valor que devuelven puede variar.

Existen algunos casos en donde métodos del objeto y del núcleo poseen los mismos nombres, como sucede con $.each y $.fn.each. En estos casos, debe ser cuidadoso de leer bien la documentación para saber que objeto utilizar correctamente.

2.2 Métodos Utilitarios

jQuery ofrece varios métodos utilitarios dentro del espacio de nombres $. Estos métodos son de gran ayuda para llevar a cabo tareas rutinarias de programación. A continuación se muestran algunos ejemplos, para una completa documentación sobre ellos, visite http://api.jquery.com/category/utilities/.

```
$.trim
        Remueve los espacios en blanco del principio y final.
$.trim('   varios espacios en blanco   ');
   // devuelve 'varios espacios en blanco'
$.each
```

Interactúa en vectores y objetos.

```
$.each([ 'foo', 'bar', 'baz' ], function(idx, val) {
    console.log('elemento ' + idx + 'es ' + val);
  });

  $.each({ foo : 'bar', baz : 'bim' }, function(k, v) {
    console.log(k + ' : ' + v);
  });
```

Nota

Como se dijo antes, existe un método llamado $.fn.each, el cual interactúa en una selección de elementos.

$.inArray

Devuelve el índice de un valor en un vector, o -1 si el valor no se encuentra en el vector.

```
var myArray = [ 1, 2, 3, 5 ];

  if ($.inArray(4, myArray) !== -1) {
    console.log('valor encontrado');
  }
```

$.extend

Cambia la propiedades del primer objeto utilizando las propiedades de los subsecuentes objetos.

```
var firstObject = { foo : 'bar', a : 'b' };
  var secondObject = { foo : 'baz' };

  var newObject = $.extend(firstObject, secondObject);
  console.log(firstObject.foo); // 'baz'
  console.log(newObject.foo);   // 'baz'
```

Si no se desea cambiar las propiedades de ninguno de los objetos que se utilizan en $.extend, se debe incluir un objeto vacío como primer argumento.

```
var firstObject = { foo : 'bar', a : 'b' };
  var secondObject = { foo : 'baz' };

  var newObject = $.extend({}, firstObject, secondObject);
  console.log(firstObject.foo); // 'bar'
  console.log(newObject.foo);   // 'baz'
```

$.proxy

Devuelve una función que siempre se ejecutará en el alcance (*scope*) provisto — en otras palabras, establece el significado de *this* (incluido dentro de la función) como el segundo argumento.

```
var myFunction = function() { console.log(this); };
  var myObject = { foo : 'bar' };

  myFunction(); // devuelve el objeto window

  var myProxyFunction = $.proxy(myFunction, myObject);
  myProxyFunction(); // devuelve el objeto myObject
```

Si se posee un objeto con métodos, es posible pasar dicho objeto y el nombre de un método para devolver una función que siempre se ejecuta en el alcance de dicho objeto.

```
var myObject = {
    myFn : function() {
      console.log(this);
    }
```

```
};

$('#foo').click(myObject.myFn); // registra el elemento DOM #foo
$('#foo').click($.proxy(myObject, 'myFn')); // registra myObject
```

2.3 Comprobación de Tipos

Como se mencionó en el capítulo "Conceptos Básicos de JavaScript", jQuery ofrece varios métodos útiles para determinar el tipo de un valor específico.

Comprobar el tipo de un determinado valor

```
var myValue = [1, 2, 3];

// Utilizar el operador typeof de JavaScript para comprobar tipos primitivos
typeof myValue == 'string'; // falso (false)
typeof myValue == 'number'; // falso (false)
typeof myValue == 'undefined'; // falso (false)
typeof myValue == 'boolean'; // falso (false)

// Utilizar el operador de igualdad estricta para comprobar valores nulos (null)
myValue === null; // falso (false)

// Utilizar los métodos jQuery para comprobar tipos no primitivos
jQuery.isFunction(myValue); // falso (false)
jQuery.isPlainObject(myValue); // falso (false)
jQuery.isArray(myValue); // verdadero (true)
jQuery.isNumeric(16); // verdadero (true). No disponible en versiones inferiores a jQuery 1.7
```

2.4 El Método Data

A menudo encontrará que existe información acerca de un elemento que necesita guardar. En JavaScript es posible hacerlo añadiendo propiedades al DOM del elemento, pero esta práctica conlleva enfrentarse a pérdidas de memoria (en inglés *memory leaks*) en algunos navegadores. jQuery ofrece una manera sencilla para poder guardar información relacionada a un elemento, y la misma biblioteca se ocupa de manejar los problemas que pueden surgir por falta de memoria.

Guardar y recuperar información relacionada a un elemento

```
$('#myDiv').data('keyName', { foo : 'bar' });
$('#myDiv').data('keyName'); // { foo : 'bar' }
```

A través del método $.fn.data es posible guardar cualquier tipo de información sobre un elemento. Es difícil exagerar la importancia de este concepto cuando se está desarrollando una aplicación compleja.

Por ejemplo, si desea establecer una relación entre el ítem de una lista y el div que hay dentro de este ítem, es posible hacerlo cada vez que se interactúa con el ítem, pero una mejor solución es hacerlo una sola vez, guardando un puntero al div utilizando el método $.fn.data:

Establecer una relación entre elementos utilizando el método $.fn.data

```
$('#myList li').each(function() {
    var $li = $(this), $div = $li.find('div.content');
    $li.data('contentDiv', $div);
});
```

```
// luego, no se debe volver a buscar al div;
// es posible leerlo desde la información asociada al item de la lista
var $firstLi = $('#myList li:first');
$firstLi.data('contentDiv').html('nuevo contenido');
```

Además es posible pasarle al método un objeto conteniendo uno o más pares de conjuntos palabra clave-valor.

A partir de la versión 1.5 de la biblioteca, jQuery permite utilizar al método $.fn.data para obtener la información asociada a un elemento que posea el atributo HTML5 data-*:

Elementos con el atributo data-*

```
<a id='foo' data-foo='baz' href='#'>Foo</a>

<a id='foobar' data-foo-bar='fol' href='#'>Foo Bar</a>
```

Obtener los valores asociados a los atributos data-* con $.fn.data

```
// obtiene el valor del atributo data-foo
// utilizando el método $.fn.data
console.log($('#foo').data('foo')); // registra 'baz'

// obtiene el valor del segundo elemento
console.log($('#foobar').data('fooBar')); // registra 'fol'
```

Nota

A partir de la versión 1.6 de la biblioteca, para obtener el valor del atributo data-foo-bar del segundo elemento, el argumento en $.fn.data se debe pasar en estilo *CamelCase*.

Nota

Para más información sobre el atributo HTML5 data-* visite http://www.w3.org/TR/html5/global-attributes.html#embedding-custom-non-visible-data-with-the-data-attributes.

2.5 Detección de Navegadores y Características

Más allá que jQuery elimine la mayoría de las peculiaridades de JavaScript entre cada navegador, existen ocasiones en que se necesita ejecutar código en un navegador específico.

Para este tipo de situaciones, jQuery ofrece el objeto $.support y $.browser (este último en desuso). Una completa documentación sobre estos objetos puede encontrarla en http://api.jquery.com/jQuery.support/ y http://api.jquery.com/jQuery.browser/

El objetivo de $.support es determinar qué características soporta el navegador web.

El objeto $.browser permite detectar el tipo de navegador y su versión. Dicho objeto está en desuso (aunque en el corto plazo no está planificada su eliminación del núcleo de la biblioteca) y se recomienda utilizar al objeto $.support para estos propósitos.

2.6 Evitar Conflictos con Otras Bibliotecas JavaScript

Si esta utilizando jQuery en conjunto con otras bibliotecas JavaScript, las cuales también utilizan la variable $, pueden llegar a ocurrir una serie de errores. Para poder solucionarlos, es necesario poner a jQuery en su modo "no-conflicto". Esto se debe realizar inmediatamente después que jQuery se cargue en la página y antes del código que se va a ejecutar.

Cuando se pone a jQuery en modo "no-conflicto", la biblioteca ofrece la opción de asignar un nombre para reemplazar a la variable $.

Poner a jQuery en modo no-conflicto

```
<script src="prototype.js"></script>          // la biblioteca prototype
                                              // también utiliza $
   <script src="jquery.js"></script>          // se carga jquery
                                              // en la página
   <script>var $j = jQuery.noConflict();</script>  // se inicializa
                                              // el modo "no-conflicto"
```

También es posible seguir utilizando $ conteniendo el código en una función anónima autoejecutable. Éste es un patrón estándar para la creación de extensiones para la biblioteca, ya que $ queda encerrada dentro del alcance de la misma función anónima.

Utilizar $ dentro de una función anónima autoejecutable

```
<script src="prototype.js"></script>
   <script src="jquery.js"></script>
   <script>
   jQuery.noConflict();

   (function($) {
     // el código va aquí, pudiendo utilizar $
   })(jQuery);
   </script>
```

3 Eventos

3.1 Introducción

jQuery provee métodos para asociar controladores de eventos (en inglés *event handlers*) a selectores. Cuando un evento ocurre, la función provista es ejecutada. Dentro de la función, la palabra clave this hace referencia al elemento en que el evento ocurre.

Para más detalles sobre los eventos en jQuery, puede consultar http://api.jquery.com/category/events/.

La función del controlador de eventos puede recibir un objeto. Este objeto puede ser utilizado para determinar la naturaleza del evento o, por ejemplo, prevenir el comportamiento predeterminado de éste. Para más detalles sobre el objeto del evento, visite http://api.jquery.com/category/events/event-object/.

3.2 Vincular Eventos a Elementos

jQuery ofrece métodos para la mayoría de los eventos — entre ellos $.fn.click, $.fn.focus, $.fn.blur, $.fn.change, etc. 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.

Vincular un evento utilizando un método reducido

```
$('p').click(function() {
      console.log('click');
   });
```

Vincular un evento utilizando el método $.fn.on

```
$('p').on('click', function() {
      console.log('click');
   });
```

Vincular un evento utilizando el método $.fn.on con información asociada

```
$('input').on(
      'click blur',  // es posible vincular múltiples eventos al elemento
      { foo : 'bar' }, // se debe pasar la información asociada como argumento

      function(eventObject) {
         console.log(eventObject.type, eventObject.data);
         // registra el tipo de evento y la información asociada { foo : 'bar' }
      }
   );
```

3.2.1 Vincular Eventos para Ejecutar una vez

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.

Cambiar controladores utilizando el método $.fn.one

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

El método $.fn.one es útil para situaciones en que necesita ejecutar cierto código la primera vez que ocurre un evento en un elemento, pero no en los eventos sucesivos.

3.2.2 Desvincular Eventos

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.

Desvincular todos los controladores del evento click en una selección

```
$('p').off('click');
```

Desvincular un controlador particular del evento click

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

```
$('p').on('click', foo).on('click', bar);
$('p').off('click', bar); // foo esta atado aún al evento click
```

3.2.3 Espacios de Nombres para Eventos

Cuando se esta desarrollando aplicaciones complejas o extensiones de jQuery, puede ser útil utilizar espacios de nombres para los eventos, y de esta forma evitar que se desvinculen eventos cuando no lo desea.

Asignar espacios de nombres a eventos

```
$('p').on('click.myNamespace', function() { /* ... */ });
$('p').off('click.myNamespace');
$('p').off('.myNamespace'); // desvincula todos los eventos con
                    // el espacio de nombre 'myNamespace'
```

3.2.4 Vinculación de Múltiples Eventos

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.

Vincular múltiples eventos a un elemento

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

3.3 El Objeto del Evento

Como se menciona en la introducción, la función controladora de eventos recibe un objeto del evento, el cual contiene varios métodos y propiedades. El objeto es comúnmente utilizado para prevenir la acción predeterminada del evento a través del método *preventDefault*. Sin embargo, también contiene varias propiedades y métodos útiles:

pageX, pageY

> La posición del puntero del ratón en el momento que el evento ocurrió, relativo a las zonas superiores e izquierda de la página.

type

> El tipo de evento (por ejemplo “click”).

which

> El botón o tecla presionada.

data

Alguna información pasada cuando el evento es ejecutado.

target

El elemento DOM que inicializó el evento.

preventDefault()

Cancela la acción predeterminada del evento (por ejemplo: seguir un enlace).

stopPropagation()

Detiene la propagación del evento sobre otros elementos.

Por otro lado, la función controladora también tiene acceso al elemento DOM que inicializó el evento a través de la palabra clave this. Para convertir a dicho elemento DOM en un objeto jQuery (y poder utilizar los métodos de la biblioteca) es necesario escribir $(this), como se muestra a continuación:

```
var $this = $(this);
```

Cancelar que al hacer click en un enlace, éste se siga

```
$('a').click(function(e) {
    var $this = $(this);
    if ($this.attr('href').match('evil')) {
      e.preventDefault();
      $this.addClass('evil');
    }
  });
```

3.4 Ejecución automática de Controladores de Eventos

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. 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.

Disparar un controlador de eventos de la forma correcta

```
var foo = function(e) {
    if (e) {
      console.log(e);
    } else {
      console.log('esta ejecucción no provino desde un evento');
    }
  };


  $('p').click(foo);
```

```
foo(); // en lugar de realizar $('p').trigger('click')
```

3.5 Incrementar el Rendimiento con la 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; si posee un gran conjunto de elementos, debería considerar utilizar la delegación de eventos a un elemento contenedor.

Nota

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.

Delegar un evento utilizando $.fn.on

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

Delegar un evento utilizando $.fn.delegate

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

3.5.1 Desvincular Eventos Delegados

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.

Desvincular eventos delegados

```
$('#myUnorderedList').off('click', 'li');
$('#myUnorderedList').undelegate('li', 'click');
```

3.6 Funciones Auxiliares de Eventos

jQuery ofrece dos funciones auxiliares para el trabajo con eventos:

3.6.1 $.fn.hover

El método $.fn.hover permite pasar una o dos funciones que se ejecutarán cuando los eventos mouseenter y mouseleave ocurran en el elemento seleccionado. Si se pasa una sola función, está será ejecutada en ambos

eventos; en cambio si se pasan dos, la primera será ejecutada cuando ocurra el evento mouseenter, mientras que la segunda será ejecutada cuando ocurra mouseleave.

Nota

A partir de la versión 1.4 de jQuery, el método requiere obligatoriamente dos funciones.

La función auxiliar hover

```
$('#menu li').hover(function() {
      $(this).toggleClass('hover');
   });
```

3.6.2 $.fn.toggle

Al igual que el método anterior, $.fn.toggle recibe dos o más funciones; cada vez que un evento ocurre, la función siguiente en la lista se ejecutará. Generalmente, $.fn.toggle es utilizada con solo dos funciones. En caso que utiliza más de dos funciones, tenga cuidado, ya que puede ser dificultar la depuración del código.

La función auxiliar toggle

```
$('p.expander').toggle(
      function() {
         $(this).prev().addClass('open');
      },
      function() {
         $(this).prev().removeClass('open');
      }
   );
```

4 Efectos

4.1 Introducción

Con jQuery, agregar efectos a una página es muy fácil. Estos efectos poseen una configuración predeterminada pero también es posible proveerles parámetros personalizados. Además es posible crear animaciones particulares estableciendo valores de propiedades CSS.

Para una completa documentación sobre los diferentes tipos de efectos puede visitar la sección effects: http://api.jquery.com/category/effects/.

4.2 Efectos Incorporados en la Biblioteca

Los efectos más utilizado ya vienen incorporados dentro de la biblioteca en forma de métodos:

$.fn.show

Muestra el elemento seleccionado.

$.fn.hide

Oculta el elemento seleccionado.

$.fn.fadeIn

De forma animada, cambia la opacidad del elemento seleccionado al 100%.

$.fn.fadeOut

De forma animada, cambia la opacidad del elemento seleccionado al 0

$.fn.slideDown

Muestra el elemento seleccionado con un movimiento de deslizamiento vertical.

$.fn.slideUp

Oculta el elemento seleccionado con un movimiento de deslizamiento vertical.

$.fn.slideToggle

Muestra o oculta el elemento seleccionado con un movimiento de deslizamiento vertical, dependiendo si actualmente el elemento está visible o no.

Uso básico de un efecto incorporado

```
$('h1').show();
```

4.2.1 Cambiar la Duración de los Efectos

Con la excepción de $.fn.show y $.fn.hide, todos los métodos tienen una duración predeterminada de la animación en 400ms. Este valor es posible cambiarlo.

Configurar la duración de un efecto

```
$('h1').fadeIn(300);     // desvanecimiento en 300ms
  $('h1').fadeOut('slow');  // utilizar una definición de velocidad interna
```

4.2.1.1 jQuery.fx.speeds

jQuery posee un objeto en jQuery.fx.speeds el cual contiene la velocidad predeterminada para la duración de un efecto, así como también los valores para las definiciones *"slow"* y *"fast"*.

```
speeds: {
      slow: 600,
      fast: 200,
      // velocidad predeterminada
      _default: 400
  }
```

Por lo tanto, es posible sobrescribir o añadir nuevos valores al objeto. Por ejemplo, puede que quiera cambiar el valor predeterminado del efecto o añadir una velocidad personalizada.

Añadir velocidades personalizadas a jQuery.fx.speeds

```
jQuery.fx.speeds.muyRapido = 100;
  jQuery.fx.speeds.muyLento = 2000;
```

4.2.2 Realizar una Acción Cuando un Efecto fue Ejecutado

A menudo, querrá ejecutar una acción una vez que la animación haya terminado — ya que si ejecuta la acción antes que la animación haya acabado, puede llegar a alterar la calidad del efecto o afectar a los elementos que forman parte de la misma. [Definición: *Las funciones de devolución de llamada* (en inglés *callback functions*) proveen una forma para ejecutar código una vez que un evento haya terminado.] En este caso, el evento que responderá a la función será la conclusión de la animación. Dentro de la función de devolución, la palabra clave this hace referencia al elemento en donde el efecto fue ejecutado y al igual que sucede con los eventos, es posible transformarlo a un objeto jQuery utilizando $(this).

Ejecutar cierto código cuando una animación haya concluido

```
$('div.old').fadeOut(300, function() { $(this).remove(); });
```

Note que si la selección no retorna ningún elemento, la función nunca se ejecutará. Este problema lo puede resolver comprobando si la selección devuelve algún elemento; y en caso que no lo haga, ejecutar la función de devolución inmediatamente.

Ejecutar una función de devolución incluso si no hay elementos para animar

```
var $thing = $('#nonexistent');

  var cb = function() {
    console.log('realizado');
  };

  if ($thing.length) {
    $thing.fadeIn(300, cb);
  } else {
    cb();
  }
```

4.3 Efectos Personalizados con $.fn.animate

Es posible realizar animaciones en propiedades CSS utilizando el método $.fn.animate. Dicho método permite realizar una animación estableciendo valores a propiedades CSS o cambiando sus valores actuales.

Efectos personalizados con $.fn.animate

```
$('div.funtimes').animate(
    {
      left : "+=50",
      opacity : 0.25
    },
    300, // duration
    function() { console.log('realizado'); // función de devolución de llamada
  });
```

Nota

Las propiedades relacionadas al color no pueden ser animadas utilizando el método $.fn.animate, pero es posible hacerlo a través de la extensión color plugin. Más adelante en el libro de discutirá la utilización de extensiones.

4.3.1 Easing

[Definición: El concepto de *Easing* describe la manera en que un efecto ocurre — es decir, si la velocidad durante la animación es constante o no.] jQuery incluye solamente dos métodos de easing: *swing* y *linear*. Si desea transiciones más naturales en las animaciones, existen varias extensiones que lo permiten.

A partir de la versión 1.4 de la biblioteca, es posible establecer el tipo de transición por cada propiedad utilizando el método $.fn.animate.

Transición de easing por cada propiedad

```
$('div.funtimes').animate(
    {
      left : [ "+=50", "swing" ],
      opacity : [ 0.25, "linear" ]
    },
    300
  );
```

Para más detalles sobre las opciones de easing, consulte http://api.jquery.com/animate/.

4.4 Control de los Efectos

jQuery provee varias herramientas para el manejo de animaciones.

$.fn.stop

Detiene las animaciones que se están ejecutando en el elemento seleccionado.

$.fn.delay

Espera un tiempo determinado antes de ejecutar la próxima animación.

```
$('h1').show(300).delay(1000).hide(300);
```

jQuery.fx.off

Si el valor es verdadero (*true*), no existirán transiciones para las animaciones; y a los elementos se le establecerá el estado final de la animación. Este método puede ser especialmente útil cuando se esta trabajando con navegadores antiguos.

5 Ajax

5.1 Introducción

El método *XMLHttpRequest* (XHR) permite a los navegadores comunicarse con el servidor sin la necesidad de recargar la página. Este método, también conocido como Ajax (*Asynchronous JavaScript and XML*), permite la creación de aplicaciones ricas en interactividad.

Las peticiones Ajax son ejecutadas por el código JavaScript, el cual envía una petición a una URL y cuando recibe una respuesta, una función de devolución puede ser ejecutada la cual recibe como argumento la respuesta del servidor y realiza algo con ella. Debido a que la respuesta es asíncrona, el resto del código de la aplicación continua ejecutándose, por lo cual, es imperativo que una función de devolución sea ejecutada para manejar la respuesta.

A través de varios métodos, jQuery provee soporte para Ajax, permitiendo abstraer las diferencias que pueden existir entre navegadores. Los métodos en cuestión son $.get(), $.getScript(), $.getJSON(), $.post() y $().load().

A pesar que la definición de Ajax posee la palabra “XML”, la mayoría de las aplicaciones no utilizan dicho formato para el transporte de datos, sino que en su lugar se utiliza HTML plano o información en formato JSON (*JavaScript Object Notation*).

En general, Ajax no trabaja a través de dominios diferentes. Sin embargo, existen excepciones, como los servicios que proveen información en formato JSONP (*JSON with Padding*), los cuales permiten una funcionalidad limitada a través de diferentes dominios.

5.2 Conceptos Clave

La utilización correcta de los métodos Ajax requiere primero la comprensión de algunos conceptos clave.

5.2.1 GET vs. POST

Los dos métodos HTTP más comunes para enviar una petición a un servidor son GET y POST. Es importante entender la utilización de cada uno.

El método GET debe ser utilizado para operaciones no-destructivas — es decir, operaciones en donde se esta “obteniendo” datos del servidor, pero no modificando. Por ejemplo, una consulta a un servicio de búsqueda podría ser una petición GET. Por otro lado, las solicitudes GET pueden ser almacenadas en la cache del navegador, pudiendo conducir a un comportamiento impredecible si no se lo espera. Generalmente, la información enviada al servidor, es enviada en una cadena de datos (en inglés *query string*).

El método POST debe ser utilizado para operaciones destructivas — es decir, operaciones en donde se está incorporando información al servidor. Por ejemplo, cuando un usuario guarda un artículo en un blog, esta acción debería utilizar POST. Por otro lado, este tipo de método no se guarda en la cache del navegador. Además, una cadena de datos puede ser parte de la URL, pero la información tiende a ser enviada de forma separada.

5.2.2 Tipos de Datos

Generalmente, jQuery necesita algunas instrucciones sobre el tipo de información que se espera recibir cuando se realiza una petición Ajax. En algunos casos, el tipo de dato es especificado por el nombre del método, pero en otros casos se lo debe detallar como parte de la configuración del método:

text

: Para el transporte de cadenas de caracteres simples.

html

: Para el transporte de bloques de código HTML que serán ubicados en la página.

script

: Para añadir un nuevo *script* con código JavaScript a la página.

json

: Para transportar información en formato JSON, el cual puede incluir cadenas de caracteres, vectores y objetos.

Nota

A partir de la versión 1.4 de la biblioteca, si la información JSON no está correctamente formateada, la petición podría fallar. Visite http://json.org para obtener detalles sobre un correcto formateo de datos en JSON.

Es recomendable utilizar los mecanismos que posea el lenguaje del lado de servidor para la generación de información en formato JSON.

jsonp

Para transportar información JSON de un dominio a otro.

xml

Para transportar información en formato XML.

A pesar de los diferentes tipos de datos de que se puede utilizar, es recomendable utilizar el formato JSON, ya que es muy flexible, permitiendo por ejemplo, enviar al mismo tiempo información plana y HTML.

5.2.3 Asincronismo

Debido a que, de forma predeterminada, las llamadas Ajax son asíncronas, la respuesta del servidor no esta disponible de forma inmediata. Por ejemplo, el siguiente código no debería funcionar:

```
var response;
  $.get('foo.php', function(r) { response = r; });
  console.log(response); // indefinido (undefined)
```

En su lugar, es necesario especificar una función de devolución de llamada; dicha función se ejecutará cuando la petición se haya realizado de forma correcta ya que es en ese momento cuando la respuesta del servidor esta lista.

```
$.get('foo.php', function(response) { console.log(response); });
```

5.2.4 Políticas de Mismo Origen y JSONP

En general, las peticiones Ajax están limitadas a utilizar el mismo protocolo (*http* o *https*), el mismo puerto y el mismo dominio de origen. Esta limitación no se aplica a los scripts cargados a través del método Ajax de jQuery.

La otra excepción es cuando se hace una petición que recibirá una respuesta en formato JSONP. En este caso, el proveedor de la respuesta debe responder la petición con un script que puede ser cargado utilizando la etiqueta <script>, evitando así la limitación de realizar peticiones desde el mismo dominio. Dicha respuesta contendrá la información solicitada, contenida en una función

5.2.5 Ajax y Firebug

Firebug (o el inspector WebKit que viene incluido en Chrome o Safari) son herramientas imprescindibles para trabajar con peticiones Ajax, ya que es posible observarlas desde la pestaña Consola de Firebug (o yendo a Recursos > Panel XHR desde el inspector de Webkit) y revisar los detalles de dichas peticiones. Si algo esta fallando cuando trabaja con Ajax, este es el primer lugar en donde debe dirigirse para saber cual es el problema.

5.3 Métodos Ajax de jQuery

Como se indicó anteriormente, jQuery posee varios métodos para trabajar con Ajax. Sin embargo, todos están basados en el método $.ajax, por lo tanto, su comprensión es obligatoria. A continuación se abarcará dicho método y luego se indicará un breve resumen sobre los demás métodos.

Generalmente, es preferible utilizar el método $.ajax en lugar de los otros, ya que ofrece más características y su configuración es muy comprensible.

5.3.1 $.ajax

El método $.ajax es configurado a través de un objeto, el cual contiene todas las instrucciones que necesita jQuery para completar la petición. Dicho método es particularmente útil debido a que ofrece la posibilidad de especificar acciones en caso que la petición haya fallado o no. Además, al estar configurado a través de un objeto, es posible definir sus propiedades de forma separada, haciendo que sea más fácil la reutilización del código. Puede visitar http://api.jquery.com/jQuery.ajax/ para consultar la documentación sobre las opciones disponibles en el método.

Utilizar el método $.ajax

```
$.ajax({
    // la URL para la petición
    url : 'post.php',

    // la información a enviar
    // (también es posible utilizar una cadena de datos)
    data : { id : 123 },

    // especifica si será una petición POST o GET
    type : 'GET',

    // el tipo de información que se espera de respuesta
    dataType : 'json',

    // código a ejecutar si la petición es satisfactoria;
    // la respuesta es pasada como argumento a la función
    success : function(json) {
       $('<h1/>').text(json.title).appendTo('body');
       $('<div class="content"/>')
          .html(json.html).appendTo('body');
    },

    // código a ejecutar si la petición falla;
    // son pasados como argumentos a la función
    // el objeto jqXHR (extensión de XMLHttpRequest), un texto con el estatus
    // de la petición y un texto con la descripción del error que haya dado el servidor
    error : function(jqXHR, status, error) {
       alert('Disculpe, existió un problema');
    },

    // código a ejecutar sin importar si la petición falló o no
    complete : function(jqXHR, status) {
       alert('Petición realizada');
    }
  });
```

Nota

Una aclaración sobre el parámetro dataType: Si el servidor devuelve información que es diferente al formato especificado, el código fallará, y la razón de porque lo hace no siempre quedará clara debido a que la respuesta HTTP no mostrará ningún tipo de error. Cuando esté trabajando con peticiones Ajax, debe estar seguro que el servidor esta enviando el tipo de información que esta solicitando y verifique que la cabecera Content-type es exacta al tipo de dato. Por ejemplo, para información en formato JSON, la cabecera Content-type debería ser application/json.

5.3.1.1 Opciones del método $.ajax

El método $.ajax posee muchas opciones de configuración, y es justamente esta característica la que hace que sea un método muy útil. Para una lista completa de las opciones disponibles, puede consultar http://api.jquery.com/jQuery.ajax/; a continuación se muestran las más comunes:

async

> Establece si la petición será asíncrona o no. De forma predeterminada el valor es true. Debe tener en cuenta que si la opción se establece en false, la petición bloqueará la ejecución de otros códigos hasta que dicha petición haya finalizado.

cache

> Establece si la petición será guardada en la cache del navegador. De forma predeterminada es true para todos los *dataType* excepto para "*script*" y "*jsonp*". Cuando posee el valor false, se agrega una cadena de caracteres anti-cache al final de la URL de la petición.

complete

> Establece una función de devolución de llamada que se ejecuta cuando la petición esta completa, aunque haya fallado o no. La función recibe como argumentos el objeto jqXHR (en versiones anteriores o iguales a jQuery 1.4, recibe en su lugar el objeto de la petición en crudo XMLHTTPRequest) y un texto especificando el estatus de la misma petición (success, notmodified, error, timeout, abort, o parsererror).

context

> Establece el alcance en que la/las funciones de devolución de llamada se ejecutaran (por ejemplo, define el significado de this dentro de las funciones). De manera predeterminada this hace referencia al objeto originalmente pasado al método $.ajax.

data

> Establece la información que se enviará al servidor. Esta puede ser tanto un objeto como una cadena de datos (por ejemplo foo=bar&baz=bim.)

dataType

> Establece el tipo de información que se espera recibir como respuesta del servidor. Si no se especifica ningún valor, de forma predeterminada, jQuery revisa el tipo de *MIME* que posee la respuesta.

error

> Establece una función de devolución de llamada a ejecutar si resulta algún error en la petición. Dicha función recibe como argumentos el objeto jqXHR (en versiones anteriores o iguales a jQuery 1.4, recibe en su lugar el objeto de la petición en crudo XMLHTTPRequest), un texto especificando el

estatus de la misma petición (timeout, error, abort, o parsererror) y un texto con la descripción del error que haya enviado el servidor (por ejemplo Not Found o Internal Server Error).

jsonp

Establece el nombre de la función de devolución de llamada a enviar cuando se realiza una petición *JSONP*. De forma predeterminada el nombre es "*callback*

success

Establece una función a ejecutar si la petición a sido satisfactoria. Dicha función recibe como argumentos el objeto jqXHR (en versiones anteriores o iguales a jQuery 1.4, recibe en su lugar el objeto de la petición en crudo XMLHTTPRequest), un texto especificando el estatus de la misma petición y la información de la petición (convertida a objeto JavaScript en el caso que *dataType* sea *JSON*), el estatus de la misma.

timeout

Establece un tiempo en milisegundos para considerar a una petición como fallada.

traditional

Si su valor es true, se utiliza el estilo de serialización de datos utilizado antes de jQuery 1.4. Para más detalles puede visitar http://api.jquery.com/jQuery.param/.

type

De forma predeterminada su valor es “GET”. Otros tipos de peticiones también pueden ser utilizadas (como PUT y DELETE), sin embargo pueden no estar soportados por todos los navegadores.

url

Establece la URL en donde se realiza la petición.

La opción url es obligatoria para el método $.ajax;

Como se comentó anteriormente, para una lista completa de las opciones disponibles, puede consultar http://api.jquery.com/jQuery.ajax/.

Nota

A partir de la versión 1.5 de jQuery, las opciones beforeSend, success, error y complete reciben como uno de sus argumentos el objeto jqXHR siendo este una extensión del objeto nativo XMLHTTPRequest. El objeto jqXHR posee una serie de métodos y propiedades que permiten modificar u obtener información particular de la petición a realizar, como por ejemplo sobreescribir el tipo de *MIME* que posee la respuesta que se espera por parte del servidor. Para información sobre el objeto jqXHR puede consultar http://api.jquery.com/jQuery.ajax/#jqXHR.

Nota

A partir de la versión 1.5 de jQuery, las opciones success, error y complete pueden recibir un vector con varias funciones de devolución, las cuales serán ejecutadas en turnos.

5.3.2 Métodos Convenientes

En caso que no quiera utilizar el método $.ajax, y no necesite los controladores de errores, existen otros métodos más convenientes para realizar peticiones Ajax (aunque, como se indicó antes, estos están basados el método $.ajax con valores pre-establecidos de configuración).

Los métodos que provee la biblioteca son:

$.get

Realiza una petición GET a una URL provista.

$.post

Realiza una petición POST a una URL provista.

$.getScript

Añade un script a la página.

$.getJSON

Realiza una petición GET a una URL provista y espera que un dato JSON sea devuelto.

Los métodos deben tener los siguientes argumentos, en orden:

url

La URL en donde se realizará la petición. Su valor es obligatorio.

data

La información que se enviará al servidor. Su valor es opcional y puede ser tanto un objeto como una cadena de datos (como foo=bar&baz=bim).

Nota

Esta opción no es valida para el método $.getScript.

success callback

Una función opcional que se ejecuta en caso que petición haya sido satisfactoria. Dicha función recibe como argumentos la información de la petición y el objeto en bruto de dicha petición.

data type

El tipo de dato que se espera recibir desde el servidor. Su valor es opcional.

Nota

Esta opción es solo aplicable para métodos en que no está especificado el tipo de dato en el nombre del mismo método.

Utilizar métodos convenientes para peticiones Ajax

```
// obtiene texto plano o html
   $.get('/users.php', { userId : 1234 }, function(resp) {
      console.log(resp);
   });

   // añade un script a la página y luego ejecuta la función especificada
   $.getScript('/static/js/myScript.js', function() {
      functionFromMyScript();
   });

   // obtiene información en formato JSON desde el servidor
   $.getJSON('/details.php', function(resp) {
      $.each(resp, function(k, v) {
         console.log(k + ' : ' + v);
      });
   });
```

5.3.3 $.fn.load

El método $.fn.load es el único que se puede llamar desde una selección. Dicho método obtiene el código HTML de una URL y rellena a los elementos seleccionados con la información obtenida. En conjunto con la URL, es posible especificar opcionalmente un selector, el cual obtendrá el código especificado en dicha selección.

Utilizar el método $.fn.load para rellenar un elemento

```
$('#newContent').load('/foo.html');
```

Utilizar el método $.fn.load para rellenar un elemento basado en un selector

```
$('#newContent').load('/foo.html #myDiv h1:first', function(html) {
     alert('Contenido actualizado');
   });
```

5.4 Ajax y Formularios

Las capacidades de jQuery con Ajax pueden ser especialmente útiles para el trabajo con formularios. Por ejemplo, la extensión jQuery Form Plugin es una extensión para añadir capacidades Ajax a formularios. Existen dos métodos que debe conocer para cuando este realizando este tipo de trabajos: $.fn.serialize y $.fn.serializeArray.

Transformar información de un formulario a una cadena de datos

```
$('#myForm').serialize();
```

Crear un vector de objetos conteniendo información de un formulario

```
$('#myForm').serializeArray();

   // crea una estructura como esta:
   [
      { name : 'field1', value : 123 },
      { name : 'field2', value : 'hello world' }
   ]
```

5.5 Trabajar con JSONP

En los últimos tiempos, la introducción de JSONP, ha permitido la creación de aplicaciones híbridas de contenidos. Muchos sitios importantes ofrecen JSONP como servicio de información, el cual se accede a través de una API (en inglés *Application programming interface*) predefinida. Un servicio particular que permite obtener información en formato JSONP es Yahoo! Query Language, el cual se utiliza a continuación para obtener, por ejemplo, noticias sobre gatos:

Utilizar YQL y JSONP

```
$.ajax({
    url : 'http://query.yahooapis.com/v1/public/yql',

    // se agrega como parámetro el nombre de la función de devolución,
    // según se especifica en el servicio de YQL
    jsonp : 'callback',

    // se le indica a jQuery que se espera información en formato JSONP
    dataType : 'jsonp',

    // se le indica al servicio de YQL cual es la información
    // que se desea y que se la quiere en formato JSON
    data : {
      q : 'select title,abstract,url from search.news where query="cat"',
      format : 'json'
    },

    // se ejecuta una función al ser satisfactoria la petición
    success : function(response) {
      console.log(response);
    }
  });
```

jQuery se encarga de solucionar todos los aspectos complejos de la petición JSONP. Lo único que debe hacer es especificar el nombre de la función de devolución (en este caso "*callback*", según lo especifica YQL) y el resultado final será como una petición Ajax normal.

5.6 Eventos Ajax

A menudo, querrá ejecutar una función cuando una petición haya comenzado o terminado, como por ejemplo, mostrar o ocultar un indicador. En lugar de definir estas funciones dentro de cada petición, jQuery provee la posibilidad de vincular eventos Ajax a elementos seleccionados. Para una lista completa de eventos Ajax, puede consultar http://docs.jquery.com/Ajax_Events.

Mostrar/Ocultar un indicador utilizando Eventos Ajax

```
$('#loading_indicator')
    .ajaxStart(function() { $(this).show(); })
    .ajaxStop(function() { $(this).hide(); });
```

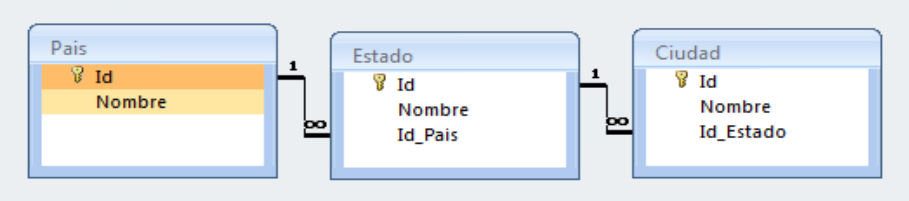
EJERCICIOS

1. Preparar un formulario, con los siguientes campos:
   - Nombre, Cedula, Dirección, Teléfono, Sexo, Ciudad, Nacionalidad.
   - Validar el Formulario, entendiendo que todos los campos son obligatorios.
2. Realizar y validar:
   - Un Formulario Con un Campo Tipo Lista Menú de nombre Sexo de 2 opciones, Masculino y Femenino.
   - Según la opción que escoja el visitante, se desplegara otros campos
   - Para Masculino: Cerveza Favorita y Deporte Favorito.
   - Para Femenino: Color Favorito y Género musical.
   - Si el usuario no escoge, todo permanece bloqueado, si escoge masculino habilita la sección de campos de masculino, si escoge femenino bloque las otras secciones y habilita visualmente los campos para femenino.
3. Envió por Ajax del Formulario del Ejercicio1
4. Desarrollar 3 listas menú, las 3 deben estar asociadas y trabajadas con AJAX, cada una realizara una consulta SQL para mostrar los datos.

Se recomienda:

- Crear una Base de Datos (CursoAjax), de 3 Tablas (País, Estado, Ciudad), la misma tendrá la siguiente estructura.

Imagen1



PAIS	ESTADO	CIUDAD	CAMPOS
Id	Id	Id	int(11) PK AI
Nombre	Nombre	Nombre	varchar(60)
	Id_Pais	Id_Estado	int(11)

- Crear Archivo de Conexión
- Crear 1 Archivo Javascript, donde colocaremos funciones Jquery que nos ayudaran a trabajar con Ajax.
- Se puede trabajar la estructura con 2 páginas en PHP, o se puede hacer paso a paso trabajando con 1 página por cada lista, en este caso 3 páginas.

Imagen2

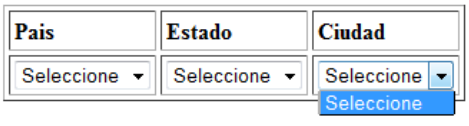
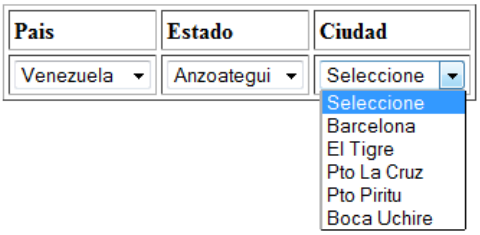


Imagen3



5. Ejercicio3 formulario enviado por AJAX, insertarlo en una base de datos mediante PHP.

   Crear Tabla Usuario, con los respectivos 8 Campos del formulario y Utilizar las estructuras de conexión a la base de datos creadas en el ejercicio N° 4, Adicional vamos a mostrar el listado de registros cargados en la tabla sin recargar la pagina.

BIBLIOGRAFIA

LINKS DE AYUDA

http://jquery.com/
http://web.tursos.com/tutoriales/javascript/como-hacer-un-formulario-de-contacto-ii-validar-con-jquery
http://www.masquewordpress.com/selects-anidados-con-ajax-y-jquery/#
http://www.miguelmanchego.com/2009/ajax-enviar-formularios-sin-recargar-jquery/
http://www.ajaxshake.com/es/JS/235871/banner-animado-y-customizado-con-jquery-animationbanner.html
http://www.infrabios.com/index.php?topic=5637.0
http://jonraasch.com/blog/a-simple-jquery-slideshow
http://programarivm.com/2012/04/un-acordeon-anidado-con-jquery-ui/
http://www.tutorialjquery.com/acordeon-menu-jquery-muy-facil-de-implementar/
http://www.actualidadjquery.es/2010/10/26/comprobar-si-un-checkbox-esta-checked-con-jquery/
http://www.arteyanos.com/2012/02/validar-radio-buttons-con-jquery/