Sintaxis Básica © ADRINFOR S.L.

www.adrformacion.com © ADRINFOR S.L.
Héctor García González
www.adrformacion.com © ADRINFOR S.L.
Héctor García González

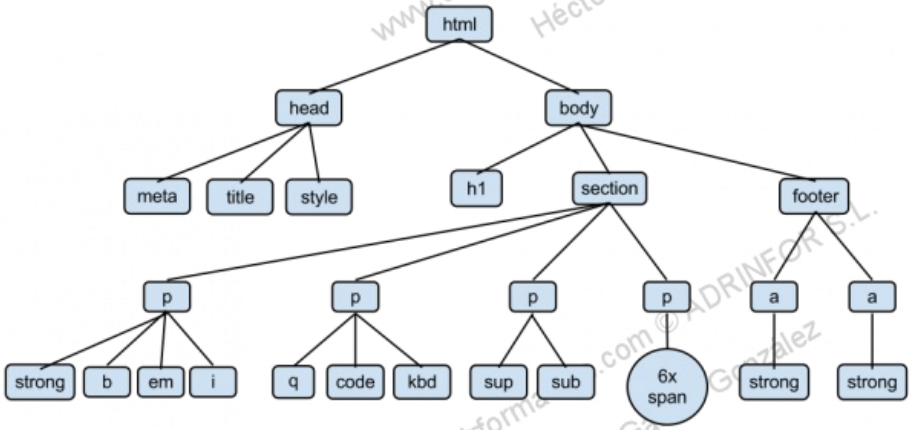
Indice

Sintaxis Básica de jQuery

Sintaxis Básica

Una vez incluida la librería de jQuery en nuestro archivo HTML ya dispondremos del entorno necesario para hacer uso de la misma.

Dado que el funcionamiento de jQuery está en gran parte vinculado con el DOM, ejecutaremos el código jQuery una vez la página haya sido cargada.



Interpretación gráfica de los elementos del DOM de una página

¿Por qué es necesario que el DOM haya sido cargado?

Puede parecer muy restrictivo que la librería nos obligue a esperar a que el DOM de la página haya sido cargada. Sin embargo,es fácil de explicar el porqué de esta susceptibilidad de la librería. Como veremos más adelante, jQuery hace uso de multitud de propiedades de todos los elementos del DOM de una página.

Los motores internos de los navegadores no establecen estas propiedades a los elementos del DOM directamente, sino que se les asigna cuando las hojas de estilo CSS han sido procesadas y aplicadas a la página y sus elementos.

Por ello, y dado que jQuery va a consultar cientos de propiedades de los objetos del DOM es necesario que los elementos existentes dentro de una página hayan sido renderizados, así como que las propiedades aplicadas sobre cada uno de ellos por las hojas de estilo se hayan aplicado correctamente para que sus interacciones y consultas sobre estos elementos sean verídicas y funcionen de forma estable.

Esperar a que el documento haya sido renderizado

Si anteriormente has programado en JavaScript es bastante probable que en algún momento hayas necesitado ejecutar un fragmento de código justo en el momento que la página había sido cargada. Para ello, se podía hacer uso del evento "onLoad" del elemento "body"

```
<body onload="JavaScript:alert('hola mundo');">
```

Sin embargo, este evento se lanza cuando la página se ha cargado completamente.

En otras palabras, este evento involucra no sólo la carga del documento HTML y su interpretación para formar el árbol DOM, sino que también espera a que se complete la carga de imágenes y otros recursos necesarios para mostrar la página.

A pesar de ello, el código JavaScript comienza a ejecutarse antes de que se complete la carga del DOM.

Entonces, ¿cómo podemos asegurarnos de que nuestro código jQuery se ejecuta cuando los elementos del DOM se hayan cargado?

jQuery nos ofrece diferentes métodos para capturar este evento, todos ellos completamente válidos y con resultados idénticos, que tendremos que utilizar para retrasar la ejecución de nuestro código hasta el momento adecuado.

Capturando el evento "ready" del documento

La función $(document).ready(funcion) es el primer código que cualquier usuario de la librería debería conocer. Mediante el uso de esta función, se delega la ejecución del código "envuelto" por la misma, para que éste sea ejecutado solamente cuando todos los elementos del DOM hayan sido cargados, sin necesidad de esperar a la descarga de los demás recursos solicitados por la página.

Cualquier código que escribamos y que haga uso de la librería tendrá que incluirse dentro de este u otro de los métodos de delegación que nos ofrece jQuery.

Ejemplo

En este primer ejemplo, vamos a hacer uso de la función$(document).ready(funcion) para que una vez el DOM se encuentre disponible, se nos muestre un mensaje de alerta.

Para ello, necesitamos una página básica que incorpore la librería:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Documento sin título</title>
<script type="text/javascript" src="js/jquery-1.8.2.min.js"></script>
</head>

<body>
</body>
</html>
```

Este código generará una página que, pese a no tener ningún contenido (el "body" del documento se encuentra vacío), solicita la carga de la versión 1.8.2 de la librería.

Con la librería jQuery disponible en nuestra página podremos hacer uso de la función $(document).ready(funcion) , con la siguiente sintaxis:

```
 ...
<script type="text/javascript" src="js/jquery-1.8.2.min.js"></script>
<script type="text/javascript">
$(document).ready(function(){
//código que queremos ejecutar
});
</script>
</head>
...
```

Si intentamos traducir este código a "lenguaje humano", podríamos leer: "Cuando el documento esté preparado, ejecuta la función...". En este punto, aún no hemos definido cuál es el "código que queremos ejecutar". De momento, vamos a mostrar una alerta con el mensaje "el DOM ya está disponible".

```
 ...
<script type="text/javascript">
$(document).ready(function(){
//código que queremos ejecutar
 alert("el DOM ya está disponible!");
});
</script>
</head>
...
```

La sintaxis de jQuery puede suponer un cambio importante para cualquier persona acostumbrada a programar en JavaScript .

Por otro lado, aquellas personas que se encuentre familiarizada con las hojas de estilo CSS y los selectores utilizados en éstas, descubrirán que estos conocimientos van a ser muy útiles al utilizar la librería jQuery.

Una vez nos dispongamos a escribir nuestro código (o leamos código de cualquier otra persona) es posible encontrarse llamadas a este método de las siguientes dos formas.

```
 ...
<script type="text/javascript">
jQuery(document).ready( function(){
//código que queremos ejecutar
 alert("el DOM ya está disponible!");
});
</script>
</head>
...
```

```
 ...
<script type="text/javascript">
$(document).ready( function(){
//código que queremos ejecutar
 alert("el DOM ya está disponible!");
});
</script>
</head>
...
```

DOM Ready

De momento simplemente comentaremos que las dos llamadas son idénticas. Más adelante veremos la explicación de porqué podemos utilizar estas dos sintaxis para realizar la misma acción.

Acostúmbrate a estos métodos.
Es INDISPENSABLE que te acostumbres a envolver tu código dentro de estas funciones.
Acostúmbrate a ello y evitarás problemas en tus scripts.

Utilizar el "atajo" $(function())

Otro método de esperar a la carga completa del DOM antes de que el código sea procesado es mediante el uso del atajo $(function()).

Al igual que con la función $(document).ready(funcion), con este método aseguraremos que el código contenido dentro del "atajo" será ejecutado tras la carga de DOM.

```
 ...
<script type="text/javascript">
$(function(){
//código que queremos ejecutar
 alert("el DOM ya está disponible!");
});
</script>
</head>
...
```

Este código es idéntico en funcionamiento al que el que encontramos en el apartado anterior. El uso de uno u otro tipo de sintaxis es libre para cada desarrollados, pudiendo hacer uso indistintamente del método con el que cada uno se encuentre más cómodo.

La función jQuery() o su alias $()

Al incluir jQuery en nuestra página podremos hacer uso de la función jQuery() a través de la cual se podrán utilizar todos los recursos de la librería. Puede parecer demasiado simple que incluir jQuery en una página solamente añada esta función. Sin embargo, hay que aclarar que esta función incorpora cientos de métodos con los que podremos realizar maravillas en nuestras páginas con muy poco código.

Sin embargo, en el uso cotidiano de la librería, resulta tedioso repetir continuamente la llamada a la función jQuery(). Para ello, la librería proporciona un método de acceso rápido - o alias - con el que hacer referencia a la librería. Este alias es el símbolo $.

Este símbolo, fácilmente localizable en el código, hace que nuestro código sea más legible y las llamadas a la librería son más fácilmente localizables entre el código JavaScript estándar.

Existe una situación en la que no podremos utilizar el alias $: si en una página incluimos otra librería que haga uso del carácter $.

Esta situación es llamada "conflicto". Pese a ello, jQuery dispone de opciones con las que poder salvar este obstáculo.

1. Uso de la sintaxis extendida.
   Cuando encontremos una situación de conflicto con otra librería, podremos hacer uso directo de la función jQuery() en vez de utilizar el alias $().
2. Uso de un alias propio
   Otra opción es definir un alias propio con el que referenciar a la función jQuery().

   ```
   $j = jQuery;
   ```

jQuery() y su alias $()

Sintaxis Básica de comandos jQuery

Cuando programamos utilizando jQuery, la perspectiva desde la que debemos afrontar los problemas es diferente a la que se utiliza cuando se programa código JavaScript estándar.

El principal concepto que hemos de tener en cuenta cuando utilicemos jQuery es la idea deencuentra algo, haz algo.

La explicación a esta afirmación en un aspecto puramente práctico es, primero selecciona un conjunto de elementos del DOM de un documento HTML y sobre ellos ejecuta una o varias acciones (consulta sus propiedades, modifica sus estilos, y un largo etcétera de posibilidades) por medio de los métodos que te ofrece jQuery.

Estructura de un comando de jQuery

Siempre que vayamos a hacer uso de la librería jQuery, deberemos utilizar una sintaxis estructurada, dividida en fragmentos fácilmente diferenciables:

- Función jQuery
- Un selector como parámetro
- Uno o varios métodos de la librería
- Opcionalmente, parámetros del método.

Vamos a analizar la estructura de dos comandos jQuery básicos, tanto en la llamada directa a la función jQuery como la llamada por medio del alias $. Los comandos que vamos a analizar son los siguientes:

```
 <script type="text/javascript">
$(document).ready( function(){
   $("p") .css ("background-color","red");
   jQuery("p") .css ("background-color","red");

   $("input") .attr ("value","25");
   jQuery("input") .attr ("value","25");
});
</script>
```

Función jQuery

El primer fragmento de un comandos jQuery será una llamada a la función jQuery o el alias $. Cualquiera de las dos opciones es válida, pero generalmente, utilizaremos el alias por dos motivos:

- requiere menor cantidad de código
- "salta" mucho más a la vista, siendo muy útil para diferenciarlo del código JavaScript estándar.

Si observamos nuestro fragmento de código, observamos que en cada uno de los comandos hacemos uso de uno u otro método para llamar a la librería:

```
 ...
   $("p") .css ("background-color","red");
   jQuery("p") .css ("background-color","red");

   $("input") .attr ("value","25");
   jQuery("input") .attr ("value","25");
...
```

Un selector

Al hacer la llamada a la función jQuery, deberemos proporcionarle un parámetro que permita indicarle a la librería los elementos de la página sobre los que actuará. Este parámetro es conocido como "selector", una cadena de texto que indicará a la librería el conjunto de elementos del DOM sobre los que aplicar la acción indicada.

El uso de selectores dota a jQuery de una increíble potencia a la hora de acotar los elementos sobre los que vamos a interactuar. Una vez domines y perfecciones el uso de esta técnica, conseguirás ejecutar comandos tan precisos como sean necesarios.

Volvamos al fragmento de código que estamos analizando. En este caso son dos los selectores utilizados:

- El selector "p"
- El selector "input"

Los selectores utilizados son bastante sencillos. En los dos primeros comandos, el selector obtendrá los elementos "p" de la página, y los dos segundos, los elementos de tipo "input".

```
...
   $(''p'') .css ("background-color","red");
   jQuery(''p'') .css ("background-color","red");

   $(''input'') .attr ("value","25");
   jQuery(''input'') .attr ("value","25");
...
```

Al definir un selector, podremos utilizar comillas simples -$(**'input'**) - o comillas dobles - $(**"input"**) -. Cualquiera de las dos sintaxis será válida.

Un método de la librería

El siguiente paso es indicar la acción que vamos a realizar sobre cada uno de los elementos seleccionados. Existe una gran cantidad de métodos soportados por la librería: show , hide , css , hasClass , addClass ...

Una vez más, vamos a observar el código de nuestro ejemplo.

En los dos primeros comandos, utilizamos el método "css", con el que modificamos una propiedad CSS de los elementos obtenidos por el selector. En los dos siguientes, el método utilizado es el método "attr" que nos permitirá modificar un atributo de los elementos.

```
...
   $("p") .css ("background-color","red");
   jQuery("p") .css ("background-color","red");

   $("input") .attr ("value","25");
   jQuery("input") .attr ("value","25");
...
```

Parámetros del método

Por último, proporcionaremos a este método un conjunto de parámetros, que serán facilitados al método para configurar las acciones que el método aplicará sobre el conjunto de elementos que coincidan con el selector especificado.

En esta ocasión, vamos a ver los comandos por separado.

```
...
$("p") .css ("background-color","red");
jQuery("p") .css ("background-color","red");
...
```

En los primeros dos comandos, se aplica el método "css" sobre todos los elementos "p" del documento, pasándole a este método los parámetros "background-color" y "red".

```
 ...
$("input") .attr ("value","25");
jQuery("input") .attr ("value","25");
...
```

Este otro comando, aplicará el método "attr" sobre los elementos "input" del documento. En este caso, los dos parámetros asignados a este método son "value" y "25".

Algunos de los métodos de la librería no requerirán de ningún parámetro. Otros, podrán recibir un conjunto muy amplio de parámetros, y modificar una gran cantidad de propiedades sobre los objetos seleccionados y otros, incluso podrán recibir como parámetro una función JavaScript. El formato y número de parámetros puede variar de un método a otro e incluso, podremos hacer llamadas a un mismo método con diferente número de parámetros y orden de los mismos. Independientemente de esta característica, todos ellos cumplirán con la estructura básica de los comandos jQuery.

Estructura Básica de un comando

Encadenar varios métodos

Los métodos de la librería jQuery pueden ser de dos diferentes tipos según su naturaleza:

- Setters o establecedores, con los que se podrá establecer una propiedad a todos los elementos incluidos en la selección.
- Getters u obtenedores, que podrán devolver un valor concreto o un objeto jQuery referenciando una selección.

Cuando, como resultado de la llamada a un método obtenemos un objeto jQuery tendremos la posibilidad de encadenar varias llamadas a métodos de la librería, aplicándose éstas sobre la selección devuelta por el método anterior.

```
 $('input').hide().val('Indique un dato');
```

Analicemos este fragmento de código. El selector $('input') seleccionará los elementos de tipo input, se les aplicará el método hide() que los ocultará y finalmente se establecerá en el atributo value de todos ellos el valor 'Indique un dato'.

Para obtener un código más limpio y comprensible, podremos utilizar saltos de línea entre estos métodos, terminando siempre la secuencia de los mismos con el símbolo punto y coma.

```
 $('input')
   .hide()
   .val('Indique un dato');
```

Cuando uno de los métodos utilizados modifique una selección, los siguientes métodos utilizados en la llamada solamente actuarán sobre la selección resultado. En el siguiente fragmento de código, aplicaremos el método first() a la selección, consiguiendo así reducirla al primero de los elementos de la misma.

```
$('input')
   .first()
   .hide()
   .val('Indique un dato');
```

Una vez filtrado el primer elemento, los siguientes métodos únicamente actuarán sobre la nueva selección.

jQuery nos ofrece un método por el que podremos volver a la selección previa. Esta acción podrá realizarse por medio del método $.fn.end

```
$('input')
   .first()
      .hide()
   .end()
   .val('Indique un dato');
```

En el código anterior, el primer elemento de tipo input será ocultado, y elmétodo end() restablecerá la selección a todos los elementos de tipo input del documento para que les sea establecido el valor 'Indique un dato'.

Por medio del encadenamiento de métodos podremos realizar varias acciones en un mismo comando. Esta característica nos permitirá reducir el código, pero debe ser utilizada con cuidado. El abuso de encadenamiento de métodos pueden hacer un código difícil de comprender y modificar, y complicará la depuración de errores en el mismo.

No existe una regla estricta que defina el número máximo de comandos a encadenar, pero se recomienda que estas llamadas no sean excesivamente complejas ya que un código complicado será más complicado de entender.

Interoperabilidad con otras librerías

Dado que cada día las páginas web incluyen elementos más sofisticados, y las tecnologías utilizadas son más extensas, es posible que se nos de el caso de necesitar varios frameworks que puedan ser incompatibles con su configuración básica. Esta es una posibilidad latente que puede afectar al funcionamiento de nuestras páginas.

Este escenario ha sido estudiado por los desarrolladores de jQuery. Esta es la explicación de que el símbolo $ se corresponda con un alias de la función jQuery() . El símbolo $ nos permitirá reducir el código y hacer uso de la librería, pero en cualquier momento podremos modificar el funcionamiento de ésta para desvincular el alias.

Activación del modo no conflicto

Para demostrar la capacidad de jQuery de convivir con otras librerías, vamos a realizar un pequeño experimento.

Vamos a crear un documento HTML en el que incluir dos librerías. La primera de ellas, obviamente, va a ser jQuery. La segunda, otra librería llamada Prototype (versión 1.7). La librería Prototype también hace internamente uso del símbolo $, por lo que tendremos que intervenir para que jQuery libere el uso delsímbolo $ como alias. Esta acción se realiza mediante la llamada al método noConflict() de jQuery.

La activación del modo no conflicto puede realizarse de diversas formas.

Activación básica

Veamos el código de inclusión de ambas librerías y de activación básica del modo de no conflicto de jQuery:

```
 <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/prototype/1.7.1.0/prototype.js"></script>
<script src="jquery.js"></script>
<script>jQuery.noConflict();</script>
```

En la tercera línea de este fragmento, activamos el modo de no conflicto de jQuery. Una vez llamado a este método, las llamadas al símbolo $ harán referencia al uso del mismo en la librería Prototype.

Como podrás observar, la llamada al método noConflict() se ejecuta sin esperar a que cargue el DOM de la página. Este tipo de métodos puede ser ejecutado sin necesidad de esperar a que el DOM haya sido cargado, dado que al no realizar ninguna selección no consultará los elementos existentes en éste ni las propiedades de los mismos.

Un pequeño truco para poder seguir utilizando el símbolo $ como alias es pasar dicho símbolo como parámetro a la función ejecutada por el evento ready.

```
 <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/prototype/1.7.1.0/prot
otype.js"></script>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
  jQuery.noConflict();
  jQuery(document).ready(function($) {

    $('input').hide().val('Indique un dato');

  });

  // fuera del envento ready, el símbolo $ hará referencia a su uso por la librería Prototype
</script>
```

Activación con reasignación del alias

Otra opción que se podría utilizar sería la reasignación del alias en el momento de la llamada al método noConflict() . La reasignación del alias permite definir otro elemento que hará las veces de alias de reemplazo que podrá ser utilizado para acceder a la librería del mismo modo que ocurría con el símbolo $.

```
 <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/prototype/1.7.1.0/prototype.js"></script>
<script src="jquery.js"></script>
<script>
  var j = jQuery.noConflict();
  j(document).ready(function(){
    //...
  });
</script>
```

jQuery Migrate

Paralelamente al uso de jQuery en modo no conflicto y para aquellas aplicaciones que necesitan actualizar la versión de jQuery, manteniendo la compatibilidad con versiones anteriores de la librería, disponemos de un plugin con el que poder incluir en nuestras páginas una versión actual de jQuery sin que nuestro código deje de funcionar.

El plugin jQuery Migrate aplicará modificaciones sobre la librería para que, aquellos métodos marcados como "depreacted" no provoquen errores de ejecución y sigan funcionando "como se supone".

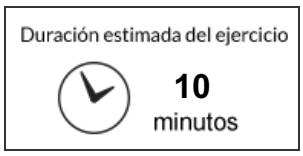
Para incluir jQuery Migrate en nuestro código, añadiremos el script posteriormente a la carga de jQuery

```
 <script src="http://code.jquery.com/jquery-1.9.0.js"></script>
<script src="http://code.jquery.com/jquery-migrate-1.3.0.js"></script>
```

Alternativamente, podremos descargar la librería desde el repositorio GitHub del proyecto en la dirección https://github.com/jquery/jquery-migrate

Ejercicios

Ejercicio 1: Captura del evento "ready" del documento



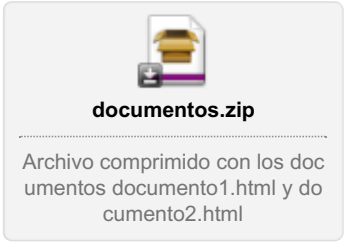
Como se ha explicado en la unidad, existen diferentes métodos para capturar el momento en el que el DOM ha sido completamente procesado.

Descarga el siguiente archivo comprimido y extrae los documentos en el directorio de trabajo. Una vez extraídos, captura el evento "ready" en ambos, ejecutando el siguiente codigo cuando esté completo:

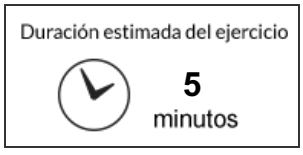
```
alert("El DOM se encuentra preparado");
```

Utiliza los siguientes modos de captura del evento:

- En el documento "documento1.html" por medio de la función .ready() de jQuery.
- En el documento "documento2.html" haciendo uso del "atajo" $(function())



Ejercicio 2: Comando básico



El siguiente comando ha sido desordenado, provocando errores en la interpretación del código. Reconstrúyelo para formar una llamada a la librería jQuery válida.

words Comando desordenado

Ejercicio 3: Interoperabilidad

Duración estimada del ejercicio
5
minutos

Descarga el documento base del ejercicio, en el que se incluye por medio del CDN de Google la librería Prototype y ubícalo en tu directorio de trabajo.

Modifica el documento para incluir jQuery e incluye el código necesario para que funcione en modo "no conflicto".

documento-ejercicio3.zip

Archivo comprimido con el documento documento3.html

Recursos

Enlaces de Interés

https://github.com/jquery/jquery-migrate

jQuery Migrate - librería que proporciona para nuevas versiones de jQuery compatibilidad con métodos marcados como deprecated.