

As bases de jQuery

Como base, jQuery enfócase á recuperación de elementos das páxinas HTML e realización de operacións sobre os mesmos. Se vostede está familiarizado con CSS, darase conta da potencia que teñen os **selectores**, os cales describen grupos de elementos polos seus atributos ou situación dentro do documento.

Sumario

- 1 A base de jQuery
- 2 **A clave de jQuery** A crave no aprendizaxe de jQuery está no uso da función **\$()** - alias de **jQuery()** -. Esta función poderíase comparar co clásico **document.getElementById()** pero coa diferenza moi importante de que **\$()** soporta selectores CSS e pode devolver arrays, polo tanto **\$()** é unha versión mellorada do **document.getElementById()**. Esta función **\$("selector")** acepta como parámetro unha cadea de texto que será un selector CSS, pero tamén pode aceptar un segundo parámetro **\$("selector", contexto)** que será o contexto no cal se vai a facer a búsqueda do selector citado. Ver un exemplo máis abaixo. Tamén poderemos ter outro uso da función con **\$(funcion)** equivalente a **\$(document).ready(función)**, que nos permitirá detectar cando o DOM está completamente cargado. Ver un exemplo máis abaixo
- 3 O encapsulamento en jQuery
- 4 **\$(selector, contexto)** A función **\$()** tamén pode ter un segundo parámetro que nos indicaría o contexto no cal se vai a facer a selección.
- 5 **\$(funcion) jQuery(funcion) \$(document).ready(funcion())** Permite detectar cando o DOM (Modelo de Obxectos do Documento) está completamente cargado, para evitar posibles erros de acceso ós obxectos.
- 6 Funcións de utilidade en jQuery
- 7 Explicación en detalle do selector document ready
- 8 Creando elementos DOM
- 9 Emprego de jQuery con outras librerías

A base de jQuery



A clave de jQuery

A crave no aprendizaxe de jQuery está no uso da función **\$()** - alias de **jQuery()** -. Esta función poderíase comparar co clásico **document.getElementById()** pero coa diferenza moi importante de que **\$()** soporta selectores CSS e pode devolver arrays, polo tanto **\$()** é unha versión mellorada do **document.getElementById()**.

Esta función **\$("selector")** acepta como parámetro unha cadea de texto que será un selector CSS, pero tamén pode aceptar un segundo parámetro **\$("selector", contexto)** que será o contexto no cal se vai a facer a búsqueda do selector citado. Ver un exemplo máis abaixo.

Tamén poderemos ter outro uso da función con **\$(funcion)** equivalente a **\$(document).ready(función)**, que nos permitirá detectar cando o DOM está completamente cargado. Ver un exemplo máis abaixo

[Consultar máis información sobre a función \\$\(\)](#).

O encapsulamento en jQuery

Cando se introduciu CSS nas tecnoloxías web fíxose co fin de separar o deseño do contido, de forma que puideramos facer referencia ós elementos da páxina dende as follas de estilo.

O método foi desenvolvido facendo uso de **selectores**, que **referencian os elementos en base os seus atributos ou a súa posición** no documento HTML.

Por exemplo o selector

```
p a
```

```
fai referencia ó grupo dos enlaces (elementos <a>) que están contidos dentro dun elemento p (párrafo).
```

jQuery fai uso dos mesmos selectores, dando soporte ós selectores máis comúns empregados en CSS, e tamén a moitos selectores que aínda non están soportados por moitos navegadores.

Por exemplo o selector **nth-child** que vimos nun exemplo anterior é un bo exemplo da potencia de selección en jQuery.

Para facer referencia a un grupo de elementos, usaremos unha sintaxis moi simple:

```
$( "selector" )  
  
ou  
  
jQuery( "selector" )
```

O máis cómodo é empregar a notación **\$()**, aínda que resulte un pouco estraña ó principio.

Por exemplo, para recuperar o grupo de enlaces que está dentro dun elemento p, usaríamos o seguinte:

```
$( "p a" )
```

A función **\$()** (alias de **jQuery()**) devolve un obxecto especial JavaScript que contén un array de todos os elementos DOM que cumpren o selector.



\$(selector,contexto)

A función **\$()** tamén pode ter un segundo parámetro que nos indicaría o contexto no cal se vai a facer a selección.

Por exemplo:

```
$( "input", document.forms[0] );  
  
// Seleccionaría todos os obxectos input no primeiro formulario  
// do documento.
```

En termos de programación, estase a producir un encapsulamento xa que estamos accedendo a todos os elementos seleccionados e estamos aplicándolle unha funcionalidade extendida.

Por exemplo se queremos ocultar todos os elementos div dun documento que empregan a clase "colorido", en jQuery faríamos o seguinte:

```
$( "div.colorido" ).fadeOut();
```

Unha característica especial en moitos comandos de jQuery é que unha vez que ese comando terminou a súa acción (como por exemplo a operación anterior de ocultar fadeout()), ese mesmo comando devolve o mesmo grupo de elementos listos para poder ser usados nunha nova acción.

Por exemplo, se queremos engadir unha nova clase os elementos que ocultamos recentemente poderíamos facer:

```
$( "div.colorido" ).fadeOut().addClass( "eliminados" );
```

Este encadeamento de accións podería continuar indefinidamente.

Outro exemplo:

```
$( "#unelemento" ).html( "Engadimos texto ó elemento" );  
  
ou  
  
$( "#unelemento" )[0].innerHTML = "Engadimos texto ó elemento";
```

No exemplo anterior empregamos un selector de ID ("**#id**"), polo que soamente un obxecto do documento cumprirá ese selector. No primeiro caso empregamos o método **html()**, o cal reemplaza o contido do elemento co novo contido HTML.

Se queremos face-lo mesmo pero con varios elementos poríamos:

```

$("div.novotexto").html("Engadimos texto a un grupo de div que están usando a clase novotexto");

ou

var elements = $("div.novotexto");
for(i=0;i<elements.length;i++)
elements[i].innerHTML="Engadimos texto a un grupo de div que están usando a clase novotexto";

```

Outros exemplos:

```

$("p:even"); //Este selector selecciona todos os párrafos que sexan pares.
$("tr:nth-child(1)"); /*Este selector selecciona a primeira fila de tódalas táboas do documento. */
$("body > div"); //Este selector selecciona os <div> fillos que colgan directamente de <body>.
$("a[href$=pdf]"); //Este selector selecciona enlaces a ficheiros PDF.
$("body > div:has(a)"); //Este selector selecciona <div> que son fillos
// directos de <body> e que ademáis conteñan enlaces.

```



\$(function) jQuery(function) \$(document).ready(function())

Permítenos detectar cando o DOM (Modelo de Obxectos do Documento) está completamente cargado, para evitar posibles erros de acceso ós obxectos.

Exemplos de uso dos tres métodos anteriores:

```

$(function(){
/*
Cando o documento está preparado.
Poderemos acceder de forma exitosa a tódolos
obxectos do documento.
A partir de aquí teclearíamos o código de jquery....
*/

```

```

});

```

```

jQuery(function(){
/*
Cando o documento está preparado.
Poderemos acceder de forma exitosa a tódolos
obxectos do documento.
A partir de aquí teclearíamos o código de jquery....
*/

```

```

});

```

```

$(document).ready(function(){
/*
Cando o documento está preparado.
Poderemos acceder de forma exitosa a tódolos
obxectos do documento.
A partir de aquí teclearíamos o código de jquery....
*/

```

```

});

```

Poderemos empregar calqueira dos tres métodos anteriores nos nosos exemplos, xa que **os tres métodos son equivalentes**.

Funcións de utilidade en jQuery

Aínda que unha das tarefas máis frecuente da función `jQuery()` é a de seleccionar un grupo de elementos, ésta non é a única tarefa que ten asignada.

Unha desas tarefas adicionais que tamén ten asignada é a de servir de prefixo para a chamada de funcións de certa utilidade.

Por exemplo:

```
$.trim(algo);
```

Se che parece raro o uso do prefixo `$`, recorda que `$` é un identificador como calquera outro en JavaScript.

Poderíamos facer a mesma chamada empregando a función `jQuery`:

```
jQuery.trim(algo);
```

Aquí queda claro que a función `trim()` é chamada no espazo de `jQuery` ou polo seu alias `$`

NOTA: Aínda que estes elementos son chamados funcións de utilidade na documentación de `jQuery`, queda claro que son métodos da función `$()`

Explicación en detalle do selector document ready

Cando falamos do JavaScript non intrusivo, estamos dicindo que o comportamento está separado da estrutura do documento. Polo tanto realizaremos operacións sobre os elementos da páxina dende fora da estrutura da mesma.

Para poder facer isto, necesitamos esperar a que todos os elementos DOM (Document Object Model - Modelo de Obxectos do Documento) na páxina estén completamente cargados, antes de comenzo a operar con eles.

Tradicionalmente facíase cun evento `onload` do obxecto `window` de JavaScript. A sintaxis típica era:

```
window.onload = function() {  
  $("table tr:nth-child(even)").addClass("colorido");  
};
```

Este código facía que o JavaScript non se executara ata que a táboa esté completamente cargada.

Desafortunadamente co evento `onload` o navegador ralentiza a execución deste código JavaScript ata que o DOM esté completamente cargado, e ademais ata que tódalas imaxes e recursos externos tamén estén cargados.

Como resultado os visitantes notarán un certo retraso dende que ven a páxina ata que se executa o script. Incluso pode ser peor, se unha imaxe necesita moito tempo para cargar, teríamos que esperar ata que estivera completamente cargada para que o Javascript se execute, e isto faría que non fora moi enriquecedor o emprego de certas funcións visuais en Javascript.

Unha mellora desta técnica sería o ter que esperar simplemente a que o arbol DOM esté construído. Programar ésto para que funcione en varios navegadores é unha tarefa complexa que con `jQuery` pódese facer relativamente sinxelo, da seguinte forma:

```
$(document).ready(function()  
{  
  $("table tr:nth-child(even)").addClass("colorido");  
}  
);
```

O primeiro que facemos é encapsular o documento coa función `jQuery()`, e a continuación aplicámoslle o método `ready()` ó cal se lle pasa unha función que se executará cando o documento xa esté preparado para a súa manipulación.

A forma reducida do exemplo anterior sería:

```
$(function() {  
    $("table tr:nth-child(even)").addClass("colorido");  
});
```

Pasando unha función a `$()`, estamos indicándolle ó navegador que espere ata que o DOM esté completamente cargado (pero soamente o DOM) antes de comezar a executa-lo código. Incluso mellor, podemos usar esta técnica moitas veces dentro do documento HTML, e o navegador executará tódalas funcións indicadas no orden no que foron declaradas na páxina.

Polo contrario a función `onLoad` do Javascript soamente permite o uso dunha soa función.

Creando elementos DOM

Os autores de jQuery co emprego de `$()` (alias de `jQuery()`) conseguiron facer unha función o suficientemente flexible para realizar moitas das tarefas que vimos ata agora.

Pero imos a explicar unha nova función de `$()`, que é a creación de novos obxectos DOM en execución. Para elo simplemente teremos que pasarlle á función o código html para os novos elementos.

Exemplo:

```
$("#<p>Hola a todos!</p>")
```

Vexamos un exemplo máis completo:

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
    <meta charset="UTF-8">  
    <title>Saudos a todos!</title>  
</head>  
<body>  
<p id="parrafo">E Benvidos ó curso!</p>  
    <script src="http://code.jquery.com/jquery-2.1.3.min.js"></script>  
    <script>  
        $(document).ready(function() {  
            $("#<p>Hola a todos!</p>").insertBefore("#parrafo");  
        });  
    </script>  
</body>  
</html>
```

No exemplo anterior vemos que a función creará un novo párrafo xusto antes do obxecto con ID parrafo.

Emprego de jQuery con outras librerías

Ainda que jQuery proporciona un potente conxunto de ferramentas que satisfacen á maioría dos programadores, pode haber momentos en que unha páxina requira de múltiples bibliotecas JavaScript. Esta situación podería producirse porque estamos no proceso de transición dunha páxina para o seu uso coa biblioteca jQuery, ou talvez por que queiramos empregar ambas jQuery e outra biblioteca nas nosas páxinas.

Por exemplo si queremos empregar a librería Prototype xunto con jQuery encontraremosnos co conflito de que Prototype tamén usa o símbolo `$` como identificador, polo que chocarían no espacio de nomes con jQuery.

Os autores de jQuery proporcionan unha forma de eliminar este conflito cunha función denominada **`noConflict()`**. En calquera momento cando a librería conflictiva sexa cargada, unha chamada a **`jQuery.noConflict()`**; cederá ó símbolo `$` o significado que ten na outra librería.

--Veiga ([discusión](#)) 13:50 26 ene 2015 (CET)