Control absoluto dunha petición Ajax

Sumario

O método ajax()

O método principal para realizar peticións AJAX é **$.ajax()** (importante non olvidar o punto entre $ e ajax(). A partir desta función básica, definíronse outras funcións Ajax relacionadas, de máis alto nivel e especializadas en tarefas concretas: **$.get(), $.post(), $.load(),** etc.

Polo tanto se queremos levar un control pormenorizado da petición ajax dispomos desta función **$.ajax()**.

A sintaxe é a seguinte: **$.ajax(opcions)**.

En principio parece moi simple, pero o número de opcións dispoñible é relativamente extenso.

Ésta é a estructura básica

```
$.ajax({
   url: [URL],
   type: [GET/POST],
   success: [function callback exito(data)],
   error: [function callback error],
   complete: [function callback error],
   ifModified: [bool comprobar E-Tag],
   data: [mapa datos GET/POST],
   async: [bool que indica sincronía/asincronia]
});

// Por exemplo:

$.ajax({
  url: '/ruta/paxina.php',
  type: 'POST',
  async: true,
  data: 'parametro1=valor1&parametro2=valor2',
  success: function (resposta)
                   {
                         alert(resposta);
                   },
  error: amosarErro
});
```

Listado de opcións dispoñibles no método $.ajax()

NOME	TIPO	DESCRIPCION
url	**String**	A url para a solicitude
type	**String**	O método HTTP a empregar. Pode ser POST ou GET. Si se omite empregarase GET por defecto.
data	**Object**	Un obxeto no que se especifican parámetros que serán enviados á solicitude. Si a solicitude é de tipo GET, estos datos son pasados como cadea na URL. Si se emprega o método POST, os datos son enviados como parte do Body. En calqueira caso a codificación dos valores é xestionada pola función $.ajax(). Aconséllovos empregar a función **serialize()** que é moi útil nestes casos (ver información máis abaixo).
dataType	**String**	É unha palabra clave que identifica o tipo de datos que se espera que se devolvan na resposta. Este valor determina qué tipo de procesamento faremos cos datos recibidos na resposta, si é que temos que procesalos:

		• xml - O texto de resposta é analizado como un documento XML e o DOM XML resultante será enviado á función de retorno. • html - O texto de resposta é enviado sen procesamiento previo á función de retorno. Calqueira bloque <script> contido no texto HTML será evaluado. • json - O texto de resposta é analizado como unha cadena JSON e o obxecto resultante enviaráse á función de retorno. • jsonp - Similar a json, salvo que se permite scripting remoto, sempre e cando o servidor remoto o soporte. • script - O texto de resposta será enviadoá función de retorno. Previamente a que dita función se execute a resposta de tipo script será evaluada como código de Javascript. • text - O texto de resposta será do tipo texto plano. O servidor será o responsable de axustar as cabeceiras correspondentes ó tipo de contido. Si se omite esta propiedade, o texto de resposta é enviado á función de retorno sin ningún tipo de procesamento.
timeout	**Number**	Establece un tempo máximo de execución en milisegundos. Si a solicitude non se executa no tempo estimado, abortaráse a solicitude e a función de error será chamada (sempre e cando ésta fose definida na sección correspondente).
global	**Boolean**	Habilita (true) o deshabilita (false) as funcións globais de Ajax. Estas funcións poden ser asignadas a diferentes obxectos no documento e serán chamadas nos diferentes estados da execución da petición Ajax. Por defecto están habilitadas.
contentType	**String**	O tipo de contido que será especificado na respuesta.Si se omite, o contenido por defecto é *application/x-www-form-urlencoded,* o mismo tipo utilizado cando se envía un formulario.
success	**Function**	Unha función que será chamada si a respuesta á solicitude rematou con éxito. O corpo da resposta será enviado como primeiro parámetro da función e formateado según o formato especificado na propiedade dataType. O segundo parámetro é un string que contén o valor de status - neste caso, a cadea success.
error	**Function**	Está función será chamada sempre e cando a solicitude devolva un código de erro. A esta función se lle pasarán 3 argumentos: a instancia XHR, unha cadea de texto que contén o texto da mensaxe de error (neste caso, error), e opcionalmente un obxecto excepción devolto pola instancia XHR.
complete	**Function**	Esta función será chamada cando a solicitude foi completada. Recibe dous parámetros: a instancia XHR e unha mensaxe de estado co texto success ou error. Tanto si rematou con éxito ou error ésta función será chamada cando a solicitude remata.
beforeSend	**Function**	Unha función chamada previamente ó envio da solicitude.Se lle pasa a instancia XHR e pode ser empregada para definir cabeceiras específicas ou para facer tarefas previas ó envío da solicitude.
async	**Boolean**	Si se especifica como false, a solicitude será enviada como unha petición *síncrona*. Por defecto a petición será *asíncrona*.
processData	**Boolean**	Si se pon a false, evita que os datos enviados sexan procesados e codificados en formato URL. Por defecto os datos serán codificados en formato de URL para o seu uso con solicitudes de tipo *application/x-www-form-urlencoded*.
ifModified	**Boolean**	Si é true, permite que a solicitude termine con éxito sempre e cando o texto de resposta non fora modificado dende a última solicitude feita ó servidor, de acordo coa cabeceira Last-Modified. Si se omite non se chequeará a cabeceira de resposta.

Máis información de referencia sobre as opcións do método ajax()

A función serialize()

Esta función xenera unha cadea de datos que estará composta polos campos dun formulario e os seus respectivos valores. Serialize emprégase para preparar o conxunto de datos que se van a enviar ó servidor. Os datos serializados irán nun formato standard que é compatible coa maioría de linguaxes de programación e frameworks.

É moi útil para formatear os datos que se enviarán na opción **data** do método $.ajax() xa que o serialize devolverá os datos no seguinte formato: campo1=valor1&campo2=valor2&campo3=valor3&.....

Para que serialize funcione correctamente é necesario que os campos do formulario teñan atributo name. Se soamente teñen atributo id non funcionará.

Exemplo:

```
<input id="email" name="email" type="text" />
```

Máis información sobre serialize().

A función de retorno (callback) nos métodos ajax de jQuery

Cando se fai unha petición Ajax a un servidor esta petición pode demorar máis ou menos tempo, non é algo que dependa de nós, dependerá de si o servidor está saturado ou non, do tipo de consulta que se lle faga, etc..etc..

¿Qué queremos dicir con isto?

Pois que cando facemos a solicitude ajax, o navegador vai a continuar executando as seguintes instruccións, non vai a esperar a que remate o **.load(), .post(), .get(), ou .ajax().**

Exemplo 1:

```
$("#novacapa").load("formupdates.html");
alert($("#codigo").val());
alert($("#data").val());
```

Polo tanto no exemplo superior, vai a executar a segunda instrucción inmediatamente despois de executar a primeira, co que teríamos que ter moita sorte para que estivese cargado o formulario antes de que accedamos ó seu contido con $("#codigo"). E como iso non vai a ocurrir por que sempre vai a ser máis lenta a instrucción ajax .load(), van a fallar as instruccións alert($("#codigo").val()); e alert($("#data").val()); por que simplemente eses obxectos ós que facemos referencia todavía non están dispoñibles no documento (son obxectos do formulario formupdates.html que estamos cargando con .load()).

A solución polo tanto radica en programar un parámetro máis que será unha función de retorno ou callback (Exemplo 2), que se executará cando teñamos dispoñible a resposta do servidor, e dentro desa función xa poderemos acceder ós datos de resposta do servidor (o formulario recén cargado, neste caso) ou ós datos da consulta, etc..

Si na función de callback introducimos un parámetro, nese parámetro teremos a resposta que deu o servidor.

A programación deste tipo de función de retorno está dispoñible nos **4 métodos ajax de jQuery**: **.load() .ajax() .post() e .get()**

Exemplo 2:

```
$("#novacapa").load("formupdates.html", function(){
    alert($("#codigo").val());
    alert($("#data").val());
.............
});
```

Parámetros por defecto na función $.ajax()

Pode darse o caso de que empreguemos a función **$.ajax()** moitas veces no documento e que na maior parte dos casos teñamos moitas opcións comúns. Para eses casos dispomos do método **$.ajaxSetup()** que nos permitirá axustar as opcións por defecto para as peticións ajax() posteriores que fagamos despois do axuste.

Por exemplo:

```
$.ajaxSetup({
  type: 'POST',
  timeout: 5000,
  dataType: 'html',
  error: function(xhr) {
          $('#errorDisplay').html('Error: ' + xhr.status + ' ' + xhr.statusText);
        }
})
```

NOTA: Os parámetros por defecto non serán aplicados á función load(). En funcións de utilidade como $.get() ou $.post(), o método HTTP non se poderá sobreescribir co uso destes parámetros por defecto. Por exemplo si axustamos como tipo por defecto GET, isto non provocará que $.post() use o método GET.

Máis información sobre os parámetros e funcións Ajax en jQuery.

Funcións globais

Además da posibilidade de especificar funcións que se executarán por defecto (mediante as opcións de ajaxSetup()), jQuery tamén nos permitirá enlazar funcións a elementos do DOM. ***Estas funcións serán chamadas de forma automática durante as diferentes fases de procesamento da solicitude Ajax ou cando dita solicitude remate de forma exitosa ou con erros***.

Por exemplo, para enlazar unha función a un elemento cun id errorConsola co único propósito de que amose mensaxes de error, escribiremos o seguinte:

```
$('#errorConsola').ajaxError(function(request, settings));
```

A función reportError será chamada no caso de que ocurra algún erro na petición Ajax. Cando esta ou calquera das funcións globais é invocada, o primeiro parámetro que se pasa á función consistirá nun obxecto JavaScript coas seguintes propiedades:

? **type**? Unha cadea que contén o tipo de función global invocada - ajax - Error, por exemplo.

? **target**? A referencia ó elemento do DOM ó cal se enlazou a función global. No caso do exemplo anterior, será o elemento co id errorConsola.

Os comandos que nos permitirán enlazar estas funcións globais son: **ajaxStart(), ajaxSend(), ajaxSuccess(), ajaxError(), ajaxComplete(), and ajaxStop()**.

Funcións Globais Ajax, listadas según orden de disparo:

Tipo retorno Global	Cando é chamada	Parámetros
ajaxStart	Cando se comeza unha función Ajax ou comando, pero antes de que a instancia XHR sexa creada.	• Un obxecto global de retorno co tipo ajaxStart.
ajaxSend	Despois de que a instancia XHR foi creada, pero antes de ser enviada ó servidor.	• Un obxecto global de retorno co tipo ajaxSend. • A instancia XHR. • As propiedades empregadas polo método $.ajax()
ajaxSuccess	Despois de que a solicitude fora devolta polo servidor e a resposta contén o código de status success.	• Un obxecto global de retorno co tipo ajaxSuccess. • A instancia XHR. • As propiedades empregadas polo método $.ajax()
ajaxError	Despois de que a solicitude fora devolta polo servidor e a resposta contén o código de fallo.	• Un obxecto global de retorno co tipo ajaxError. • A instancia XHR. • As propiedades empregadas polo método $.ajax() • Un obxecto excepción devolto pola instancia XHR, si fora o caso.
ajaxComplete	Despois de que a solicitude fora devolta polo servidor e despois de que calqueira chamada ajaxSuccess ou ajaxError fora realizada.	• Un obxecto global de retorno co tipo ajaxComplete. • A instancia XHR. • As propiedades empregadas polo método $.ajax()
ajaxStop	Despois de que todos os pasos anteriores foron realizados éste sería o derradeiro paso.	• Un obxecto global de retorno co tipo ajaxStop.

Exemplo:

Temos un div chamado errores no que amosaremos os erros que se produzan nas chamadas ajax.

```
<fieldset>
<legend>Amosamos Erros</legend>
<div id="erros"></div>
</fieldset>
```

```
$("#erros").ajaxError(function(event, request, settings){
       $(this).append("<li>Error solicitando a paxina " + settings.url + "</li>");
 });
```

Máis información sobre estas funcións globais.

--Veiga (discusión) 14:17 26 ene 2015 (CET)