jQuery API

- New or Changed in 1.6
- Raw XML API Dump
- Dynamic API Browser
- jQuery API Book

Browse the jQuery API

- All
- + Ajax
- Attributes
- Core
- CSS
- Data
- Deferred Object
- Dimensions
- + Effects
- + Events
- Forms
- Internals
- + Manipulation
- + Miscellaneous
- Offset
- + Plugins
- + Properties
- + Selectors
- + Traversing
- Utilities
- + Version

jQuery.ajax()

Categories: Ajax > Low-Level Interface

jQuery.ajax(url, [settings]) *Returns: jqXHR*

Description: *Perform an asynchronous HTTP (Ajax) request.*

jQuery.ajax(url, [settings])

version added: 1.5

url A string containing the URL to which the request is sent.

settings A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) below for a complete list of all settings.

jQuery.ajax(settings)

version added: 1.0

settings A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup().

accepts — Map
Default: depends on DataType

The content type sent in the request header that tells the server what kind of response it will accept in return. If the `accepts` setting needs modification, it is recommended to do so once in the `$.ajaxSetup()` method

async — Boolean
Default: true

By default, all requests are sent asynchronously (i.e. this is set to `true` by default). If you need synchronous requests, set this option to `false`. Cross-domain requests and `dataType: "jsonp"` requests do not support synchronous operation. Note that synchronous requests may temporarily lock the browser, disabling any actions while the request is active.

beforeSend(jqXHR, settings) — Function

A pre-request callback function that can be used to modify the jqXHR (in jQuery 1.4.x, XMLHTTPRequest) object before it is sent. Use this to set custom headers, etc. The jqXHR and settings maps are passed as arguments. This is an Ajax Event. Returning `false` in the `beforeSend` function will cancel the request. **As of jQuery 1.5**, the `beforeSend` option will be called regardless of the type of request.

cache — Boolean
Default: true, false for dataType 'script' and 'jsonp'

If set to `false`, it will force requested pages not to be cached by the browser. Setting cache to `false` also appends a query string parameter, "_=[TIMESTAMP]", to the URL.

complete(jqXHR, textStatus) — Function, Array

A function to be called when the request finishes (after `success` and `error` callbacks are executed). The function gets passed two arguments: The jqXHR (in jQuery 1.4.x, XMLHTTPRequest) object and a string categorizing the status of the request (`"success"`, `"notmodified"`, `"error"`, `"timeout"`, `"abort"`, or `"parsererror"`). **As of jQuery 1.5**, the `complete` setting can accept an array of functions. Each function will be called in turn. This is an Ajax Event.

contents (added 1.5) — Map

A map of string/regular-expression pairs that determine how jQuery will parse the response, given its content type.

contentType — String
Default: 'application/x-www-form-urlencoded'

When sending data to the server, use this content-type. Default is "application/x-www-form-urlencoded", which is fine for most cases. If you explicitly pass in a content-type to `$.ajax()` then it'll always be sent to the server (even if no data is sent). Data will always be transmitted to the server using UTF-8 charset; you must decode this appropriately on the server side.

context Object

This object will be made the context of all Ajax-related callbacks. By default, the context is an object that represents the ajax settings used in the call (`$.ajaxSettings` merged with the settings passed to `$.ajax`). For example specifying a DOM element as the context will make that the context for the `complete` callback of a request, like so:

```
$.ajax({
  url: "test.html",
  context: document.body,
  success: function(){
    $(this).addClass("done");
  }
});
```

converters (added 1.5) Map
Default: {"* text": window.String, "text html": true, "text json": jQuery.parseJSON, "text xml": jQuery.parseXML}

A map of dataType-to-dataType converters. Each converter's value is a function that returns the transformed value of the response

crossDomain (added 1.5)
Default: false for same-domain requests, true for cross-domain requests

If you wish to force a crossDomain request (such as JSONP) on the same domain, set the value of crossDomain to `true`. This allows, for example, server-side redirection to another domain

data Object, String

Data to be sent to the server. It is converted to a query string, if not already a string. It's appended to the url for GET-requests. See processData option to prevent this automatic processing. Object must be Key/Value pairs. If value is an Array, jQuery serializes multiple values with same key based on the value of the `traditional` setting (described below).

dataFilter(data, type) Function

A function to be used to handle the raw response data of XMLHttpRequest. This is a pre-filtering function to sanitize the response. You should return the sanitized data. The function accepts two arguments: The raw data returned from the server and the 'dataType' parameter.

dataType String
Default: Intelligent Guess (xml, json, script, or html)

The type of data that you're expecting back from the server. If none is specified, jQuery will try to infer it based on the MIME type of the response (an XML MIME type will yield XML, in 1.4 JSON will yield a JavaScript object, in 1.4 script will execute the script, and anything else will be returned as a string). The available types (and the result passed as the first argument to your success callback) are:

"xml": Returns a XML document that can be processed via jQuery.

"html": Returns HTML as plain text; included script tags are evaluated when inserted in the DOM.

"script": Evaluates the response as JavaScript and returns it as plain text. Disables caching by appending a query string parameter, "_=[TIMESTAMP]", to the URL unless the `cache` option is set to `true`. **Note:** This will turn POSTs into GETs for remote-domain requests.

"json": Evaluates the response as JSON and returns a JavaScript object. In jQuery 1.4 the JSON data is parsed in a strict manner; any malformed JSON is rejected and a parse error is thrown. (See json.org for more information on proper JSON formatting.)

"jsonp": Loads in a JSON block using JSONP. Adds an extra "?callback=?" to the end of your URL to specify the callback. Disables caching by appending a query string parameter, "_=[TIMESTAMP]", to the URL unless the `cache` option is set to `true`.

"text": A plain text string.

multiple, space-separated values: **As of jQuery 1.5**, jQuery can convert a dataType from what it received in the Content-Type header to what you require. For example, if you want a text response to be treated as XML, use "text xml" for the dataType. You can also make a JSONP request, have it received as text, and interpreted by jQuery as XML: "jsonp text xml." Similarly, a shorthand string such as "jsonp xml" will first attempt to convert from jsonp to xml, and, failing that, convert from jsonp to text, and then from text to xml.

error(jqXHR, textStatus, errorThrown) Function

A function to be called if the request fails. The function receives three arguments: The jqXHR (in jQuery 1.4.x, XMLHttpRequest) object, a string describing the type of error that occurred and an optional exception object, if one occurred. Possible values for the second argument (besides `null`) are `"timeout"`, `"error"`, `"abort"`, and

"parsererror". When an HTTP error occurs, errorThrown receives the textual portion of the HTTP status, such as "Not Found" or "Internal Server Error." **As of jQuery 1.5**, the error setting can accept an array of functions. Each function will be called in turn. **Note:** *This handler is not called for cross-domain script and JSONP requests.* This is an Ajax Event.

global Boolean
Default: true

Whether to trigger global Ajax event handlers for this request. The default is true. Set to false to prevent the global handlers like ajaxStart or ajaxStop from being triggered. This can be used to control various Ajax Events.

headers (added 1.5) Map
Default: {}

A map of additional header key/value pairs to send along with the request. This setting is set before the beforeSend function is called; therefore, any values in the headers setting can be overwritten from within the beforeSend function.

ifModified Boolean
Default: false

Allow the request to be successful only if the response has changed since the last request. This is done by checking the Last-Modified header. Default value is false, ignoring the header. In jQuery 1.4 this technique also checks the 'etag' specified by the server to catch unmodified data.

isLocal (added 1.5.1) Boolean
Default: depends on current location protocol

Allow the current environment to be recognized as "local," (e.g. the filesystem), even if jQuery does not recognize it as such by default. The following protocols are currently recognized as local: file, *-extension, and widget. If the isLocal setting needs modification, it is recommended to do so once in the $.ajaxSetup() method.

jsonp String

Override the callback function name in a jsonp request. This value will be used instead of 'callback' in the 'callback=?' part of the query string in the url. So {jsonp:'onJSONPLoad'} would result in 'onJSONPLoad=?' passed to the server. **As of jQuery 1.5**, setting the jsonp option to false prevents jQuery from adding the "?callback" string to the URL or attempting to use "=?" for transformation. In this case, you should also explicitly set the jsonpCallback setting. For example, { jsonp: false, jsonpCallback: "callbackName" }

jsonpCallback String, Function

Specify the callback function name for a JSONP request. This value will be used instead of the random name automatically generated by jQuery. It is preferable to let jQuery generate a unique name as it'll make it easier to manage the requests and provide callbacks and error handling. You may want to specify the callback when you want to enable better browser caching of GET requests. **As of jQuery 1.5**, you can also use a function for this setting, in which case the value of jsonpCallback is set to the return value of that function.

mimeType (added 1.5.1) String

A mime type to override the XHR mime type.

password String

A password to be used in response to an HTTP access authentication request.

processData Boolean
Default: true

By default, data passed in to the data option as an object (technically, anything other than a string) will be processed and transformed into a query string, fitting to the default content-type "application/x-www-form-urlencoded". If you want to send a DOMDocument, or other non-processed data, set this option to false.

scriptCharset String

Only for requests with "jsonp" or "script" dataType and "GET" type. Forces the request to be interpreted as a certain charset. Only needed for charset differences between the remote and local content.

statusCode (added 1.5) Map
Default: {}

A map of numeric HTTP codes and functions to be called when the response has the corresponding code. For example, the following will alert when the response status is a 404:

```
$.ajax({
  statusCode: {
    404: function() {
```

```
    alert('page not found');
  }
 }
});
```

If the request is successful, the status code functions take the same parameters as the success callback; if it results in an error, they take the same parameters as the `error` callback.

success(data, textStatus, jqXHR) Function, Array

A function to be called if the request succeeds. The function gets passed three arguments: The data returned from the server, formatted according to the `dataType` parameter; a string describing the status; and the `jqXHR` (in jQuery 1.4.x, XMLHttpRequest) object. **As of jQuery 1.5**, *the success setting can accept an array of functions. Each function will be called in turn.* This is an Ajax Event.

timeout Number

Set a local timeout (in milliseconds) for the request. This will override the global timeout, if one is set with $.ajaxSetup(). For example, you could use this property to give a single request a longer timeout than all other requests that you've set to time out in one second. See `$.ajaxSetup()` for global timeouts. **In jQuery 1.4.x and below,** please note that the XMLHttpRequest object will be in an invalid state should the request time out. Where this is the case, accessing any object members may result in an exception being thrown. **In jQuery 1.5.2 and above,** `$.ajax()` does not handle JSONP requests as expected should the request fail due to a timeout in Firefox 3.0+. This is a browser-based issue due to FF currently not providing a way to abort cross-domain requests once the script tag has been appended. This issue does not currently affect other browsers.

traditional Boolean

Set this to `true` if you wish to use the traditional style of param serialization.

type String
Default: 'GET'

The type of request to make ("POST" or "GET"), default is "GET". **Note:** Other HTTP request methods, such as PUT and DELETE, can also be used here, but they are not supported by all browsers.

url String
Default: The current page

A string containing the URL to which the request is sent.

username String

A username to be used in response to an HTTP access authentication request.

xhr Function
Default: ActiveXObject when available (IE), the XMLHttpRequest otherwise

Callback for creating the XMLHttpRequest object. Defaults to the ActiveXObject when available (IE), the XMLHttpRequest otherwise. Override to provide your own implementation for XMLHttpRequest or enhancements to the factory.

xhrFields (added 1.5.1) Map

A map of fieldName-fieldValue pairs to set on the native `XHR` object. For example, you can use it to set `withCredentials` to `true` for cross-domain requests if needed.

```
$.ajax({
  url: a_cross_domain_url,
  xhrFields: {
    withCredentials: true
  }
});
```

In jQuery 1.5, the `withCredentials` property was not propagated to the native `XHR` and thus CORS requests requiring it would ignore this flag. For this reason, we recommend using jQuery 1.5.1+ should you require the use of it.

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get()` and `.load()` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
$.ajax();
```

Note: Default settings can be set globally by using the `$.ajaxSetup()` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, we can implement one of the callback functions.

The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()` **as of jQuery 1.5** is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
$.ajax({
  url: 'http://fiddle.jshell.net/favicon.png',
  beforeSend: function( xhr ) {
    xhr.overrideMimeType( 'text/plain; charset=x-user-defined' );
  },
  success: function( data ) {
    if (console && console.log){
      console.log( 'Sample of data:', data.slice(0,100) );
    }
  }
});
```

The jqXHR objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, jqXHR also provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the `$.ajax()` request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.)

> **Deprecation Notice:** The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks will be deprecated in jQuery 1.8. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.ajax( "example.php" )
   .success(function() { alert("success"); })
   .error(function() { alert("error"); })
   .complete(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.complete(function(){ alert("second complete"); });
```

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

- `readyState`
- `status`
- `statusText`
- `responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively
- `setRequestHeader(name, value)` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one
- `getAllResponseHeaders()`
- `getResponseHeader()`
- `abort()`

No `onreadystatechange` mechanism is provided, however, since `success`, `error`, `complete` and `statusCode` cover all conceivable requirements.

Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `error` (`fail`), `success` (`done`), and `complete` (`always`, as of jQuery 1.6) callback hooks are first-in, first-out managed queues. This means you can assign more than one callback for each hook. See Deferred object methods, which are implemented internally for these `$.ajax()` callback hooks.

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

Some types of Ajax requests, such as JSONP and cross-domain GET requests, do not use XHR; in those cases the `XMLHttpRequest` and `textStatus` parameters passed to the callback are `undefined`.

Here are the callback hooks provided by `$.ajax()`:

1. `beforeSend` callback is invoked; it receives the `jqXHR` object and the `settings` map as parameters.
2. `error` callbacks are invoked, in the order they are registered, if the request fails. They receive the `jqXHR`, a string indicating the error type, and an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".
3. `dataFilter` callback is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.
4. `success` callbacks are then invoked, in the order they are registered, if the request succeeds. They receive the returned data, a string containing the success code, and the `jqXHR` object.
5. `complete` callbacks fire, in the order they are registered, when the request finishes, whether in failure or success. They receive the `jqXHR` object, as well as a string containing the success or error code.

For example, to make use of the returned HTML, we can implement a `success` handler:

```
$.ajax({
  url: 'ajax/test.html',
  success: function(data) {
    $('.result').html(data);
    alert('Load was performed.');
  }
});
```

Data Types

The `$.ajax()` function relies on the server to provide information about the retrieved data. If the server reports the return data as XML, the result can be traversed using normal XML methods or jQuery's selectors. If another type is detected, such as HTML in the example above, the data is treated as text.

Different data handling can be achieved by using the `dataType` option. Besides plain `xml`, the `dataType` can be `html`, `json`, `jsonp`, `script`, or `text`.

The `text` and `xml` types return the data with no processing. The data is simply passed on to the success handler, either through the `responseText` or `responseXML` property of the `jqXHR` object, respectively.

Note: We must ensure that the MIME type reported by the web server matches our choice of `dataType`. In particular, XML must be declared by the server as `text/xml` or `application/xml` for consistent results.

If `html` is specified, any embedded JavaScript inside the retrieved data is executed before the HTML is returned as a string. Similarly, `script` will execute the JavaScript that is pulled back from the server, then return nothing.

The `json` type parses the fetched data file as a JavaScript object and returns the constructed object as the result data. To do so, it uses `jQuery.parseJSON()` when the browser supports it; otherwise it uses a `Function` **constructor**. Malformed JSON data will throw a

parse error (see json.org for more information). JSON data is convenient for communicating structured data in a way that is concise and easy for JavaScript to parse. If the fetched data file exists on a remote server, specify the `jsonp` type instead.

The `jsonp` type appends a query string parameter of `callback=?` to the URL. The server should prepend the JSON data with the callback name to form a valid JSONP response. We can specify a parameter name other than `callback` with the `jsonp` option to `$.ajax()`.

Note: JSONP is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. More information about it can be found in the original post detailing its use.

When data is retrieved from remote servers (which is only possible using the `script` or `jsonp` data types), the `error` callbacks and global events will never be fired.

Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by setting a value for the `type` option. This option affects how the contents of the `data` option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHTTPRequest standard.

The `data` option can contain either a query string of the form `key1=value1&key2=value2`, or a map of the form `{key1: 'value1', key2: 'value2'}`. If the latter form is used, the data is converted into a query string using `jQuery.param()` before it is sent. This processing can be circumvented by setting `processData` to `false`. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the `contentType` option from `application/x-www-form-urlencoded` to a more appropriate MIME type.

Advanced Options

The `global` option prevents handlers registered using `.ajaxSend()`, `.ajaxError()`, and similar methods from firing when this request would trigger them. This can be useful to, for example, suppress a loading indicator that was implemented with `.ajaxSend()` if the requests are frequent and brief. With cross-domain script and JSONP requests, the global option is automatically set to `false`. See the descriptions of these methods below for more details. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup()` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or `jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes prefilters, converters, and transports that allow you to extend Ajax with a great deal of flexibility. For more information about these

advanced features, see the Extending Ajax page.

Additional Notes:

- Due to browser security restrictions, most "Ajax" requests are subject to the same origin policy; the request can not successfully retrieve data from a different domain, subdomain, or protocol.
- Script and JSONP requests are not subject to the same origin policy restrictions.

Examples:

Example: *Load and execute a JavaScript file.*

```
$.ajax({
   type: "GET",
   url: "test.js",
   dataType: "script"
 });
```

Example: *Save some data to the server and notify the user once it's complete.*

```
$.ajax({
   type: "POST",
   url: "some.php",
   data: "name=John&location=Boston",
   success: function(msg){
     alert( "Data Saved: " + msg );
   }
 });
```

Example: *Retrieve the latest version of an HTML page.*

```
$.ajax({
  url: "test.html",
  cache: false,
  success: function(html){
    $("#results").append(html);
  }
});
```

Example: *Loads data synchronously. Blocks the browser while the requests is active. It is better to block user interaction by other means when synchronization is necessary.*

```
var html = $.ajax({
  url: "some.php",
  async: false
 }).responseText;
```

Example: *Sends an xml document as data to the server. By setting the processData option to false, the automatic conversion of data to strings is prevented.*

```
var xmlDocument = [create xml document];
 $.ajax({
   url: "page.php",
   processData: false,
   data: xmlDocument,
   success: handleResponse
 });
```

Example: *Sends an id as data to the server, save some data to the server and notify the user once it's complete. Note that this usage - returning the result of the call into a variable - requires a synchronous (blocking) request! (async:false)*

```
var bodyContent = $.ajax({
      url: "script.php",
      global: false,
      type: "POST",
      data: {id : this.getAttribute('id')},
      dataType: "html",
      async:false,
      success: function(msg){
         alert(msg);
      }
   }
).responseText;
```

Support and Contributions

Need help with **jQuery.ajax()** or have a question about it? Visit the jQuery Forum or the **#jquery** channel on irc.freenode.net.

Think you've discovered a jQuery bug related to **jQuery.ajax()**? Report it to the jQuery core team.

Found a problem with this documentation? Report it to the jQuery API team.

DISQUS

Comments for this page are closed.

Showing 20 of 84 comments

Sort by Newest first Subscribe by email Subscribe by RSS

fredrick 8 months ago

Here is a good usage example
http://sharepoint-snippets.com...

29 people liked this. Like

Bob 8 months ago

ajax-in-function-with-custom-parameter-for-persistence - http://forum.jquery.com/topic/...

23 people liked this. Like

Asdas 8 months ago

how i can send string: !@#$%^&*()"

7 people liked this. Like

Albanx 8 months ago in reply to Asdas

user encodeuricomponent(!@#$%^&*()")

27 people liked this. Like

McCoy 8 months ago

I have the following 3 lines:
$(document).ready(function(){
1 $("#personSearchForm ").submit(function(){
2 $("body").css("cursor", "progress") ;
3 jsonobj=$.ajax({url:server_url, async:false.....

Changing the cursor to progress gets executed after the ajax call - weird !

35 people liked this. Like

Linus Unnebäck 8 months ago in reply to McCoy

It's not weird since you have async:false. The page then never gets to update the styles, or for that matter anything. Use setTimeout(..., 0); or async:true to work around the issue.

Billy Braga and 18 more liked this Like

Devlshone 8 months ago

Could someone explain to me why this line would be causing my application to bomb:

xhr.send(noContent || s.data == null ? null : s.data);

13 people liked this. Like

nyuszika7h 8 months ago in reply to Devlshone

If `s.data` is null, you don't need another check for it. Just use this:
`xhr.send(noContent || s.data)`

19 people liked this. Like

Chrissy 9 months ago

Is there anyway to set priority? I do 50 $.ajax calls to get certain information regarding the results. Once the page loads ($.ajax posts are still coming back) and the user clicks on the link, then that new ajax xhr is at the bottom of the list. Am I not utilizing this properly? Kind of confused on what my options are.

20 people liked this. Like

Tom Sartain 9 months ago in reply to Chrissy

You should probably consider restructuring the way that you're doing things.

Try to consolidate your calls to the server when you can. If you know that you'll need to do so many calls as soon as the page loads, batch them up into a handful of calls instead.

Leonardo Silveira and 15 more liked this Like

marcelo 9 months ago

How can I send a multipart form-data uploading form with jQuery Ajax? I have a form sending text fields and image file. When image file is large, greater than server is setting, the text fields and image file are null on the server loosing the data of forms on response. I would like use Ajax (with jQuery) to solve this issue, but $('form').serialize() don't work with image tag.

28 people liked this. Like

Washplanjack 9 months ago in reply to marcelo

Hi Marcelo have you tried using an input type image and pass in a name attribute as well as a Id attribute to get this to work

8 people liked this. Like

Greasy Sneakers 10 months ago

Here is a plugin i found to make ajax calls for user input more efficient:

http://github.com/dougle/jQuer...

I'm using this plugin to limit load on my server, working well so far.

17 people liked this. Like

Dave_degraaf 10 months ago

when i use the second example then i get the source code in the alert box

10 people liked this. Like

TheBizzTech 10 months ago in reply to Dave_degraaf

In my php script I was getting back the source code of the html in the page so I just used echo to only return the data I needed via ajax.

7 people liked this. Like

Meher Ranjan 10 months ago in reply to Dave_degraaf

make use of the datatype parameter, like if its a json use data type json and then you can use it as json. Though jQuery intelligently understands the data, sometimes specifying explicitly makes it easy. See the last example for usage.

12 people liked this. Like

Claudio M. Alessi 11 months ago

I think there should be a paragraph about ajax file uploads, which seems to be a common misunderstanding of the $.ajax() jQuery method. It would be nice to explain that, for portability reasons, the http PUT should *not* be used and that you *can't* upload files via $.ajax() without using an iframe (or non javascript-based technologies like flash, java, and so forth).

abcnewsalbania and 57 more liked this Like

Kozie 11 months ago

For users who also want to execute an AJAX request, in safari, in the close events.. use the synchronous method; asynchronous methods are ignored.

6 people liked this. Like

whatevssssss 11 months ago

Note: as a personal experience, catching an exception thrown from an asynchronous call cannot be caught (with try ... catch) in most browsers (in Firefox the exception gets caught, at least in Firefox 3.6.10).

In jquery 1.4.2, even with "async: false" the exceptions thrown by "error" callbacks DISAPPEAR SILENTLY. After doing some debugging I saw this line:

var onreadystatechange = xhr.onreadystatechange = function(isTimeout) {

which makes the synchronous calls to behave asynchronous, thus making it impossible to catch an exception thrown by the error callback.

To fix this the line should change to:

```
var onreadystatechange = function( isTimeout ) {
....
(and after the onreadystatechange block add)
if (s.async)
{
xhr.onreadystatechange = onreadystatechange;
}
```

22 people liked this. Like

Axelvitali 1 year ago

calling a Webservice - Webmethod Hello(). Got a response {"d","hello"}. In FireFox call to Success function first and then Error function .In Crhome just call the success. In IE 8 just call the error and giving a parsererror. Any ideas? Thanks

18 people liked this. Like

Load more comments