How to Use jQuery's $.ajax() Function

Aurelio De Rosa August 26, 2015 +134

Without any doubt Ajax has taken web development by storm and it's one of the most successful paradigms ever. In my article *An Introduction to jQuery's Shorthand Ajax Methods*, I discussed some of jQuery's most used Ajax shorthand methods: `$.get()`, `$.post()`, and `$.load()`. They are convenient methods for making Ajax requests in a few lines of code.

Sometimes, we need more control over the Ajax calls we want to make. For example, we want to specify what should happen in case an Ajax call fails or we need to perform an Ajax request but its result is only needed if retrieved within a certain amount of time. In such situations, we can rely on another function provided by jQuery, called `$.ajax()`, that is the topic of this tutorial.

The `$.ajax()` Function

More from this author
- An Introduction to jQuery's Deferred Objects
- Building a Mobile JavaScript Powered Audio Player

The jQuery's `$.ajax()` function is used to perform an asynchronous HTTP request. It was added to the library a long time ago, existing since version 1.0. The `$.ajax()` function is what every function discussed in the previously mentioned article calls behind the scene using a preset configuration. The signatures of this function are shown below:

```
$.ajax(url[, options])
$.ajax([options])
```

The `url` parameter is a string containing the URL you want to reach with the Ajax call, while `options` is an object literal containing the configuration for the Ajax request.

In its first form, this function performs an Ajax request using the `url` parameter and the options specified in `options`. In the second form, the URL is specified in the `options` parameter, or can be omitted in which case the request is made to the current page.

The list of the options accepted by this function, described in the next section, is very long. So, I'll keep their description short. In case you want to study in-depth their meaning, you can refer to the official documentation of `$.ajax()`.

The `option` Parameter

There are a lot of different options you can specify to bend `$.ajax()` to your need. In the list below you can find their names and their description sorted in alphabetic order:

- `accepts`: The content type sent in the request header that tells the server what kind of response it will accept in return
- `async`: Set this options to `false` to perform a synchronous request
- `beforeSend`: A pre-request callback function that can be used to modify the `jqXHR` object before it is sent
- `cache`: Set this options to `false` to force requested pages not to be cached by the browser
- `complete`: A function to be called when the request finishes (after `success` and `error` callbacks are executed)
- `contents`: An object that determines how the library will parse the response
- `contentType`: The content type of the data sent to the server
- `context`: An object to use as the context (`this`) of all Ajax-related callbacks
- `converters`: An object containing dataType-to-dataType converters
- `crossDomain`: Set this property to `true` to force a cross-domain request (such as JSONP) on the same domain
- `data`: The data to send to the server when performing the Ajax request
- `dataFilter`: A function to be used to handle the raw response data of XMLHttpRequest
- `dataType`: The type of data expected back from the server
- `error`: A function to be called if the request fails
- `global`: Whether to trigger global Ajax event handlers for this request
- `headers`: An object of additional headers to send to the server
- `ifModified`: Set this option to `true` if you want to force the request to be successful only if the response has changed since the last request
- `isLocal`: Set this option to `true` if you want to force jQuery to recognize the current environment as "local"
- `jsonp`: A string to override the callback function name in a JSONP request
- `jsonpCallback`: Specifies the callback function name for a JSONP request
- `mimeType`: A string that specifies the mime type to override the XHR mime type
- `password`: A password to be used with XMLHttpRequest in response to an HTTP access authentication request
- `processData` : Set this option to `false` if you don't want that the data passed in to the `data` option (if not a string already) will be processed and transformed into a query string
- `scriptCharset`: Sets the charset attribute on the script tag used in the request but only applies when the "script" transport is used
- `statusCode`: An object of numeric HTTP codes and functions to be called when the response has the corresponding code
- `success`: A function to be called if the request succeeds
- `timeout`: A number that specifies a timeout (in milliseconds) for the request
- `traditional`: Set this to `true` if you wish to use the traditional style of param serialization
- `type`: The type of request to make, which can be either "POST" or "GET"
- `url`: A string containing the URL to which the request is sent
- `username`: A username to be used with XMLHttpRequest in response to an HTTP access authentication request
- `xhr`: A callback for creating the XMLHttpRequest object
- `xhrFields`: An object to set on the native XHR object

That was pretty long, isn't it? Well, as developers you probably have stopped reading this list at the fifth or sixth element I guess, but that's fine. The next section will be more exciting because we'll put the `$.ajax()` function and some of these options into action.

Putting It All Together

In this section we'll see this function and some of its options in action.

A First Example of `$.ajax()`

We'll start with a simple demo that reproduces the same code we developed in the previous article, but this time we'll adopt `$.ajax()`. The code I'm talking about is shown below for your commodity:

```
$('#main-menu a').click(function(event) {
   event.preventDefault();

   $('#main').load(this.href + ' #main *', function(responseText, status) {
      if (status === 'success') {
         $('#notification-bar').text('The page has been successfully loaded');
      } else {
         $('#notification-bar').text('An error occurred');
      }
   });
});
```

Updating this snippet to employ the `$.ajax()` function, we obtain the code shown below:

```
$('#main-menu a').click(function(event) {
   event.preventDefault();

   $.ajax(this.href, {
      success: function(data) {
         $('#main').html($(data).find('#main *'));
         $('#notification-bar').text('The page has been successfully loaded');
      },
      error: function() {
         $('#notification-bar').text('An error occurred');
      }
   });
});
```

Here you can see that I used the first form of the function. I've specified the URL to send the request to as the first parameter and then an object of options as the second parameter. The latter takes advantage of just two of the several properties discussed in the previous section: `success` and `error` to specify what to do in case of success or failure of the request respectively.

Retrieving a Talk From Joind.in Using `$.ajax()`

The second example I want to discuss creates a JSONP request to retrieve some information from a service called Joind.in. The latter is a website where event attendees can leave feedback on an event and its sessions. Specifically, I'm going to create a snippet of code that, using the `$.ajax()` function, retrieves the title and the description of my talk *Modern front-end with the eyes of a PHP developer*.

The code to achieve this goal is as follows:

```
$.ajax({
   url: 'http://api.joind.in/v2.1/talks/10889',
   data: {
      format: 'json'
   },
   error: function() {
      $('#info').html('<p>An error has occurred</p>');
   },
   dataType: 'jsonp',
   success: function(data) {
      var $title = $('<h1>').text(data.talks[0].talk_title);
      var $description = $('<p>').text(data.talks[0].talk_description);
      $('#info')
         .append($title)
         .append($description);
   },
   type: 'GET'
});
```

In the snippet above, I employed several of the properties discussed. First of all, you can see that I'm using the second form of `$.ajax()`, which allows to specify the URL to which the request is sent as a property (`url`) of the object literal. Because the Joind.in's API accepts a JSONP request, in the code above I'm setting the type of request I want to use by specifying the `dataType` property. Then, I used the `data` property to define the format's type that I want to obtain from the server as required by the API. However, the latter requires this data as part of the query string of a GET request, hence I'm also specifying the `type` property setting `GET` as its value. Finally, I wrote an `error` callback to display a message in case of error, and a `success` callback to display the title and the description of the talk in case of success.

A live demo of this code is shown below and is also available as a JSfiddle:

JavaScript | HTML | CSS | Result | Edit in JSFiddle

```
$('#action-button').click(function() {
   $.ajax({
      url: 'http://api.joind.in/v2.1/talks/10889',
      data: {
         format: 'json'
      },
      error: function() {
         $('#info').html('<p>An error has occurred</p>');
      },
      dataType: 'jsonp',
      success: function(data) {
```

Note: in case you need a library to embed a talk from Joind.in, I developed a library called JoindIn.js.

Conclusion

In this tutorial we've discussed the most powerful of the Ajax functions offered by jQuery, `$.ajax()`. It allows you to perform Ajax request with a lot of control over how the request is sent to the server and how the response is processed. Thanks to this function you have the tools you need to satisfy every need your project may have in case none of the shorthand functions are a good fit.

To have an even better understanding of the potential of this function, I encourage you to play with the code samples, and to try to modify it to use some other options accepted by the `options` parameter. Have fun!

Aurelio De Rosa

I'm a (full-stack) web and app developer with more than 5 years' experience programming for the web using HTML, CSS, Sass, JavaScript, and PHP. I'm an expert of JavaScript and HTML5 APIs but my interests include web security, accessibility, performance, and SEO. I'm also a regular writer for several networks, speaker, and author of the books *jQuery in Action, third edition* and *Instant jQuery Selectors*.

Recommended for you
- Make a Voice-Controlled Audio Player with the Web Speech API
- A Comparison of JavaScript HTTP Libraries for the Browser
- Bringing Componentization to the Web: An Overview of Web Components
- 11 Best jQuery Charting Libraries
- Building a Twitter App Using AngularJS

COURSES >
JavaScript: Next Steps — M. David Green — 3:07:36
React The ES6 Way — Darin Haener — 1:11:20
Your First Meteor 1.2 Application — David Turnbull — 1:49:07

BOOKS >
ECMAScript 2015: A SitePoint Anthology — James Hibbard
Full Stack JavaScript Development with MEAN — Colin Ihrig
JavaScript: Novice to Ninja — Darren Jones

SCREENCASTS >
Building Realtime Web Apps with Firebase — Thomas Greco
Using Helpers and Conditionals in Ember — Thomas Greco
Modifying a Bootstrap Theme with Handlebars — Kauress