jQuery AJAX

Jakob Jenkov

Last update: 2014-09-26

The jQuery AJAX features makes it possible and easy use AJAX in your HTML pages. The term *AJAX* is short for Asynchronous Javascript And XML. AJAX makes it possible to fetch content from a server in the background (asynchronously), and update parts of your page with the new content - all without having to reload the complete HTML page.

The jQuery AJAX features are very advanced, and very comprehensive. I will cover most of jQuery's AJAX features in this text, but you can lookup the finer detail in jQuery's AJAX documentation:

jQuery AJAX.

jQuery AJAX Example

Here is first a jQuery AJAX example showing how to make an AJAX call in jQuery:

```
var jqxhr =
    $.ajax({
        url: "/theServiceToCall.html",
        data: {
            name : "The name",
            desc : "The description"
        }
    })
    .done  (function(data, textStatus, jqXHR)        { alert("Success: " + response) ; })
    .fail  (function(jqXHR, textStatus, errorThrown) { alert("Error")   ; })
    .always(function(jqXHROrData, textStatus, jqXHROrErrorThrown)     { alert("complete"); })
    ;
```

First the `$.ajax()` function is called. To this function is passed a JavaScript object which contains information about the AJAX call to make. In the example this object contains the URL and data to be sent to the server.

The `$.ajax()` function returns an object. On this object the example calls three functions: `done()`, `fail()` and `always()`.

The `done()` function is given a function as parameter. The callback function passed as parameter to the `done()` function is executed if the AJAX request succeeds. The callback function gets three parameters: `data`, `textStatus` and `jqXHR`. The `data` parameter is the data returned by the server. The `textStatus` parameter is the textual status message returned by the server. The `jqXHR` parameter is the jqXHR object which is also returned by the `$.ajax()` function.

The `fail()` is also given a function as parameter. The callback function passed as parameter to the `fail()` function is called if the AJAX request fails. The callback function gets three parameters: `jqXHR`, `textStatus` and `errorThrown`. The `jqXHR` parameter is the `jqXHR` object also returned by the `$.ajax()` function. The `textStatus` is the textual status message returned by the server. The `errorThrown` parameter is the error thrown by jQuery.

The callback function passed to the `always()` function is called whenever the AJAX request finishes, regardless of whether or not the AJAX request succeeds or fails. The three parameters passed to the callback function will be either the same three parameters passed to `done()` or `fail()`, depending on whether the AJAX request succeeds or fails.

jQuery AJAX Before Version 1.8.0

Before version 1.8.0 of jQuery the object returned from the jQuery `$.ajax()` function did not contain the `done()`, `fail()` and `always()` functions. Instead, these functions were called `success()`, `error()` and `complete()`.

jQuery AJAX After Version 1.8.0

From jQuery 1.8.0 the `$.ajax()` function returns a `jqXHR` object which implements the promise interface (`done()`, `fail()`, `always()` etc.) instead of of the `success()`, `error()` and `complete()` functions. The `success()`, `error()` and `complete()` functions are now deprecated. The promise interface is described in my text about **jQuery deferred objects**.

Receiving HTML With AJAX

By default jQuery AJAX function does not parse the data received from the server. You can insert it raw into a `div`, like this:

```
var jqxhr =
    $.ajax({
        url: "/theServiceToCall.html",
        data: {
            name : "The name",
            desc : "The description"
        }
    })
    .done (function(data) { $('#theDiv').html(data); })
    .fail   (function()     { alert("Error")   ; })
    ;
```

Notice how the `done()` function selects a `div` with the id `theDiv`, and calls its `html()` function, passing as parameter the data received from the server.

Receiving JSON With AJAX

If you want jQuery to interpret the data received from the server as JSON, you must add the `dataType : 'json'` field to the JavaScript object passed as parameter to the `$.ajax()` call. Imagine you get a JSON object back from the server that looks like this:

```
{
    "param1" : "hello world"
}
```

Then you can parse that JSON object, and reference the `param1` property, with this `$.ajax()` call:

```
var jqxhr =
    $.ajax({
        url: "/theServiceToCall.json",
        dataType: 'json',
        data: {
            name : "The name",
            desc : "The description"
        }
    })
    .done (function(data) { alert("Success: " + data.param1) ; })
    .fail   (function()     { alert("Error")   ; })
    ;
```

Sending Parameters in The AJAX Request

You can send parameters to the server via jQuery's AJAX functions. You have already seen examples of that. It's the `data` property of the JavaScript object passed to the `$.ajax()` function which contain the data to send to the server. Here is an example with the `data` object marked in bold:

```
var jqxhr =
    $.ajax({
        url: "/theServiceToCall.html",
        data: {
            name : "The name",
            desc : "The description"
            }
    })
    .done (function(data) { alert("Success: " + data.param1) ; })
    .fail   (function()     { alert("Error")   ; })
    ;
```

The `data` property should always be a JavaScript object. It's properties are serialized into a regular query string (for GET requests), or a normal post body parameter string (for POST requests). This serialized string is then sent to the server, along with the AJAX request.

On the server you can read the properties of the `data` object as if they were sent as simple request parameters, via either GET or POST. Just like if the properties had been fields in a form. In the example above, the server would be able to read two request properties: `name` and `desc`.

Sending Raw Data in The AJAX Request

If you do not want jQuery to convert the `data` object into a serialized parameter string, you can avoid this by setting the `processData : false` in the JavaScript object passed to the `$.ajax()` function. Here is an example:

```
var jqxhr =
    $.ajax({
        url: "/test.jsp",
        processData : false,
        type : "POST",
        data: "THE DATA"
    })
    .done (function(data) { $('#ajaxDiv').html(data) })
    .fail   (function()  { alert("Error ")   ; })
    ;
```

Notice the `processData` property which is set to `false`. This tells jQuery not to process the `data` property before sending it to the server.

Second, notice the `type` property, which is set to `"POST"`. This tells jQuery to POST the data to the server.

Third, notice how the `data` property is now just a string of raw data to be sent to the server. You cannot use a JavaScript object as `data` property, when you do not process the data before sending them.

Sending JSON in The AJAX Request

If you need to send a JavaScript object as a JSON string to the server with an AJAX request, you need to convert the JavaScript object to a JSON string and send the string as raw data. Here is a jQuery JSON post example:

```
var theObject = { p1: "v1", p2 : "v2 };

var jqxhr =
    $.ajax({
        url: "/test.jsp",
        processData : false,
        type : "POST",
        data: JSON.stringify(theObject)
```

```
        data: JSON.stringify(theObject)
    })
    .done (function(data) { $('#ajaxDiv').html(data) })
    .fail (function()  { alert("Error ")   ; })
    ;
```

HTTP GET and POST

HTTP requests can be send as either, GET, POST, PUT, DELETE or HEAD requests. Here I will show you how to do a GET and POST request. Since AJAX requests are HTTP requests, you can also specify which HTTP method to use with your jQuery AJAX requests.

As you have seen earlier, the HTTP method to use can be passed to the `$.ajax()` via its JavaScript parameter object. You do so by setting the `type` parameter of the parameter object. Here is first a HTTP GET example using jQuery's AJAX function:

```
var jqxhr =
    $.ajax({
        url: "/target.jsp",
        type : "GET",
        data: {
            param1 : "value1",
            param2 : "value2"
        }
    })
    .done (function(data) { /* process data */ })
    .fail (function()  { alert("Error ")   ; })
    ;
```

And here is a HTTP POST example using jQuery's AJAX function:

```
var jqxhr =
    $.ajax({
        url: "/target.jsp",
        type : "POST",
        data: {
            param1 : "value1",
            param2 : "value2"
        }
    })
    .done (function(data) { /* process data */ })
    .fail (function()  { alert("Error ")   ; })
    ;
```

The $.get() and $.post() Functions

jQuery has two functions that can be used to send simplified HTTP GET and HTTP POST requests. These functions are the `$.get()` and `$.post()` functions.

Here is an example showing how to use jQuery's `$.get()` function:

```
var parameters = { p1 : "val1", p2 : "val2"};
$.get("data.html", parameters )
    .done(function(data) {
        $("#targetElement").html(data);
    }) ;
```

The `$.get()` function takes a URL and a request parameter object as parameters. The `$.get()` function returns a `jqXHR` object just like the `$.ajax()` function. The handling of the response is thus similar to how you handle the response with the `$.ajax()` function.

how you handle the response with the `$.ajax()` function.

jQuery's `$.post()` function works in the same way. Here is an example:

```
var parameters = { p1 : "val1", p2 : "val2"};
$.post("data.html", parameters )
    .done(function(data) {
        $("#targetElement").html(data);
    }) ;
```

The $.getJSON() Function

The `$.get()` and `$.post()` functions do not process the data returned by the server. If you want the data returned by the server interpreted as JSON, you can use jQuery's `$.getJSON()` function: Here is a `$.getJSON()` example:

```
var parameters = { p1 : "val1", p2 : "val2"};
$.getJSON("data.json", parameters )
    .done(function(data) {
        $("#getJSONTarget").html(data.param1);
    }) ;
```

As you can see, the `$.getJSON()` function works much like the `$.get()` and `$.post()` functions. The only difference is that the `data` parameter passed to the callback function set via `done()` is now a JavaScript object.

The load() Function

jQuery also has a function called `load()` which can be called on a selected element. The `load()` element loads some HTML via AJAX and inserts it into the selected element. Here is a jQuery `load()` example:

```
$("#loadTarget").load("html-fragment.html");
```

And with request parameters:

```
var parameters = { p1 : "val1", p2 : "val2"};
$("#loadTarget").load("html-fragment.html", parameters);
```

And with a callback that is called when `load()` finishes:

```
var parameters = { p1 : "val1", p2 : "val2"};
$("#loadTarget").load("html-fragment.html", parameters, function() {
    console.log("load done");
});
```

You can also insert just a part of the HTML loaded. If you append a space + jQuery selector string after the url then `load()` will only inserted the part of the loaded HTML matching the selector. Here is an example:

```
$("#loadTarget2").load("html-fragment.jsp #div2");
```

This example loads the `html-fragment.jsp` HTML fragment, selects the element with the id `div2` from that fragment and inserts only that element, regardless of what more the HTML fragment contains.

Note: If the loaded HTML contains any JavaScript it will get executed when the HTML is inserted into the target HTML element. However, if you load a fragment (URL + jQuery selector) then any JavaScript

Note: If the loaded HTML contains any JavaScript it will get executed when the HTML is inserted into the target HTML element. However, if you load a fragment (URL + jQuery selector) then any JavaScript found in the loaded file is remove before the HTML is inserted. In general, don't use `load()` to load JavaScript (unless you absolutely need to load HTML and JavaScript together). jQuery has the `$.getScript()` function for that purpose.

The $.getScript() Function

The `$.getScript()` function in jQuery loads a JavaScript file and executes it. This function uses jQuery's underlying AJAX functions so the `$.getScript()` function cannot load scripts from other domains than the HTML page making the request was loaded from (just like with ordinary AJAX calls).

Here is a jQuery `$.getScript()` example:

```
$.getScript("sample-script.js");
```

A jQuery `$.getScript()` with parameters example:

```
var parameters = {};
$.getScript("sample-script.js", parameters);
```

And a jQuery `$.getScript()` with callback example:

```
var parameters = {};
$.getScript("sample-script.js", parameters, function() {
    console.log("sample-script.js loaded");
});
```

The last function call would also work with the `parameters` object omitted.

Global AJAX Functions

jQuery has a set of global AJAX functions which you can use to listen for AJAX events across all AJAX requests sent via jQuery. These global AJAX functions are:

- .ajaxSend()
- .ajaxStart()
- .ajaxStop()
- .ajaxSuccess()
- .ajaxError()
- .ajaxComplete()

The callback function registered with the `ajaxSend()` function is called before every AJAX request is sent. Here is an example:

```
$(document).ajaxSend(function() {
    console.log("called before each send");
});
```

Notice that the `ajaxSend()` is called on a jQuery selection object.

The callback function registered with the `ajaxStart()` function is called before an AJAX request is sent, if there are no currently executing AJAX requests. Here is an example:

```
$(document).ajaxStart(function() {
    console.log("called before each AJAX request if no other request are executing");
});
```

The callback function registered with the `ajaxStop()` function is called after an AJAX request finishes, if there are no other executing AJAX requests. Here is an example:

```
$(document).ajaxStop(function() {
    console.log("called after an AJAX request finishes, if no other request are executing");
});
```

The callback function registered with the `ajaxSuccess()` function is called whenever an AJAX request succeeds. Here is an example:

```
$(document).ajaxSuccess(function(event, jqxhr, ajaxOptions, data) {
    console.log("called if an AJAX request succeeds");
});
```

The callback function registered with the `ajaxError()` function is called whenever an AJAX request fails. Here is an example:

```
$(document).ajaxError(function(event, jqxhr, ajaxOptions, errorThrown) {
    console.log("called if an AJAX request fails");
});
```

The callback function registered with the `ajaxComplete()` function is called whenever an AJAX request completes, regardless of whether the AJAX request succeeds or fails. Here is an example:

```
$(document).ajaxComplete(function(event, jqxhr, ajaxOptions) {
    console.log("called after an AJAX request completes");
});
```

The jqXHR Object

The `jqXHR` object returned by many of jQuery's AJAX functions contains some useful information. Note, that some of this information is not available until the web server has sent back a response, meaning it is available from inside the `done()`, `fail()` or `always()` callback functions.

The `jqXHR` object contains these properties and functions:

- `status`
- `statusText`
- `responseText`
- `responseXML`
- `getAllResponseHeaders()`
- `getResponseHeader()`
- `abort()`

The `status` property contains the HTTP status code (e.g. 200 or 404 etc.) sent back by the web server.

The `statusText` property contains the text `success` or `error` depending on whether the AJAX request succeeded or failed.

The `responseText` contains the body of the HTTP response sent back by the server, if the response is sent as text (e.g. content type `text/html`, `text/plain` or `application/json`).

The `responseXML` property contains the body of the HTTP response sent back by the server, if the response is sent back as XML (e.g. content type `text/xml` or `application/xml`).

The `getAllResponseHeaders()` return the HTTP response headers as a string. Each header is listed on its own line with a header name, colon and header value. For instance:

```
Content-Encoding: gzip
Server: Jetty(9.2.1.v20140609)
```

```
Content-Length: 54
Content-Type: application/json
```

The `getResponseHeader()` function can be used to access individual HTTP response headers. You pass the name of the HTTP header like this:

```
var contentType = jqXHR.getResponseHeader("Content-Type");
```

The `abort()` function on the `jqXHR` object can be used to abort the AJAX (HTTP) request before it finishes. You should call this function before any of the `done()`, `fail()` or `always()` callback functions are called. These callback functions are called when the server has sent back a response, and at that time it is too late to abort the AJAX call.

Handling Errors

If an AJAX request fails, you can react to the failure inside the callback function added via the `fail()` function of the object returned by the `$.ajax()` function. Here is a jQuery AJAX error handling example:

```
var jqxhr =
    $.ajax({
        url: "/this-page-does-not-exist.jsp",
    })
    .done (function(data) { /* will never happen - page not found*/)
    .fail (function(jqxhr, textStatus, errorThrown)  {
        alert("Error: " + textStatus + " : " + errorThrown) ;
    })
    ;
```

The callback function passed to the `done()` function will get executed if an error occurs in the AJAX request above. Inside this callback function you can handle the error. Handling the error will normally consist of notifying the user that the request failed. This example displays an alert, but you could also insert the error message into an HTML element somewhere in the page.

Remember, you have access to the HTTP status code returned with response from the server. In some cases you may need to react differently depending on the value of this code. Here is a simple example:

```
var jqxhr =
    $.ajax({
        url: "/this-page-does-not-exist.jsp",
    })
    .done (function(data) { /* will never happen - page not found*/)
    .fail (function(jqxhr, textStatus, errorThrown)  {
        if(jqxhr.status == 404) {
            alert("page not found!");
        }
    })
    ;
```

You may also look at the response headers or the response body. Here is an example of that:

```
var jqxhr =
    $.ajax({
        url: "/this-page-does-not-exist.jsp",
    })
    .done (function(data) { /* will never happen - page not found*/)
    .fail (function(jqxhr, textStatus, errorThrown)  {
        var contentType   = jqxhr.getResponseHeader("Content-Type");
        var responseBody  = jqxhr.responseText;

        //do something depending on response headers and response body.
    })
    ;
```

ajaxError()

Often, you will handle all AJAX errors in the same way. Instead of having to set `fail()` handlers on every single AJAX call, you can use the global function `ajaxError()` to handle a single AJAX error callback function. Here is a jQuery `ajaxError()` example:

```
$(document).ajaxError(function(event, jqxhr, ajaxOptions, errorThrown) {
        var contentType   = jqxhr.getResponseHeader("Content-Type");
        var responseBody  = jqxhr.responseText;

        //do something depending on response headers and response body.
});

var jqxhr =
    $.ajax({
        url: "/this-page-does-not-exist.jsp",
    })
    .done (function(data) { /* will never happen - page not found*/)
    ;
```

Notice how no `fail()` handler is attached to the AJAX request. AJAX error handling for all AJAX requests is now handled by the callback function passed to `$(document).ajaxError()`. You have access to the `jqXHR` object in this callback too, as well as the AJAX options object that produced the request that failed etc.

$.ajaxSetup()

The `$.ajaxSetup()` function can be used to set options to be used for all AJAX calls, including those performed via `$.ajax()`, `load()`, `$.get()` etc. The options you can set are the same options as you can pass to a `$.ajax()` call. For instance, this example sets the `type` property of all AJAX calls to `POST`:

```
$.ajaxSetup({
    type : "POST"
});
```

Now all AJAX calls will be HTTP POST requests unless a request explicitly overrides that property. An AJAX request can override that like this:

```
$.ajax({
    url  : "the-service.json",
    type : "GET"
});
```

By explicitly setting the `type` property in the AJAX options object, the `$.ajax()` call overrides the global settings setup via `$.ajaxSetup()`

The jQuery documentation recommends that you use `$.ajaxSetup()` with care. Setting global options can lead to unforeseen side effects (AJAX request being fired with the wrong options).

$.ajaxPrefilter()

The `$.ajaxPrefilter()` function in jQuery is used to set a pre-filtering function that can filter all AJAX calls before they are sent. By filtering is meant that the AJAX options object passed to the `$.ajax()` function can be changed ("filtered") before the request is sent.

Here is a jQuery AJAX `$.ajaxPrefilter()` example:

```
$.ajaxPrefilter(function(options, originalOptions, jqXHR){

    if(options.url.indexOf("/app") != -1) {
```

```
        options.type = "POST";
    }
});
```

This example sets a prefilter function which checks if the `options` object's `url` property contains the substring "/app". If it does, the `options.type` is set to `POST`.

The `options` parameter passed to the filter function is the options object which is about to be processed by the `$.ajax()` function. This object contains a merge of the settings set via `$.ajaxSetup()` and the options object passed to the `$.ajax()` function when it was called.

The `originalOptions` object contains the options object that was passed to the `$.ajax()` function without any options merged in from the options set via `$.ajaxSetup()`.

The `jqXHR` object is the normal `jqXHR` object which will be used to execute this AJAX request.

The main difference between the `$.ajaxSetup()` function and `$.ajaxPrefilter()` function is, that the `$.ajaxSetup()` function takes a static options object as parameter. These options will be applied to all AJAX requests. The `$.ajaxPrefilter()` takes a function as parameter. A function can perform more intelligent filtering of the options than a simple merge of default options with request options.

Next: jQuery Deferred Objects