7 AJAX Methods

The jQuery library has a full suite of AJAX capabilities. The functions and methods therein allow us to load data from the server without a browser page refresh. In this chapter, we'll examine each of the available AJAX methods and functions. We'll see various ways of initiating an AJAX request, as well as several methods that can observe the requests that are in progress at any time.

Some of the examples in this chapter use the `$.print()` function to print results to the page. This is a simple plug-in, which will be discussed in Chapter 10, *Plug-in API*.

Low-level interface

These methods can be used to make arbitrary AJAX requests.

$.ajax()

Perform an asynchronous HTTP (AJAX) request.

```
$.ajax(settings)
```

Parameters

- `settings`: A map of options for the request. It can contain the following items:
    - `url`: A string containing the URL to which the request is sent.
    - `async` (optional): A Boolean indicating whether to perform the request asynchronously. Defaults to `true`.

- `beforeSend` (optional): A callback function that is executed before the request is sent.
- `cache` (optional): A Boolean indicating whether to allow the browser to cache the response. Defaults to `true`.
- `complete` (optional): A callback function that executes whenever the request finishes.
- `contentType` (optional): A string containing a MIME content type to set for the request. Defaults to `application/x-www-form-urlencoded`.
- `context` (optional): An object (typically a DOM element) to set as `this` within the callback functions. Defaults to `window`. New in jQuery 1.4.
- `data` (optional): A map or string that is sent to the server with the request.
- `dataFilter` (optional): A callback function that can be used to preprocess the response data before passing it to the `success` handler.
- `dataType` (optional): A string defining the type of data expected back from the server (`xml`, `html`, `json`, `jsonp`, `script`, or `text`).
- `error` (optional): A callback function that is executed if the request fails.
- `global` (optional): A Boolean indicating whether global AJAX event handlers will be triggered by this request. Defaults to `true`.
- `ifModified` (optional): A Boolean indicating whether the server should check if the page is modified before responding to the request. Defaults to `false`.
- `jsonp` (optional): A string containing the name of the JSONP parameter to be passed to the server. Defaults to `callback`.
- `password` (optional): A string containing a password to be used when responding to an HTTP authentication challenge.
- `processData` (optional): A Boolean indicating whether to convert submitted data from object form into query string form. Defaults to `true`.
- `scriptCharset` (optional): A string indicating the character set of the data being fetched; only used when the `dataType` parameter is `jsonp` or `script`.

  - `success` (optional): A callback function that is executed if the request succeeds.
  - `timeout` (optional): A number of milliseconds after which the request will time out in failure.
  - `type` (optional): A string defining the HTTP method to use for the request, such as `GET` or `POST`. Defaults to `GET`.
  - `username` (optional): A string containing a user name to be used when responding to an HTTP authentication challenge.
  - `xhr` (optional): A callback function that is used to create the `XMLHttpRequest` object. Defaults to a browser-specific implementation.

Return value

The `XMLHttpRequest` object that was created, if any.

Description

The `$.ajax()` function underlies all AJAX requests sent by jQuery. It is rarely necessary to directly call this function, as several higher-level alternatives such as `$.get()` and `.load()` are available and are easier to use. However, if less common options are required, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function must at least specify a URL from which to load the data.

```
$.ajax({
  url: 'ajax/test.html',
});
```

Even this sole required parameter may be made optional by setting a default using the `$.ajaxSetup()` function.

Using the only required option, this example loads the contents of the specified URL; but does nothing with the result. To use the result, we can implement one of the callback functions.

Callback functions

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

- beforeSend is called before the request is sent, and is passed the XMLHttpRequest object as a parameter.
- error is called if the request fails. It is passed the XMLHttpRequest object, a string indicating the error type, and an exception object if applicable.
- dataFilter is called on success. It is passed the returned data and the value of dataType, and must return the (possibly altered) data to pass on to success.
- success is called if the request succeeds. It is passed the returned data, as well as a string containing the success code.
- complete is called when the request is finished, whether in failure or success. It is passed the XMLHttpRequest object, as well as a string containing the success or error code.

To make use of the returned HTML, we can implement a success handler as follows:

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

Such a simple example would generally be better served by using .load() or $.get().

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 preceding example, 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 responseHTML property of the XMLHttpRequest object, respectively.

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, and then return the script itself as textual data.

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 JSON.parse() when the browser supports it; otherwise it uses a Function **constructor**. 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, the jsonp type can be used instead. This type will cause a query string parameter of callback=? to be appended to the URL; the server should prepend the JSON data with the callback name to form a valid JSONP response. If a specific parameter name is desired instead of callback, it can be specified with the jsonp option to $.ajax().

A detailed description of the JSONP protocol is beyond the scope of this reference guide. This protocol is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. Comprehensive treatments can be found online, or in Chapter 7 of the book *Learning jQuery 1.3*.

When data is retrieved from remote servers (which is only possible using the script or jsonp data types), the operation is performed using a <script> tag rather than an XMLHttpRequest object. In this case, no XMLHttpRequest object is returned from $.ajax(), nor is one passed to the handler functions such as beforeSend.

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.

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 before it is sent. This processing can be circumvented by setting processData to false. The processing might be undesirable if we wish to send an XML object to the server. In this case, we would also want to 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. For example, this can be useful to suppress a loading indicator that we implemented with `.ajaxSend()` if the requests are frequent and brief. See the *Descriptions* of these methods 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`. Set `ifModified` to `true` to cause the request to report failure if the asset has not been modified since the last request.

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.

> Rather than making requests synchronous using this option, better results can be had using the **blockUI** plug-in. For more information on using plug-ins, see Chapter 10, *Plug-in API*.

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 it 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.

$.ajaxSetup()

Set default values for future AJAX requests.

```
$.ajaxSetup(settings)
```

Parameters

- `settings`: A map of options for future requests; same possible items as in `$.ajax()`

Return value

None

Description

For details on the settings available for `$.ajaxSetup()`, see `$.ajax()`.

All subsequent AJAX calls using any function will use the new settings, unless overridden by the individual calls, until the next invocation of `$.ajaxSetup()`.

For example, we could set a default for the URL parameter before pinging the server repeatedly as follows:

```
$.ajaxSetup({
  url: 'ping.php'
});
```

Now each time an AJAX request is made, this URL will be used automatically.

```
$.ajax({});
$.ajax({
  data: {'date': Date()}
});
```

Global callback functions should be set with their respective global AJAX event handler methods—`.ajaxStart()`, `.ajaxStop()`, `.ajaxComplete()`, `.ajaxError()`, `.ajaxSuccess()`, and `.ajaxSend()`—rather than within the `settings` object for `$.ajaxSetup()`.

Shorthand methods

These methods perform the more common types of AJAX requests in less code.

$.get()

Load data from the server using a GET HTTP request.

```
$.get(url[, data][, success][, dataType])
```

Parameters

- `url`: A string containing the URL to which the request is sent
- `data` (optional): A map or string that is sent to the server with the request
- `success` (optional): A callback function that is executed if the request succeeds
- `dataType` (optional): A string defining the type of data expected back from the server (`xml`, `html`, `json`, `jsonp`, `script`, or `text`)

Return value

The `XMLHttpRequest` object that was created.

Description

This is a shorthand AJAX function, which is equivalent to the following:

```
$.ajax({
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

The callback is passed the returned data, which will be an XML root element, text string, JavaScript file, or JSON object, depending on the MIME type of the response.

Most implementations will specify a success handler.

```
$.get('ajax/test.html', function(data) {
  $('.result').html(data);
  $.print('Load was performed.');
});
```

This example fetches the requested HTML snippet and inserts it on the page.

.load()

Load data from the server and place the returned HTML into the matched element.

```
.load(url[, data][, success])
```

Parameters

- `url`: A string containing the URL to which the request is sent
- `data` (optional): A map or string that is sent to the server with the request
- `success` (optional): A callback function that is executed if the request succeeds

Return value

The jQuery object, for chaining purposes.

Description

This method is the simplest way to fetch data from the server. It is roughly equivalent to `$.get(url, data, success)`, except that it is a method rather than global function and has an implicit callback function. When a successful response is detected, `.load()` sets the HTML contents of the matched element to the returned data. This means that most uses of the method can be quite simple, for example:

```
$('.result').load('ajax/test.html');
```

The provided callback, if any, is executed after this post-processing has been performed:

```
$('.result').load('ajax/test.html', function() {
  $.print('Load was performed.');
});
```

The POST method is used if data is provided as an object; otherwise, GET is assumed.

The event handling suite also has a method named `.load()`. Which one is fired depends on the set of arguments passed.

Loading page fragments

The `.load()` method, unlike `$.get()`, allows only part of a remote document to be fetched. This is achieved with a special syntax for the `url` parameter. If one or more space characters are included in the string, the portion of the string following the first space is assumed to be a jQuery selector. This selector is used to identify a portion of the remote document to retrieve.

We could modify the preceding example to fetch only part of the document as follows:

```
$('.result').load('ajax/test.html #container');
```

When this method executes, the content of `ajax/test.html` is loaded, but then jQuery parses this returned document to find the element with an ID of `container`. The inner content of this element is inserted into the element with a class of `result` and the rest of the loaded document is discarded.

$.post()

Load data from the server using a POST HTTP request.

```
$.post(url[, data][, success][, dataType])
```

Parameters

- `url`: A string containing the URL to which the request is sent
- `data` (optional): A map or string that is sent to the server with the request
- `success` (optional): A callback function that is executed if the request succeeds
- `dataType` (optional): A string defining the type of data expected back from the server (`xml`, `html`, `json`, `jsonp`, `script`, or `text`)

Return value

The `XMLHttpRequest` object that was created.

Description

This is a shorthand AJAX function, which is equivalent to the following:

```
$.ajax({
  type: 'POST',
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

The callback is passed the returned data, which will be an XML root element or a text string depending on the MIME type of the response.

Most implementations will specify a success handler.

```
  $.post('ajax/test.html', function(data) {
    $('.result').html(data);
    $.print('Load was performed.');
  });
```

This example fetches the requested HTML snippet and inserts it on the page.

Pages fetched with POST are never cached, so the `cache` and `ifModified` options have no effect on these requests.

$.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

```
$.getJSON(url[, data][, success])
```

Parameters

- `url`: A string containing the URL to which the request is sent
- `data` (optional): A map or string that is sent to the server with the request
- `success` (optional): A callback function that is executed if the request succeeds

Return value

The `XMLHttpRequest` object that was created.

Description

This is a shorthand AJAX function, which is equivalent to the following:

```
  $.ajax({
    url: url,
    dataType: 'json',
    data: data,
    success: success
  });
```

The callback is passed the returned data, which will be a JavaScript object or array as defined by the JSON structure and parsed using `JSON.parse()` or a Function constructor.

> For details on the JSON format, see `http://json.org/`.

Most implementations will specify a success handler.

```
$.getJSON('ajax/test.json', function(data) {
  $('.result').html('<p>' + data.foo + '</p>'
    + '<p>' + data.baz[1] + '</p>');
  $.print('Load was performed.');
});
```

This example, of course, relies on the structure of the JSON file.

```
{
  "foo": "The quick brown fox jumps over the lazy dog.",
  "bar": "ABCDEFG",
  "baz": [52, 97]
}
```

Using this structure, the example inserts the first string and second number from the file onto the page.

If there is a syntax error in the JSON file, the request will usually fail silently. Avoid frequent hand-editing of JSON data for this reason.

If the specified URL is on a remote server, the request is treated as JSONP instead. See the *Description* of the `jsonp` data type in *$.ajax()* for more details.

$.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

```
$.getScript(url[, success])
```

Parameters

- `url`: A string containing the URL to which the request is sent
- `success` (optional): A callback function that is executed if the request succeeds

Return value

The `XMLHttpRequest` object that was created.

Description

This is a shorthand AJAX function, which is equivalent to the following:

```
$.ajax({
  url: url,
  type: 'script',
  success: success
});
```

The callback is passed the returned JavaScript file. This is generally not useful as the script will already have run at this point.

The script is executed in the global context, so it can refer to other variables and use jQuery functions. Included scripts should have some impact on the current page.

```
$('.result').html('<p>Lorem ipsum dolor sit amet.</p>');
```

The script can then be included and run by referencing the file name as follows:

```
$.getScript('ajax/test.js', function() {
  $.print('Load was performed.');
});
```

Global AJAX event handlers

These methods register handlers to be called when certain events, such as initialization or completion, take place for any AJAX request on the page.

.ajaxComplete()

Register a handler to be called when AJAX requests complete.

```
.ajaxComplete(handler)
```

Parameters

- `handler`: The function to be invoked

Return value

The jQuery object, for chaining purposes.

Description

Whenever an AJAX request completes, jQuery triggers the `ajaxComplete` event. Any and all handlers that have been registered with the `.ajaxComplete()` method are executed at this time.

To observe this method in action, we can set up a basic AJAX load request as follows:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element.

```
$('.log').ajaxComplete(function() {
  $(this).text('Triggered ajaxComplete handler.');
});
```

Now, we can make an AJAX request using any jQuery method.

```
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

When the user clicks the button and the AJAX request completes, the log message is displayed.

As `.ajaxComplete()` is implemented as a method rather than a global function, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxComplete` handlers are invoked, regardless of what AJAX request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxComplete` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. We can restrict our callback to only handling events dealing with a particular URL, for example:

```
$('.log').ajaxComplete(function(e, xhr, settings) {
  if (settings.url == 'ajax/test.html') {
    $(this).text('Triggered ajaxComplete handler.');
  }
});
```

.ajaxError()

Register a handler to be called when AJAX requests complete with an error.

```
.ajaxError(handler)
```

Parameters

- `handler`: The function to be invoked

Return value

The jQuery object, for chaining purposes.

Description

Whenever an AJAX request completes with an error, jQuery triggers the `ajaxError` event. Any and all handlers that have been registered with the `.ajaxError()` method are executed at this time.

To observe this method in action, we can set up a basic AJAX load request as follows:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element.

```
$('.log').ajaxError(function() {
  $(this).text('Triggered ajaxError handler.');
});
```

Now, we can make an AJAX request using any jQuery method.

```
$('.trigger').click(function() {
  $('.result').load('ajax/missing.html');
});
```

The log message is displayed when the user clicks the button and the AJAX request fails because the requested file is missing.

As `.ajaxError()` is implemented as a method rather than a global function, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxError` handlers are invoked, regardless of what AJAX request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxError` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. If the request failed because JavaScript raised an exception, the exception object is passed to the handler as a fourth parameter. We can restrict our callback to only handling events dealing with a particular URL, for example:

```
$('.log').ajaxError(function(e, xhr, settings, exception) {
  if (settings.url == 'ajax/missing.html') {
    $(this).text('Triggered ajaxError handler.');
  }
});
```

.ajaxSend()

Register a handler to be called when AJAX requests are begun.

```
    .ajaxSend(handler)
```

Parameters

- `handler`: The function to be invoked

Return value

The jQuery object, for chaining purposes.

Description

Whenever an AJAX request is about to be sent, jQuery triggers the `ajaxSend` event. Any and all handlers that have been registered with the `.ajaxSend()` method are executed at this time.

To observe this method in action, we can set up a basic AJAX load request as follows:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element.

```
$('.log').ajaxSend(function() {
  $(this).text('Triggered ajaxSend handler.');
});
```

Now, we can make an AJAX request using any jQuery method.

```
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

When the user clicks the button and the AJAX request is about to begin, the log message is displayed.

> As `.ajaxSend()` is implemented as a method rather than a global function, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxSend` handlers are invoked, regardless of what AJAX request is to be sent. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxSend` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. We can restrict our callback to only handling events dealing with a particular URL, for example:

```
$('.log').ajaxSend(function(e, xhr, settings) {
  if (settings.url == 'ajax/test.html') {
    $(this).text('Triggered ajaxSend handler.');
  }
});
```

.ajaxStart()

Register a handler to be called when the first AJAX request begins.

```
.ajaxStart(handler)
```

Parameters

- `handler`: The function to be invoked

Return value

The jQuery object, for chaining purposes.

Description

Whenever an AJAX request is about to be sent, jQuery checks whether there are any other outstanding AJAX requests. If none are in progress, jQuery triggers the `ajaxStart` event. Any and all handlers that have been registered with the `.ajaxStart()` method are executed at this time.

To observe this method in action, we can set up a basic AJAX load request as follows:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element.

```
$('.log').ajaxStart(function() {
  $(this).text('Triggered ajaxStart handler.');
});
```

Now, we can make an AJAX request using any jQuery method.

```
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

The log message is displayed when the user clicks the button and the AJAX request is sent.

> As `.ajaxStart()` is implemented as a method rather than a global function, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

.ajaxStop()

Register a handler to be called when all AJAX requests have completed.

```
.ajaxStop(handler)
```

Parameters

- `handler`: The function to be invoked

Return value

The jQuery object, for chaining purposes.

Description

Whenever an AJAX request completes, jQuery checks whether there are any other outstanding AJAX requests. If none remain, jQuery triggers the `ajaxStop` event. Any and all handlers that have been registered with the `.ajaxStop()` method are executed at this time.

To observe this method in action, we can set up a basic AJAX load request as follows:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element.

```
$('.log').ajaxStop(function() {
  $(this).text('Triggered ajaxStop handler.');
});
```

Now, we can make an AJAX request using any jQuery method.

```
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

The log message is displayed when the user clicks the button and the AJAX request completes.

As `.ajaxStop()` is implemented as a method rather than a global function, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

.ajaxSuccess()

Register a handler to be called when AJAX requests complete and are successful.

```
.ajaxSuccess(handler)
```

Parameters

- `handler`: The function to be invoked

Return value

The jQuery object, for chaining purposes.

Description

Whenever an AJAX request completes successfully, jQuery triggers the `ajaxSuccess` event. Any and all handlers that have been registered with the `.ajaxSuccess()` method are executed at this time.

To observe this method in action, we can set up a basic AJAX load request as follows:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element.

```
$('.log').ajaxSuccess(function() {
  $(this).text('Triggered ajaxSuccess handler.');
});
```

Now, we can make an AJAX request using any jQuery method.

```
$('.trigger').click(function() {
  $('.result').load('ajax/test.html');
});
```

The log message is displayed when the user clicks the button and the AJAX request completes.

As `.ajaxSuccess()` is implemented as a method rather than a global function, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxSuccess` handlers are invoked, regardless of what AJAX request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxSuccess` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. We can restrict our callback to only handling events dealing with a particular URL, for example:

```
$('.log').ajaxSuccess(function(e, xhr, settings) {
  if (settings.url == 'ajax/test.html') {
    $(this).text('Triggered ajaxSuccess handler.');
  }
});
```

Helper functions

These functions assist with common idioms encountered when performing AJAX tasks.

.serialize()

Encode a set of form elements as a string for submission.

```
.serialize()
```

Parameters

None

Return value

A string containing the serialized representation of the elements.

Description

The `.serialize()` method creates a text string in a standard URL-encoded notation. It operates on a jQuery object representing a set of form elements. The form elements can be of several types.

```
<form>
  <div><input type="text" name="a" value="1" id="a" /></div>
  <div><input type="text" name="b" value="2" id="b" /></div>
  <div><input type="hidden" name="c" value="3" id="c" /></div>
  <div>
    <textarea name="d" rows="8" cols="40">4</textarea>
  </div>
  <div><select name="e">
    <option value="5" selected="selected">5</option>
    <option value="6">6</option>
    <option value="7">7</option>
  </select></div>
  <div>
    <input type="checkbox" name="f" value="8" id="f" />
  </div>
  <div>
    <input type="submit" name="g" value="Submit" id="g" />
  </div>
</form>
```

The `.serialize()` method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization.

```
$('form').submit(function() {
  $.print($(this).serialize());
  return false;
});
```

This produces a standard-looking query string as follows:

```
a=1&b=2&c=3&d=4&e=5
```

.serializeArray()

Encode a set of form elements as an array of names and values.

```
.serializeArray()
```

Parameters

None

Return value

An array of objects containing the serialized representation of each element.

Description

The `.serializeArray()` method creates a JavaScript array of objects, ready to be encoded as a JSON string. It operates on a jQuery object representing a set of form elements. The form elements can be of several types.

```
<form>
  <div><input type="text" name="a" value="1" id="a" /></div>
  <div><input type="text" name="b" value="2" id="b" /></div>
  <div><input type="hidden" name="c" value="3" id="c" /></div>
  <div>
    <textarea name="d" rows="8" cols="40">4</textarea>
  </div>
  <div><select name="e">
    <option value="5" selected="selected">5</option>
    <option value="6">6</option>
    <option value="7">7</option>
  </select></div>
  <div>
```

```
      <input type="checkbox" name="f" value="8" id="f" />
    </div>
    <div>
      <input type="submit" name="g" value="Submit" id="g" />
    </div>
  </form>
```

The `.serializeArray()` method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization.

```
$('form').submit(function() {
  $.print($(this).serializeArray());
  return false;
});
```

This produces the following data structure:

```
[
  {
    name: a
    value: 1
  },
  {
    name: b
    value: 2
  },
  {
    name: c
    value: 3
  },
  {
    name: d
    value: 4
  },
  {
    name: e
    value: 5
  }
]
```