jQuery & AJAX part 2

IMY 220 ● Lecture 14

jQuery.ajax()

Notes:

Go look at all the components of the jQuery.ajax() function that are discussed in this lecture in the API (http://api.jquery.com/jquery.ajax/)

Play around with the examples to see how it works.

The jQuery Ajax Workhorse

All of jQuery’s Ajax functions (that were covered in the previous lecture) uses the $.ajax method.

All of jQuery’s Ajax functions are simply wrappers around the $.ajax method, each designed to provide you with a simple interface for a specific type of task.

The $.ajax method is the heart of jQuery’s Ajax abilities and is the most powerful and customizable Ajax method available to us.

$.ajax method syntax

The syntax used with the $.ajax method is more complex than the others we’ve encountered so far, because you need to spell out everything you want it to do: it will take nothing for granted.

It’s still fairly straightforward to use.

$.ajax method syntax

Simple GET request:

We specify

the request type,

the URL to hit, and

a success callback wherein we can manipulate the data.

```
$.ajax({
      type: 'GET',
      url: 'getDetails.php',
      data: {id: 142}
})
.done(data => {
      // do something with data
});
```

Options

The complexity emerges from the number of possible options you can provide.

There are over 20 available options ranging from error callbacks, usernames and passwords for authentication requests, and data filter functions for pre-processing returned information.

The majority of the time you’ll only need some of these options, as most common functionality uses just a few.

Common Ajax Settings

Often, you'll want to apply several of the same settings to multiple Ajax calls in the same script.

Typing them out at length each time would grow tedious, so jQuery allows you to specify global settings with the $.ajaxSetup action:

```
$.ajaxSetup({
    type: 'POST',
    url: 'send.php',
    timeout: 3000
});
```

$.ajaxSetup

This sets the defaults for further Ajax calls, though you can still override them if you need to.

By specifying the common defaults for the page, we can dramatically simplify the code required to actually send a request:

```
$.ajax({
    data: {id: 142}
});
```

> This call will send a POST request to send.php, which will time out if it takes longer than 3,000 milliseconds.

$.ajaxSetup

Some settings are unable to be set via the $.ajaxSetup method;

for example

- error,
- complete, and
- success callback functions,

should be handled with the global Ajax events

jQuery Ajax Events

When making Ajax requests with jQuery, a number of events are fired.

These can be handled anytime we’d like to do some extra processing

For example, adding a “loading” message when a request begins and removing it when it concludes, or handling any errors that may have occurred.

jQuery Ajax Events

There are two types of Ajax events in jQuery: local and global.

Local events apply only to individual Ajax requests.

You handle local events as callbacks, just as you do with any of the other events we’ve seen.

Global events are broadcast to any handlers that are listening, so they present a great way to implement functionality that should apply to all requests.

jQuery Ajax Events

As an example, jQuery defines a fail local event and an ajaxError global event.

- The fail event can be handled for an individual request,

- The ajaxError can handle an error that occurs in any request.

fail

Local events are handled inside the $.ajax call, like this:

```
$.ajax({
     url: "test.html"
})
.fail(() => {
     alert('an error occurred!');
});
```

ajaxError

Global events, on the other hand, are generally attached to the DOM node where you’d like to show a response to the user.

For instance, you may have a div element for displaying your Ajax error messages:

```
$(document).ajaxError(() => {
     $(".alert").html("The ajax request failed");
});
```

ajaxError

The callback function can take up to four parameters, for example:

```
$(document).ajaxError((event, request, settings, thrownError) => {
        $(".alert").html("Error requesting page " + settings.url);
});
```

The settings object contains all the values from the original ajax request.

The thrown error is a string containing the type of error encountered

ajaxError

Whenever any Ajax call from the page fires an error, this handler will be called.

We’re interested in more than just errors, though: there are a bunch of interesting events we can hook into if need be.

They’re all structured as in the aforementioned script, and most have a corresponding local and global version, so you can choose which granularity to handle them with.

jQuery Ajax Events

The done event (local) or the ajaxSuccess event (global)

let us know when an event has completed successfully.

The always (local) and ajaxComplete (global) events

tell us when an Ajax request has concluded, regardless of its success or failure.

jQuery Ajax Events

The beforeSend (local) and ajaxSend (global) events

  let us react just before an Ajax request is sent into the world.

And finally, the ajaxStart and ajaxStop global events

  occur when an Ajax request fires and no others are already running, and when all requests are finished, respectively.

Simple Example

Here is a simple example of the $.ajax function in action with a PHP file jQuery script:

```
$.ajax({
      url: 'greet.php',
      type: 'POST',
      data: {name: 'Rick', surname: 'Sanchez'}
})
.done(data => {
      // data contains whatever is echoed
      // by the PHP file
      // Result: "Hello there, Rick Sanchez"
      alert(data);
});
```

Simple Example

greet.php:

```php
<?php
     $name = $_POST['name'];
     $surname = $_POST['surname'];

     echo "Hello there, " . $name . " " . $surname;
?>
```

jQuery Ajax Events

We can also encapsulate our Ajax call as a Promise

However, the `$.ajax` function has an event handler for `.then`, which we can use to create orderly chains of Promises

The next example will use the result of one Ajax call to do another Ajax call

Sequential Ajax calls

Let's say we have a page called userDetails.php…

…which contains a list of users, each of which has some user details…

```
$users = [
        [
                "userid" => "16463",
                "name" => "Rick",
                "surname" => "Sanchez",
                "email" => "rick@gmail.com"
        ],
        [
                "userid" => "82755",
                "name" => "Morty",
                "surname" => "Smith",
                "email" => "morty456@hotmail.com"
        ]
];
```

Sequential Ajax calls

And we have a page called products.php…

…which contains a list of products, each of which has an ownerid which corresponds to a userid

```
$products = [
        [
                "name" => "Vroomba",
                "ownerid" => "87265"
        ],
        [
                "name" => "Used potato",
                "ownerid" => "17930"
        ],
        [
                "name" => "Teleport gun",
                "ownerid" => "16463"
        ],
        [
                "name" => "Single flip flop",
                "ownerid" => "16463"
        ],
];
```

Sequential Ajax calls

We want to a user to be able to view their products by entering their email address

To do this, we must first fetch the userid that corresponds to the email address…

…and then use that to fetch the product that correspond that the userid
(In this example we only fetch one product per user)

Sequential Ajax calls

First, we have to send the email to userDetails.php

```
let userDetails = {email: $("input#email").val()};

$(".btn").on('click', event => {
      event.preventDefault();
      $.ajax({
            url: 'userDetails.php',
            type: 'POST',
            data: userDetails
      })
      // We’re not doing anything with the data we get back
      // from userDetails yet
      // We’re still going to add that
});
```

Sequential Ajax calls

Inside userDetails.php, we simply return the userid which corresponds to the email address

(or a message if the user does not exist)

```
$email = $_POST['email'];

foreach($users as $user){
	if($user["email"] == $email){
		die($user["userid"]);
	}
}

echo "That user does not exist";
```

Sequential Ajax calls

We can call .then on the original Ajax call and make another Ajax call to get the products using the userid

Since we want to do something with the results of the second Ajax call, we return the second Ajax call as a Promise, which allows us to call .then on that function as well

Sequential Ajax calls

We can call .then on the original Ajax call and make another Ajax call to get the products using the userid

```
$.ajax({
      // details of first Ajax call
})
.then(userid => {
      return $.ajax({
            url: 'products.php',
            type: 'POST',
            data: {userid}
      })
})
.then(data => {
      console.log(data);
});
```

Note the object literal enhancement

Do something with the product name

Sequential Ajax calls

products.php

```php
$userid = $_POST['userid'];

foreach($products as $product){
	if($product["ownerid"] == $userid){
		die($product["name"]);
	}
}

echo "No products for that user";
```

```
let userDetails = {email: $("input#email").val()};

$(".btn").on('click', event => {
     event.preventDefault();
     $.ajax({
          url: 'userDetails.php',
          type: 'POST',
          data: userDetails
     })
     .then(userid => {
          return $.ajax({
               url: 'products.php',
               type: 'POST',
               data: {userid}
          })
     })
     .then(data => {
          console.log(data);
     });
});
```

```php
// userDetails.php
$users = [
       [
              "userid" => "16463",
              "name" => "Rick",
              "surname" => "Sanchez",
              "email" => "rick@gmail.com"
       ],
       [
              "userid" => "82755",
              "name" => "Morty",
              "surname" => "Smith",
              "email" => "morty456@hotmail.com"
       ]
];

$email = $_POST['email'];

foreach($users as $user){
       if($user["email"] == $email){
              die($user["userid"]);
       }
}

echo "That user does not exist";
```

```php
// products.php
$products = [
        [
                "name" => "Vroomba",
                "ownerid" => "87265"
        ],
        [
                "name" => "Used potato",
                "ownerid" => "17930"
        ],
        [
                "name" => "Teleport gun",
                "ownerid" => "16463"
        ],
        [
                "name" => "Single flip flop",
                "ownerid" => "16463"
        ],
];

$userid = $_POST['userid'];

foreach($products as $product){
        if($product["ownerid"] == $userid){
                die($product["name"]);
        }
}

echo "No products for that user";
```

References

[jQuery: Novice to Ninja by Earle Castledine and Craig Sharkie]

[https://api.jquery.com/category/ajax/]