Web Programming

Lecture 22 – AJAX 2

Shamsa Abid

Displaying and Updating Records

- A mini-application for displaying and updating employee records.
- AjaxBasics/Solutions/EmployeeAdmin.html
- Open http://localhost:8080/EmployeeAdmin.html in your browser
- be sure to start the Node.js server (by typing npm start from the command line) in the AjaxBasics/Solutions/ directory first.

Displaying and Updating Records

- Explanation
  - Function getEmployeeList, invoked when the page loads, makes an Ajax call to /EmployeeList and invokes function display. If the Ajax calls is successful, then function display displays the returned list of employees in div#EmployeeList.
  - Also in function display we iterate over each of the bulleted names, listening for clicks. A click on any of the names results in a call to function getEmployeeForm, passing along the id of the clicked-upon employee.
  - Function getEmployeeForm, in turn, makes an Ajax request, via the POST method, to /EmployeeForm, displays the returned form, and adds change handlers on each of the fields of the form.
  - As a result of the handlers added in function getEmployeeForm, a change to any of the employee-form fields invokes function updateEmployee which, in turn, makes a POST Ajax request to /EditEmployee, sending along the id of the employee, field name, and field value. The function displays a success or failure message based on the status of the returned results.

Displaying and Updating Records

- Challenge
  - In the employeeUpdated() callback function, we currently call getEmployeeList() to update the list of employees. This updates the entire list after each change. It would be better to just update the record that was changed. There is no need to make a call to the database to find out which record it was or how it was changed. The client already has that information.
- AjaxBasics/Solutions/EmployeeAdmin-challenge.html
- Open http://localhost:8080/EmployeeAdmin-challenge.html in your browser
  - In function getEmployeeForm, we now keep track of the first and last name (from the fields with ids FirstName and LastName, respectively) of each employee. We pass these values as parameters in the call to updateEmployee when any of the fields is changed.
  - Function updateEmployee works largely as before but, upon successful update of the employee (i.e. a successful Ajax response code), the function now sets the displayed name of the relevant employee in the bulleted list.

How to use jQuery's Ajax methods

- $('div.news').hide()
- all tags of class "news" - and hide them.
- jQuery/Demos/jquerysimpleexample.html
- Explanation
  - We use the jQuery ready method to run our code as soon as the DOM loads.
  - We add a click handler for the button so that any time the button is clicked, the following occurs:
    - Count the number of list items on the page (the length of the query $('li'), which returns all of the s on the page) and assign this value to the variable numListItems.
    - Set the contents of the initially-empty with id Content to display the length.

Ajax with jQuery

- jQuery provides Ajax support that abstracts away painful browser differences.
- It offers both a full-featured $.ajax() method, and simple convenience methods such as $.get(), $.getScript(), $.getJSON(), $.post(), and $.fn.load().
- Most jQuery applications don't in fact use XML; instead, they transport data as plain HTML or JSON (JavaScript Object Notation).

Data Types

- jQuery generally requires some instruction as to the type of data you expect to get back from an Ajax request; in some cases the data type is specified by the method name, and in other cases it is provided as part of a configuration object.
- There are several options:
  - text - For transporting simple strings, usually to be placed directly into the page
  - html - For transporting blocks of HTML, usually to be placed directly into the page
  - xml - For transporting blocks of XML, which can be parsed to provide data
  - script - For adding a new script to the page -- this type of request uses a concept called *dynamic script tagging* instead of XMLHttpRequest, and is therefore not limited by the same origin policy
  - json - For transporting JSON-formatted data, which can include JavaScript strings, arrays, and objects
  - jsonp - A variant of JSON, which can be used to retrieve content from other servers -- this type of request also uses dynamic script tagging

JSON and XML

JSON:

```
person = {"age" : 33, "name" : "Joshua"}
```

XML:

```
<person>
        <age>33</age>
        <name>Joshua</name>
</person>
```

jQuery's Ajax-Related Methods

- $.ajax
- jQuery's core $.ajax method is a powerful and straightforward way of creating Ajax requests.
- It takes a configuration object that contains all the instructions jQuery requires to complete the request.
- The ability to specify both success and failure callbacks.
- Also, its ability to take a configuration object that can be defined separately makes it easier to write reusable code.
- $.ajax(options) or $.ajax(url, options)
  - Note that the url can either be provided as a separate parameter or as part of the options object.

Options for $.ajax

- **url** - The URL for the request. Required either as an option or as a separate parameter.
- **type** - The type of the request, "POST" or "GET". Defaults to "GET". Other request types, such as "PUT" and "DELETE" can be used, but they may not be supported by all browsers.
- **async** - Set to false if the request should be sent synchronously. Defaults to true. Note that if you set this option to false, your request will block execution of other code until the response is received.
- **cache** - Whether to use a cached response if available. Defaults to true for all data types except script and jsonp. When set to false, the URL will simply have a cache busting parameter appended to it.
- **success** - A **callback** function to run if the request succeeds. The function receives the response data (converted to a JavaScript object if the data type was JSON), as well as the text status of the request and the raw request object.
- **error** - A **callback** function to run if the request results in an error. The function receives the raw request object and the text status of the request.
- **complete** - A **callback** function to run when the request is complete, regardless of success or failure. The function receives the raw request object and the text status of the request. This function will run after any error or success function, if those are also specified.

Options for $.ajax

- **context** - The scope in which the callback function(s) should run (i.e. what this will mean inside the callback function(s)). By default, this inside the callback function(s) refers to the object originally passed to $.ajax.
- **data** - The data to be sent to the server. This can either be an object, like { foo:'bar',baz:'bim' } , or a query string, such as foo=bar&baz=bim.
- **dataType** - The type of data you expect back from the server. By default, jQuery will look at the *MIME-type* (from the Content-Type header) of the response if no data type is specified.
- **jsonp** - The callback parameter name to send in a query string when making a JSONP request. Defaults to callback. (jQuery will add a parameter *callback=XXXXXX* to the query string; this option sets the parameter name, and the following option sets the parameter value).
- **jsonpCallback** - The value of the callback parameter to send in a query string when making a JSONP request. Defaults to an auto-generated random value.
- **timeout** - The time in milliseconds to wait before considering the request a failure.

Using the Core $.ajax Method

```
$.ajax({
                // the URL for the request
                url : 'post.php',

                // the data to send
                // (will be converted to a query string)
                data : { id : 123 },

                // whether this is a POST or GET request
                type : 'GET',

                // the type of data we expect back
                dataType : 'json',

                // code to run if the request succeeds;
                // the response is passed to the function
                success : function(json) {
                        $('<h1/>').text(json.title).appendTo('body');
                        $('<div class="content"/>')
                                .html(json.html).appendTo('body');
                },
```

Using the Core $.ajax Method

```
        // code to run if the request fails;
        // the raw request and status codes are
        // passed to the function
        error : function(xhr, status) {
                alert('Sorry, there was a problem!');
        },

        // code to run regardless of success or failure
        complete : function(xhr, status) {
                alert('The request is complete!');
        }
});
```

Example using xml

- Navigate on the command line to jQuery/Demos.
- Type npm install from the command line to install the needed Node.js modules.
- Type npm start to start the Node.js server.
- Open http://localhost:8080/UsingXMLHttpRequest-Ajax.html in a browser.
- Open jQuery/Demos/UsingXMLHttpRequest-Ajax.html in a code editor to review the code.
- Continued…

```
<script type="text/javascript">
        $(document).ready(function() {
                $('#btn').click(function() {
                        $.ajax({
                                url: "Demo",
                                data: {
                                        FirstName: "Nat",
                                        LastName: "Dunn"
                                },
                                dataType: "xml"
                        }).done(function(xml) {
                                var h1 = $(xml).find("h1");
                                $("#Content").html(h1);
                        });
                        return false;
                });
        });
</script>
</head>
<body>
        <button id="btn">Start</button>
        <div id="Content"></div>
</body>
</html>
```

Explanation

- We use jQuery's ready method to add a listener on the button; when clicked, we invoke the jQuery ajax method.
- We use three parameters in our call to $.ajax
  - **url**: "Demo" specifies that the Ajax request is sent to /Demo, a response route we've setup in our Node.js server; since we don't state otherwise, $.ajax uses the default GET method.
  - **data**: { FirstName: "Nat", LastName: "Dunn" } sets the parameters included with the request; in this case, we are requesting the URL http://localhost:8080/Demo?FirstName=Nat&LastName=Dunn; that is, we include two GET parameters, FirstName and LastName.
  - The **dataType**: "xml" parameter tells jQuery that we are expecting XML as the mime type from the response.
- The done callback allows us to handle the response, if any, that we get back from the remote resource.
- As /Demo responds with a single <h1> tag (<h1>Hello Nat Dunn</h1>), we use jQuery's find method to get the contents of the <h1> tag - that is, "Hello Nat Dunn" - and assign the value to a local variable h1.
- Lastly, we set the with id Content to that value, using jQuery's html method.

Example using JSON

- jQuery/Demos/UsingXMLHttpRequest-Ajax-JSON.html
- Open http://localhost:8080/UsingXMLHttpRequest-Ajax-JSON.html in your browser to view the demo.
- We again use jQuery's ready method to add a listener on the button; when clicked, we invoke the jQuery ajax method.
- The key difference between this example and the last one is that this ajax call is expecting to receive a JSON response. If we examine the Node.js response route /DemoJSON?FirstName=Nat&LastName=Dunn, we see that the returned data is

[{"greeting":"hello","name":"Nat Dunn"},
{"greeting":"hola","name":"Nat Dunn"},
{"greeting":"bon jour","name":"Nat Dunn"},
{"greeting":"hallo","name":"Nat Dunn"}]

Convenience Methods

- If you don't need the extensive configurability of $.ajax, and you don't care about handling errors, the Ajax convenience functions provided by jQuery can be useful
- These methods are just "wrappers" around the core $.ajax method, and simply pre-set some of the options on the $.ajax method.
- The convenience methods provided by jQuery are:
  - $.get - Perform a GET request to the provided URL.
  - $.post - Perform a POST request to the provided URL.
  - $.getScript - Add a script to the page.
  - $.getJSON - Perform a GET request, and expect JSON to be returned.

Convenience methods arguments

- The methods take the following arguments, in order:
- **url** - The URL for the request. Required.
- **data** - The data to be sent to the server. Optional. This can either be an object or a query string, such as foo=bar&baz=bim.
- **success** - A callback function to run if the request succeeds. Optional. The function receives the response data (converted to a JavaScript object if the data type was JSON), as well as the text status of the request and the raw request object.
- **dataType** - The type of data you expect back from the server. Optional.

Using jQuery's Ajax Convenience Methods

```
// get plain text or html
        $.get('/users.php', { userId : 1234 }, function(resp) {
                console.log(resp);
        });

        // add a script to the page, then run a function defined in it
        $.getScript('/js/myScript.js', function() {
                functionFromMyScript();
        });

        // get JSON-formatted data from the server
        // resp will be a single object parsed from the incoming JSON
        $.getJSON('/details.php', function(resp) {
                $.each(resp, function(k, v) {
                        console.log(k + ' : ' + v);
                });
        });
```

Using jQuery's Ajax Convenience Methods

```
<script type="text/javascript">
        $(document).ready(function() {
                $('#btn').click(function() {
                        $.get("Demo", {
                                        FirstName: "Some",
                                        LastName: "Othername"
                                },
                                function(data) {
                                        var h1 = $(data).find("h1");
                                        $("#Content").html(h1);
                                },
                                "xml"
                        );
                        return false;
                });
        });
</script>
```

$.fn.load

- **$.fn.load** method fetches HTML from a URL, and uses the returned HTML to populate the selected element(s).
- In addition to providing a URL to the method, you can optionally provide a selector as part of the URL parameter
- jQuery will fetch only the matching content from the returned HTML.

$.fn.load

- Using $.fn.load to Populate an Element
  - $('#newContent').load('/foo.html');
- Using $.fn.load to Populate an Element Based on a Selector
  - $('#newContent').load('/foo.html #myDiv h1:first', function(html) { alert('Content updated!'); });

Project Submission

- Use AJAX in your projects to optimize performance by reducing the number of page reloads/redirects
- Refer to the online tutorial https://www.webucator.com/tutorial/learn-ajax/ajax-applications.cfm to get ideas of application of AJAX

References

- https://www.webucator.com/tutorial/learn-ajax/intro-ajax-the-nodejs-server.cfm

- Code Download Link: http://www.webucator.com/class-files/index.cfm?CourseID=JSC401