

COMP519 Web Programming

Lecture 26: Ajax

Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

- 1 Ajax
 - Motivation
 - Overview
 - Creating a HTTP Request
 - Creating a HTTP Response
 - Processing a HTTP Response
- 2 Further Reading

Ajax: Motivation

Postcode:

Address:

```
<form id='pc'><label>Postcode:</label>
<input type='text' name='pc' value=''>
<br><label>Address:</label>
<select name='adr' id='adr'></select>
</form>
```

- Entering an address into an HTML form is often a two-stage process
 - ① Entering a postcode into a text field
 - ② Selecting an entry from a drop-down menu of all addresses for the entered postcode
- To implement this process using PHP
 - Step ① needs to result in a form submission to a PHP script
 - The PHP script needs to retrieve the addresses for the given postcode from a database
 - The PHP script then produces a complete HTML document with an updated form containing options for the drop-down menu
 - That HTML document is sent back
 - The user can then perform Step ②

Ajax: Motivation

Postcode:

Address:

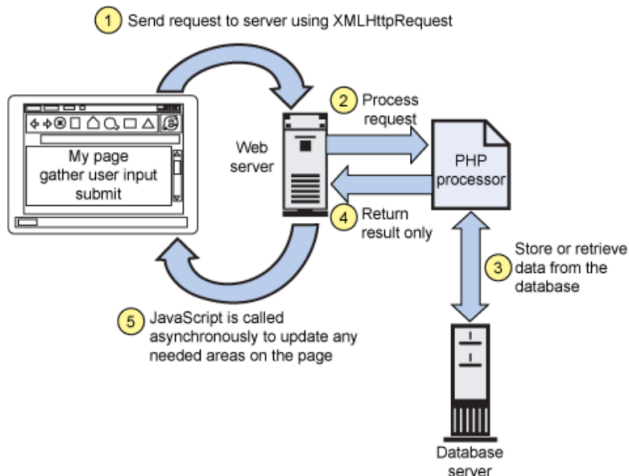
```
<form id='pc'><label>Postcode:</label>
<input type='text' name='pc' value=''>
<br><label>Address:</label>
<select name='adr' id='adr'></select>
</form>
```

- Entering an address into an HTML form is often a two-stage process
 - ① Entering a postcode into a text field
 - ② Selecting an entry from a drop-down menu of all addresses for the entered postcode
- To implement this process using JavaScript
 - A JavaScript event handler is triggered when the user has completed Step ①
 - The JavaScript event handler uses an HTTP request to get the data for the drop-down menu
 - In response the HTTP request, a PHP script retrieves the addresses for the given postcode from a database and sends them back
 - The JavaScript event handler constructs the drop-down menu
 - The user can then perform Step ②

Ajax: Overview

- Ajax, Asynchronous JavaScript and XML, is a set of JavaScript methods related to XMLHttpRequest objects and patterns for their use
- Ajax allows web applications to send to and retrieve data from a server asynchronously (in the background)
- On the server-side, PHP scripts are typically used to receive / send data and to deal with related database transactions
- Historically, data was transferred in XML format though nowadays use of JSON is much more common

Ajax: Overview



K. Ramirez: Build Ajax-based Web sites with PHP. IBM, 2 Sep 2008.
<https://www.ibm.com/developerworks/library/wa-aj-php/>
[accessed 03 Dec 2020]

XMLHttpRequest Objects: Properties

status

HTTP status code returned by server
(200, 403, 404, 500, ...)

statusText

HTTP reason phrase returned by server
("OK", "Forbidden", "Page not found", "Internal Server Error")

responseText

Data returned by the server as a string

responseXML

Data returned by the server as XMLDocument object

readyState

Integer value reporting the status of the request
(0: uninitialised, 1: loading, 2: loaded, 3: interactive, 4: completed)

onreadystatechange

function to be called when the readyState property changes

XMLHttpRequest Objects: Methods

```
open(method, url, async)
```

Prepares an HTTP request by specifying

- the HTTP method *method* (as string),
- the target URL *url* (as string), and
- a boolean value *async* indicating whether the request should be handled asynchronously

```
open('POST', 'getdata.php', TRUE)
```

```
setRequestHeader(param, value)
```

Sets a header *param* to *value* and adds it to the headers of the HTTP request

```
setRequestHeader('Content-type',  
                  'application/x-www-form-urlencoded')
```

```
send(content)
```

Send the HTTP request, optionally with POST data *content*

XMLHttpRequest Objects: Methods

`abort()`

Stop the current request

`getAllResponseHeaders()`

Returns all header/value pairs in the HTTP response as a string

`getResponseHeader(param)`

Returns the value for header *param* in the HTTP response as a string, or null if there is no value for *param*

`getResponseHeader('Content-type')`

XMLHttpRequest Objects: Example

```
function createSelectOptionsForPostcode(formData) {  
    var request = new XMLHttpRequest()  
    request.open('POST','getData.php',true)  
    request.setRequestHeader('Content-type',  
                             'application/x-www-form-urlencoded')  
    request.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            if (this.getResponseHeader('Content-type').indexOf(''  
↪xml') >= 0) {  
                // process XML data sent by getData.php  
                processXMLResponse(this.responseXML)  
            } else if (this.getResponseHeader('Content-type').  
↪indexOf('json') >= 0) {  
                // process JSON data sent by getData.php  
                processJSONResponse(this.responseText)  
            } } }  
    request.send(formData)  
}  
  
// createSelectOptionsForPostcode('postCode=L69+9AD')
```

JavaScript and Forms Revisited

Postcode:

Address:

```
<form id='pc'><label>Postcode:</label>
<input type='text' name='pc' value=''>
<br><label>Address:</label>
<select name='adr' id='adr'></select>
</form>
```

- The function `createSelectOptionsForPostcode` requires form data as argument, in particular, the user input from the Postcode field
- We can use JavaScript's `FormData` object to deal with that

`FormData([form])`

Creates a new `FormData` object

When the optional *form* argument for a HTML form is specified, the `FormData` object will be populated with name/value pairs from the form with id *form*

`append(name, value)`

Adds or updates the name/value pair for *name*

`delete(name)`

Deletes the name/value pair for *name*

JavaScript and Forms Revisited

Postcode:

Address:

```
<form id='pc'><label>Postcode:</label>
<input type='text' name='pc' value=''>
<br><label>Address:</label>
<select name='adr' id='adr'></select>
</form>
```

- The function `createSelectOptionsForPostcode` requires form data as argument, in particular, the user input from the Postcode field
- We can use JavaScript's `FormData` object to deal with that

```
var formElem = document.getElementById('pc')
var inptElem = document.querySelector('input')
// First argument of addEventListener() refers to input event
inptElem.addEventListener('input', function (e) {
  // Construct FormData object
  var data = new FormData(formElem)
  // Ideally we would validate data before sending
  // the request
  createSelectOptionsForPostcode(data)
})
```

Retrieving the Data

```
<?php
// getData.php
// We have a database with a table
//     postcodes(pc VARCHAR(8), address VARCHAR(300))
// We assume that PDO arguments have already been defined
if (isset($_POST['postCode'])) {
    try {
        $pdo      = new PDO($dsn,$db_username,$db_password,$opt);
        $tpl      = "select address from postcodes where pc = ?";
        $stm      = $pdo->prepare($tpl);
        $success  = $stm->execute(array($_POST['postCode']));
        $data     = $stm->fetchAll();
        // Construct a HTTP response using a function
        // outputData that we have written
        outputData($data);
    } catch (PDOException $e) {
        outputData(array(0 => array('error' => $e)));
    } }
?>
```

Constructing the HTTP Response

Our database query has returned a two-dimensional array, e.g.,

```
array(  
  0 => array('address' => '1 Rose Lane, Liverpool'),  
  1 => array('address' => 'Lennox, 2 Rose Lane, Liverpool'),  
  2 => array('address' => 'Flat A, 3 Rose Lane, Liverpool'),  
  ...  
)
```

In the HTTP response we can represent this array

- in a user-defined format
- in XML (eXtensible Markup Language) format
(a 'heavyweight' data-interchange format built on markup with user-defined tags)

```
<?xml version="1.0"?>  
<data><item0><address>1 Rose Lane, Liverpool</address></item0>  
      <item1><address>Lennox, 2 Rose Lane, Liverpool</address></item1>  
      <item2><address>Flat A, 3 Rose Lane, Liverpool</address></item2>  
</data>
```

- in JSON format

Constructing the HTTP Response

Our database query has returned a two-dimensional array, e.g.,

```
array(  
  0 => array('address' => '1 Rose Lane, Liverpool'),  
  1 => array('address' => 'Lennox, 2 Rose Lane, Liverpool'),  
  2 => array('address' => 'Flat A, 3 Rose Lane, Liverpool'),  
  ...  
)
```

In the HTTP response we can represent this array

- in a user-defined format
- in XML format
- in JSON format

(a lightweight data-interchange format built on

- (i) primitive values (numbers, strings), (ii) collections of name/value pairs, (iii) ordered lists of values)

```
[{"address": "1 Rose Lane, Liverpool"},  
 {"address": "Lennox, 2 Rose Lane, Liverpool"},  
 {"address": "Flat A, 3 Rose Lane, Liverpool"}]
```

Constructing the HTTP Response: XML

In PHP, we can use the `SimpleXMLElement` class to construct XML elements

`__construct(string xml)`

xml is a well-formed XML string

`new SimpleXMLElement('<?xml version="1.0"?><data></data>')`

`addChild(string elem [, string str])`

Adds an element *elem* with content *str*, if specified

`addChild('name', 'Peter')` # `<name>Peter</name>`

`addAttribute(string attr [, string val])`

Adds an attribute *attr* with value *val* to an element

`asXML([string fn])`

If a filename *fn* is not specified, returns a string representation of an element in XML 1.0 format, otherwise, writes that string to *fn*

`children()`

Finds the children of an element

Constructing the HTTP Response: XML

```
function outputData($data) {  
    $xml = new SimpleXMLElement(  
        '<?xml version="1.0"?><data></data>');  
    arrayToXML($data,$xml);  
    header('Content-type: application/xml; charset=utf-8');  
    echo($xml->asXML());  
}
```

Constructing the HTTP Response: XML

```
array(  
    0 => array('address' => '1_Rose_Lane, _Lvpl'),  
    ...  
)  
  
<?xml version="1.0"?>  
  <data><item0><address>1 Rose Lane, Lvpl</address></item0>  
  ...  
</data>  
  
function arrayToXML($data, &$xml) {  
    foreach ($data as $key => $value) {  
        if (is_numeric($key)) {  
            $key = 'item' . $key; //dealing with <0/>..<n/> issues  
        }  
        if (is_array($value)) {  
            $subnode = $xml->addChild($key);  
            arrayToXML($value, $subnode);  
        } else {  
            $xml->addChild("$key", htmlspecialchars("$value"));  
        }  
    }  
}
```

<https://stackoverflow.com/a/5965940>

Constructing the HTTP Response: JSON

```
json_encode(mixed value [, int options [, int dep]])
```

Returns the JSON representation of *value* up to nesting depth *dep* with encoding modified by *options*

Options include

JSON_UNESCAPED_UNICODE: Preserve UTF-8 characters

JSON_UNESCAPED_SLASHES: Do not escape slashes

```
json_encode(['name'=>'Peter']) # {"name":"Peter"}
```

```
json_decode(string value [, bool assoc  
[, int dep [, int options]]])
```

Returns a PHP value for a JSON encoded string *value* up to nesting depth *dep* with decoding modified by *options* and if *assoc* is **TRUE** returns arrays instead of objects for key/value pairs

```
json_decode('{"name": "Peter"}')
```

Object with a property "name" that has value "Peter"

```
json_decode('{"name": "Peter"}',TRUE)
```

```
Array ["name" => "Peter"]
```

Constructing the HTTP Response: JSON

```
function outputData($data) {  
    header('Content-Type: application/json; charset=utf-8');  
    echo json_encode($data,  
        JSON_UNESCAPED_UNICODE | JSON_UNESCAPE_SLASHES);  
}
```

The vertical bar | is the operator for bitwise inclusive Or

Processing the HTTP Response: XML

- The XML DOM defines a standard way for accessing and manipulating XML documents / XML document objects
- The XML DOM views an XML document as a tree structure of **nodes**
 - The entire document is a **document node**
 - Every XML element is an **element node**
 - The text in the XML elements are **text nodes**
 - Every attribute is an **attribute node**

Document nodes have an attribute

documentElement

Returns the root element 'below' a document node

Document nodes and **element nodes** have a method

getElementsByTagName(*name*)

Returns a collection of all element nodes in a node tree with the specified tag name *name*

Processing the HTTP Response: XML

- The XML DOM defines a standard way for accessing and manipulating XML documents / XML document objects
- The XML DOM views an XML document as a tree structure of **nodes**

All **nodes** have attributes

nodeName

The name of a node as a string, for element nodes this is the **tag name**, for attribute nodes, the **attribute name**

nodeValue

The **value of a node** as a string, for **text nodes** this is the text content of the node

childNodes

A collection of children of a node, indexed by natural numbers starting from 0

childNodes.length

The number of children

Processing the HTTP Response: XML

```
<data>
  <item0><address>1 Rose Lane, Lvpl</address></item0>
  <item1><address>Lennox, 2 Rose Lane, Lvpl</address></item1>
  <item2><address>Flat A, 3 Rose Lane, Lvpl</address></item2>
</data>
```

```
function processXMLResponse(xml) {
  var o    = document.createElement('option')
  o.label = 'Select an address'
  o.value = ''
  sel = document.getElementById('adr')
  sel.appendChild(o)
  adrs = xml.getElementsByTagName('address')
  for (i = 0; i < adrs.length; i++) {
    o = document.createElement('option')
    o.label = o.text = adrs[i].childNodes[0].nodeValue
    sel.appendChild(o)
  } }
```

```
<option value=''>Select an address</option>
<option value='1 Rose Lane'>1 Rose Lane, Lvpl</option>
<option value='Lennox, 2 Rose Lane'>Lennox, 2 Rose Lane, Lvpl</option>
<option value='Flat A, 3 Rose Lane'>Flat A, 3 Rose Lane, Lvpl</option>
```

Processing the HTTP Response: JSON

- The `JSON` object contains methods for parsing JavaScript Object Notation (JSON) and converting values to JSON
- There is no constructor for new / additional JSON objects

```
JSON.parse(text [, reviver])
```

Parses a JSON string *text* and returns the corresponding JavaScript value transformed further by the optional *reviver* function

```
JSON.stringify(value [, replacer [, space]])
```

Converts a JavaScript value to a JSON string *value*, optionally replacing values if a function *replacer* and adding space as specified by *space*

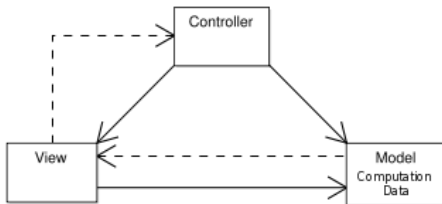
Processing the HTTP Response: JSON

```
// [{"address": "1 Rose Lane, Liverpool"},  
//  {"address": "Lennox, 2 Rose Lane, Liverpool"},  
//  {"address": "Flat A, 3 Rose Lane, Liverpool"}]
```

```
function processJSONResponse(text) {  
    adrs = JSON.parse(text)  
    var o = document.createElement("option")  
    o.label = "Select an address"  
    o.value = ""  
    sel = document.getElementById('adr')  
    sel.appendChild(o)  
    for (i = 0; i < adrs.length; i++) {  
        o = document.createElement("option")  
        o.label = o.text = adrs[i].address  
        sel.appendChild(o)  
    }  
}
```

```
<option value=''>Select an address</option>  
<option value='1 Rose Lane'>1 Rose Lane</option>  
<option value='Lennox, 2 Rose Lane'>Lennox, 2 Rose Lane</option>  
<option value='Flat A, 3 Rose Lane'>Flat A, 3 Rose Lane</option>
```

Ajax and Model-View-Controller



- Without Ajax and without JavaScript,
 - a lot of the **Controller** and
 - all of the **Model**must reside on the **server-side** and is programmed in PHP
- With Ajax and with JavaScript,
 - all of the **Controller** and
 - a lot of the **Model**can reside on the **client-side** and is programmed in JavaScript

Revision and Further Reading

- Read
 - Chapter 18: Using Asynchronous Communication of R. Nixon: Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. O'Reilly, 2018.

Revision and Further Reading

- Read

- Function Reference: XML Manipulation

<https://www.php.net/manual/en/refs.xml.php>

- Function Reference: JavaScript Object Notation

<https://www.php.net/manual/en/book.json.php>

of P. Cowburn (ed.): PHP Manual. The PHP Group, 24 Nov 2019.

<http://uk.php.net/manual/en> [accessed 25 Nov 2019]

- Read

- Mozilla and individual contributors: Document Object Model (DOM).

MDN Web Docs, 17 Nov 2019. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model [accessed 25 Nov 2019]

- Mozilla and individual contributors: JSON.

MDN Web Docs, 4 Nov 2019. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON [accessed 25 Nov 2019]