COMP519 Web Programming
Lecture 26: Ajax
Handouts (8 on 1)

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

Contents

- Ajax
  Motivation
  Overview
  Creating a HTTP Request
  Creating a HTTP Response
  Processing a HTTP Response

- Further Reading

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: Motivation

- Entering an address into an HTML form is often a two-stage process
  - Entering a postcode into a text field
  - 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

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 an XML 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
- send(content)
  Send the HTTP request, optionally with POST data content
**XMLHttpRequest Objects: Methods**

```javascript
// JavaScript and Forms Revisited

function createSelectOptionsForPostcode(formData) {
    var formElem = document.getElementById('postcodeForm');
    var data = new FormData(formElem);
    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.responseText) {
                processXMLResponse(this.responseText);
            } else if (this.responseText) {
                processJSONResponse(this.responseText);
            } else {
                processData(this.responseText);
            }
        }
    }
    request.send(formData);
}
```

**Retrieving the Data**

```php
<?php
// We have a database with a table
// postcode (pc VARCHAR(8), address VARCHAR(300))
// We assume that PDO arguments have already been defined
try {
    $pdo = new PDO($dsn, $db_username, $db_password, $opt);
    $stmt = $pdo->prepare("select address from postcodes where pc = ?");
    $data = $stmt->execute(array($param));
    $success = $stmt->execute(array($param));
}
catch (PDOException $e) {
    processData(array(0 => array('error' => $e)));}
}
```

**XMLHttpRequest Objects: Example**

```javascript
// Stop the current request
getAllResponseHeaders();

// Returns all header/value pairs in the HTTP response as a string
getHeaderNames();

// Returns the value for header param in the HTTP response
getHeader(param, true);
```

**Constructing the HTTP Response**

```php
try {
    // We assume that PDO arguments have already been defined
    $pdo = new PDO($dsn, $db_username, $db_password, $opt);
    $stmt = $pdo->prepare("select address from postcodes where pc = ?");
    $data = $stmt->execute(array($param));
    $success = $stmt->execute(array($param));
}
catch (PDOException $e) {
    processData(array(0 => array('error' => $e)));}
```

**JavaScript and Forms Revisited**

```javascript
<form id="postcodeForm">
    <label>Postcode:</label>
    <input type="text" name="postcode" value="">
    <br>
    <label>Address:</label>
    <select name="adr" id="adr"></select>
</form>
```

**Retrieving the Data**

```php
// We have a database with a table
// postcode (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);
        $stmt = $pdo->prepare("select address from postcodes where pc = ?");
        $data = $stmt->execute(array($_POST['postcode']));
        $success = $stmt->execute(array($_POST['postcode']));
    } catch (PDOException $e) {
        processData(array(0 => array('error' => $e)));}
}
```

**Constructing the HTTP Response**

```php
try {
    // We assume that PDO arguments have already been defined
    $pdo = new PDO($dsn, $db_username, $db_password, $opt);
    $stmt = $pdo->prepare("select address from postcodes where pc = ?");
    $data = $stmt->execute(array($param));
    $success = $stmt->execute(array($param));
}
catch (PDOException $e) {
    processData(array(0 => array('error' => $e)));}
```

**JavaScript and Forms Revisited**

```javascript
// Adds or updates the name/value pair for
// the form with id form
form['name'] = val;

// Creates a new FormData object
var data = new FormData(formElem);

// Ideal we would validate data before sending
// the request
createSelectOptionsForPostcode(data);
```

**Retrieving the Data**

```php
<?php
// We have a database with a table
// postcode (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);
        $stmt = $pdo->prepare("select address from postcodes where pc = ?");
        $data = $stmt->execute(array($_POST['postcode']));
        $success = $stmt->execute(array($_POST['postcode']));
    } catch (PDOException $e) {
        processData(array(0 => array('error' => $e)));}
}
```

**Constructing the HTTP Response**

```php
try {
    // We assume that PDO arguments have already been defined
    $pdo = new PDO($dsn, $db_username, $db_password, $opt);
    $stmt = $pdo->prepare("select address from postcodes where pc = ?");
    $data = $stmt->execute(array($param));
    $success = $stmt->execute(array($param));
}
catch (PDOException $e) {
    processData(array(0 => array('error' => $e)));}
```

**JavaScript and Forms Revisited**

```javascript
// First argument of addEventListener () refers to input event
function addEventListener(input) {
    // Construct FormData object
    var data = new FormData(formElem);
    // We can use JavaScript's FormData object to deal with that
    var formData = new FormData(input);
    var formElem = document.getElementById('postcodeForm');
    var inpElem = document.querySelector('input[name=postcode]');
    inpElem.addEventListener('input', function(e) {
        // Validate data before sending
        createSelectOptionsForPostcode(data);
    });
```

**Retrieving the Data**

```php
<?php
// We have a database with a table
// postcode (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);
        $stmt = $pdo->prepare("select address from postcodes where pc = ?");
        $data = $stmt->execute(array($_POST['postcode']));
        $success = $stmt->execute(array($_POST['postcode']));
    } catch (PDOException $e) {
        processData(array(0 => array('error' => $e)));}
```

**Constructing the HTTP Response**

```php
try {
    // We assume that PDO arguments have already been defined
    $pdo = new PDO($dsn, $db_username, $db_password, $opt);
    $stmt = $pdo->prepare("select address from postcodes where pc = ?");
    $data = $stmt->execute(array($param));
    $success = $stmt->execute(array($param));
}
catch (PDOException $e) {
    processData(array(0 => array('error' => $e)));}
```

**JavaScript and Forms Revisited**

```javascript
// Adds an attribute
formData.append(attr, val);

// Adds an element
data.append('name', val, a);`
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 and element nodes have a method

```
function getElementsByTagName() {...}
```

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

The number of children of a node

```
function childNodes.length {...}
```

The XML DOM views an XML document as a tree structure of nodes

All nodes have attributes

```
function nodeName() {...}
```

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

```
function nodeValue() {...}
```

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

Processing the HTTP Response: JSON

```
function processXMLResponse(xml) {...}
```

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

```
function getElementsByTagName() {...}
```

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

```
function childNodes.length {...}
```

The number of children of a node

The XML DOM views an XML document as a tree structure of nodes

All nodes have attributes

```
function nodeName() {...}
```

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

```
function nodeValue() {...}
```

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

Processing the HTTP Response: JSON

```
function processXMLResponse(xml) {...}
```

Returns a JSON object containing the HTML response.

```
function getData() {...}
```

Parses a JSON string and returns the corresponding JavaScript object.

```
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: XML

```
function processXMLResponse(xml) {...}
```

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

```
function getElementsByTagName() {...}
```

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

```
function childNodes.length {...}
```

The number of children of a node

The XML DOM views an XML document as a tree structure of nodes

All nodes have attributes

```
function nodeName() {...}
```

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

```
function nodeValue() {...}
```

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

Processing the HTTP Response: JSON

```
function processXMLResponse(xml) {...}
```

Returns a JSON object containing the HTML response.

```
function getData() {...}
```

Parses a JSON string and returns the corresponding JavaScript object.

```
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: XML

```
function processXMLResponse(xml) {...}
```

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

```
function getElementsByTagName() {...}
```

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

```
function childNodes.length {...}
```

The number of children of a node

The XML DOM views an XML document as a tree structure of nodes

All nodes have attributes

```
function nodeName() {...}
```

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

```
function nodeValue() {...}
```

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

Processing the HTTP Response: JSON

```
function processXMLResponse(xml) {...}
```

Returns a JSON object containing the HTML response.

```
function getData() {...}
```

Parses a JSON string and returns the corresponding JavaScript object.

```
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: XML

```
function processXMLResponse(xml) {...}
```

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

```
function getElementsByTagName() {...}
```

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

```
function childNodes.length {...}
```

The number of children of a node

The XML DOM views an XML document as a tree structure of nodes

All nodes have attributes

```
function nodeName() {...}
```

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

```
function nodeValue() {...}
```

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

Processing the HTTP Response: JSON

```
function processXMLResponse(xml) {...}
```

Returns a JSON object containing the HTML response.

```
function getData() {...}
```

Parses a JSON string and returns the corresponding JavaScript object.

```
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 a HTTP Response

Processing the HTTP Response: JSON

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

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

Revision and Further Reading

* Read
  * Function Reference: XML Manipulation
  * Function Reference: JavaScript Object Notation

* Read
  * Mozilla and individual contributors: Document Object Model (DOM).
  * Mozilla and individual contributors: JSON.