AJAX Requests

Lecture 6 – CPEN400A

Based on the book “JavaScript: The Definitive Guide”, David Flanagan, O’Reilly

Outline

- **What is AJAX**
- XmlhttpRequest
- Callbacks and Error handling
- JSON

What is AJAX ?

- Mechanism for modern web applications to communicate with the server after page load
  - Without refreshing the current page
  - Request can be sent *asynchronously* without holding up the main JavaScript thread
- Stands for “Asynchronous JavaScript and XML”, but does not necessarily involve XML
- Complement of COMET (Server Push)

A brief history of AJAX

- Introduced by Microsoft as part of the Outlook Web Access (OWA) in 1998
- Popularized by Google in Gmail, Maps in 2004-05
- Term AJAX coined around 2005 in an article
- Made part of the W3C standard in 2006
  - Supported by all major browsers today

Uses of AJAX

- Interactivity
  - To enable content to be brought in from the server in response to user requests
- Performance
  - Load the most critical portions of a webpage first, and then load the rest asynchronously
- Security (this is doubtful)
  - Bring in only the code/data that is needed on demand to reduce the attack surface of the web application

Outline

- What is AJAX
- **XmlhttpRequest**
- Callbacks and Error handling
- JSON

Creating a new Request

- XMLHttpRequest: Constructor function for supporting AJAX
- open – opens a new connection to the server using the specified method (GET or POST) and to the specified URL or resource

```
var x = new XMLHttpRequest();
x.open("GET", "/example.txt");
```

GET versus POST

- Two popular methods to send HTTP Request
- GET – used to retrieve data from server by client (typically with no side effects), and does not send any additional data to the server
- POST – used to store data from HTML forms on the server (typically with side effects), and sends the form data to the server
- We'll typically use GET in this course

Sending a Request

Send the data to the server *asynchronously* and returns immediately. Takes a single parameter for the data to be sent (can be omitted for GET)

```
var x = new XMLHttpRequest();
x.open("GET", "/example.txt");
x.send(null);  // can simply say x.send();
// Returns here right after the send is complete
```

Setting up a Call-back

- Because the send returns right away, the data may not be sent yet (as it's sent asynchronously). Also, we have no way of knowing when the server has responded.

- We need to setup a callback to handle the various events that can occur after a send using the *onReadyStateChange* function

Stages of an XHR Request

- XMLHttpRequest Status Codes
  - UNSENT (0): open has not been called yet
  - OPENED (1): open has been called
  - HEADERS_RECIEVED(2): Headers have been received
  - LOADING(3): Response is being received
  - DONE(4) : Response is done
- Don’t use the direct numerical values in code

onReadyStateChange

```
var x = new XMLHttpRequest();
x.open("GET", "/example.txt");
x.onreadystatechange = function() {
    // Trigerred whenever ready state changes
};
x.send(null);  // can simply say x.send();
// Returns here right after the send is complete
```

Outline

- What is AJAX
- XmlhttpRequest
- **Callbacks and Error handling**
- JSON

XHR1: Old Method (Deprecated)

- Check whether the request's state has changed to DONE
- Check if the status of the request is 200 (denotes success in the HTTP protocol)
- Also, check if the response is of a specific type by examining the header
- If all three conditions match, then perform the action on message receipt (e.g., parse it)

Example of Callback function: XHR1

```
x.onreadystatechange = function() {
    if (x.readyState == 4 && x.status = 200) {
        var type = x.getResponseHeader(
                                    “Content-type”);
        if (type == “application.json”) {
            // Parse JSON here and take action
        }
    }
```

XHR2 Model: Recommended

- Does away with the onReadyStateChange
  - Triggers different events depending on response
  - Much cleaner but not all browsers support it (yet)
- Events trigerred by the XHR2 Model
  - Load: Response was received (does not mean that it was error-free, so still need to check status)
  - Timeout: Request times out (later)
  - Abort: Request was aborted (more later)
  - Error: Some other error occurred

Example of XHR2 Model

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "example.html);
xhr.onload = function() {
    if (xhr.status==200) {
            console.log( xhr.responseText );
            console.log("Request success");
    }
}
xhr.send();
```

Aborting AJAX Requests

- A request can be aborted after it is sent by calling the abort method on the request
- Request may have been already sent. If so, the response is discarded
- Triggers the Abort event handler of the request

```
xhr.onabort = function() {
    console.log("Request aborted");
}
```

Timeouts

- Can also specify timeouts in the request (though this is not supported by all browsers)
- Set timeout property in ms

```
xhr.timeout = 200; // 200 ms timeout
xhr.ontimeout = function() {
    console.log("Request timed out");
};
```

Errors

- These occur when there is a network level error (e.g., server is unreachable).
- Trigger the error event on the request
- NOT a substitute for checking status codes

```
xhr.onerror = function() {
    console.log("error occurred on request");
}
```

Class Activity

- Write the code for a request handler that issues an AJAX request every time you press the OK button, and cancels the last request every time you press the cancel button. Make sure the appropriate error messages are printed if the request times out (after 5 s), and also if the request has an error or is aborted
- Periodically display the list of messages that are “in-flight” from the client to the server

Outline

- What is AJAX
- XmlhttpRequest
- Callbacks and Error handling
- **JSON**

What is JSON ?

- Protocol to serialize and represent JavaScript Objects (JavaScript Object Notation)
- Useful for writing JavaScript objects to files, AJAX messages etc.
- Syntax is very similar to JavaScript itself
  - Not all object types are fully supported though

JSON Examples

- “{}” → Empty Object
- “[]” → Empty Array
- “hello” → String hello
- “function foo() { }” → Function foo
- “{ x:1, y:2, z: [1, 2] }” → Object with properties x = 1, y = 2 and z = [1, 2]
- “null” → null

How to handle JSON

- JSON.parse(string)
  - Converts string to JavaScript (code/data)
- JSON.stringify(object)
  - Converts object to JSON notation
- Header must be set to “Application/JSON”

Example

```
var xhr = new XMLHttpRequest();
xhr.open(“GET”, “example.html);
xhr.onLoad = function() {
    if (xhr.status==200) {
            if ( xhr.responseHeader(“Content-type”) ==
JSON) {
            var result = JSON.parse(xhr.responseText);
          // Do something with the result variable here
    }
}
xhr.send();
```

Outline

- What is AJAX
- XmlhttpRequest
- **Callbacks and Error handling**
- JSON