

Lecture 6: AJAX Requests

CPEN400A - Building Modern Web Applications - Winter 2020-1

Karthik Pattabiraman

The University of British Columbia
Department of Electrical and Computer Engineering
Vancouver, Canada



Electrical and
Computer
Engineering



October 8, 2020

What is AJAX ?



2

- 1 What is AJAX
- 2 *XmlHttpRequest*
- 3 Callbacks and Error Handling
- 4 JSON

What is AJAX ?



3

- 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



4

- 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



5

- 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 controversial)
 - Bring in only the code/data that is needed on demand to reduce the attack surface of the web application

XmlHttpRequest



6

- 1 What is AJAX
- 2 *XmlHttpRequest*
- 3 Callbacks and Error Handling
- 4 JSON

Creating a new Request



7

- **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

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



Sending a Request



9

- 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**)

```
1 var x = new XMLHttpRequest();  
2 x.open("GET", "/example.txt");  
3 x.send(null); // can simply say x.send();  
4 // 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

onReadyStateChange

```
1  var x = new XMLHttpRequest();
2  x.open("GET", "/example.txt");
3  x.onreadystatechange = function() {
4      // Triggerred whenever ready state changes
5  };
6  x.send(null); // can simply say x.send();
7  // Returns here right after the send is complete
```

Stages of an XHR Request



11

- XMLHttpRequest Status Codes
 - UNSENT (0): open has not been called yet
 - OPENED (1): open has been called
 - HEADERS_RECEIVED(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

Callbacks and Error Handling



12

- 1 What is AJAX
- 2 *XmlHttpRequest*
- 3 Callbacks and Error Handling
- 4 JSON

XHR1: Old Method (Deprecated)



13

- 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)
- Check if 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

```
1 x.onreadystatechange = function() {  
2     if (x.readyState == 4 && x.status == 200) {  
3         var type = x.getResponseHeader(  
4             "Content-type");  
5         if (type == "application.json") {  
6             // Parse JSON here and take action  
7         }  
8     }  
9 }
```

XHR2 Model: Recommended



14

- Does away with the `onReadyStateChange`
 - Triggers different events depending on response
 - Much cleaner but not all browsers support it (yet)
- Events triggered 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

```
1 var xhr = new XMLHttpRequest();
2 xhr.open("GET", "example.html");
3 xhr.onload = function() {
4     if (xhr.status==200) {
5         console.log( xhr.responseText );
6         console.log("Request success");
7     }
8 }
9 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

```
1 xhr.onabort = function() {  
2     console.log("Request aborted");  
3 }
```

Timeouts



16

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

```
1 xhr.timeout = 200; // 200 ms timeout
2 xhr.ontimeout = function() {
3     console.log("Request timed out");
4 };
```


Errors



17

- 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

```
1 xhr.onerror = function() {  
2     console.log("error occurred on request");  
3 }
```

Class Activity



18

- 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

JSON



19

- 1 What is AJAX
- 2 *XmlHttpRequest*
- 3 Callbacks and Error Handling
- 4 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

```
1  "{}" // Empty Object
2  "[]" // Empty Array
3  "hello" // String hello
4  "function foo() { }" // Function foo
5  "{ x:1, y:2, z: [1, 2] }" // Object with
   properties x = 1, y = 2 and z = [1, 2]
6  "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

```
1  var xhr = new XMLHttpRequest();
2  xhr.open("GET", "example.html");
3  xhr.onload = function() {
4      if (xhr.status==200) {
5          if ( xhr.getResponseHeader("Content-type")
6              == JSON) {
7              var result = JSON.parse(xhr.responseText);
7              // Do something with the result variable
7              here
8          }
9      }
10  xhr.send();
```

Table of Contents



- 1 What is AJAX
- 2 *XmlHttpRequest*
- 3 Callbacks and Error Handling
- 4 JSON