

AJAX

What is AJAX?

2

- Asynchronous JavaScript and XML
 - XML? Why?
- AJAX is a web development technique
- Allow us to fetch data/HTML from the server without blocking
 - Thus, making the application more responsive
- Usually the data is injected into the DOM without refreshing the whole page
- Originally was implemented using IFrame
- These days we use XMLHttpRequest

XMLHttpRequest - History

3

- Originally created by Outlook Web Access developers
- Was shipped as COM interface named *IXMLHTTPRequest* inside the MSXML library
- Mozilla offered the same API but named it *XMLHttpRequest*
 - ▣ Became a da facto standard
- Finalized by the W3C only at 2009
- XMLHttpRequest 2 was finalized at 2011

XMLHttpRequest

4

- **send** is non blocking
- Need to monitor the **readyState** flag
- Value 4 (Loaded) means “HTTP response was received completely by the browser”

```
var url = "http://www.yahoo.com";

var status = $("#status");
status.text("Sending request to " + url + " ...");

var request = new XMLHttpRequest();
request.open("GET", url);
request.send();

request.onreadystatechange = function () {
    if (request.readyState == 4 && request.status == 200) {
        status.text("Done " + request.responseText.length);
    }
}
```

XMLHttpRequest – More Details

5

- Can send the request synchronously
 - ▣ No good reason to do that
- Can read/write response/request headers

```
request.setRequestHeader("key", "value");
```

- Use abort method to cancel the operation
- send's first parameter is the request body (for HTTP post request)

```
request.send(myDataAsString);
```

XMLHttpRequest – Threading Model

6

- JavaScript is single threaded (Almost ... HTML5)
- The HTTP request/response is handled internally by the browser using dedicated threads
- On completion a message is posted to the single thread
- Only on the next event loop the single thread may noticed the AJAX request completion and react accordingly
- This means that from developer perspective AJAX request does not create parallelism
 - ▣ No need to lock or guard against race condition

Concurrent XMLHttpRequest(s)

7

- The developer may create as much AJAX request as he would like
- However, most browsers limit the number of concurrent AJAX request
- Beyond the limit, the AJAX request will not be sent to the server until an older request has completed
- These are latest limits reported by the community
 - ▣ Chrome – 6
 - ▣ Firefox – 6
 - ▣ IE 9,10,11 – 6,8,13

AJAX Wrappers

8

- The native XMLHttpRequest API is simple but not convenient
- As developer we would like to set the HTTP verb, url and specify success and error handlers
 - ▣ All other details should be handled internally
- Most JavaScript frameworks/libraries offer their own wrappers around XMLHttpRequest
- jQuery has one too

\$.ajax - GET

9

- A global function of the global jQuery object
 - ▣ Not available on jQuery wrapped set
- Type GET is the default value and can be omitted
 - ▣ Keep it

```
$.ajax({  
  type: "GET",  
  url: "http://localhost/demo/contact",  
  success: function (responseText) {  
    status.text("Done: " + responseText);  
  },  
  error: function () {  
    status.text("Error");  
  }  
});
```

\$.ajax – Sending data with GET

10

- The data property may hold a
 - string – Is appended to the URL as is
 - object – Is serialized into query string format

```
$.ajax({  
  type: "GET",  
  url: "http://localhost/demo/contact",  
  data: {id: 1},  
  success: function (responseText) {  
    status.text("Done: " + responseText);  
  },  
  error: function () {  
    status.text("Error");  
  }  
});
```

```
GET http://localhost/demo/contact?id=1 HTTP/1.1  
Host: localhost  
Connection: keep-alive  
Accept: text/html, */*; q=0.01
```

\$.ajax - POST

11

- Use the data option when posting to the server
 - ▣ Same behavior as GET but the data is sent as the request body and not as part of the URL

```
$.ajax({  
  type: "POST",  
  url: url,  
  data: { name: "Udi", email: "udi@gmail.com" },  
  success: function (responseText) {  
    status.text("Done: " + responseText);  
  },  
  error: function () {  
    status.text("Error");  
  }  
});
```

```
POST http://localhost/Demo/contact HTTP/1.1  
Host: localhost  
Connection: keep-alive  
Content-Length: 30  
Accept: */*  
Safari/537.36  
Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
  
name=Udi&email=udi%40gmail.com
```

Caching

12

- ❑ Browser might cache AJAX GET request
- ❑ In this case second request will not be sent
 - ❑ The \$.ajax succeeds as if the request was sent
- ❑ To eliminate caching
 - ❑ Use POST request
 - ❑ Or, append a random value into the URL
 - jQuery supports this technique using **cache** property

```
$.ajax({  
  type: "GET",  
  url: url,  
  success: function (responseText) {...},  
  error: function () {...},  
  cache: false,
```

```
GET http://localhost/Demo/...html?_=1396678859948 HTTP/1.1  
Host: localhost  
Connection: keep-alive
```

JSON – Java Script Object Notation

13

- ❑ XML is difficult to parse inside the browser
- ❑ JSON is based on JavaScript object literal notation
 - ▣ Therefore, a text format
 - ▣ Easy to read and write
- ❑ Is lighter than XML
- ❑ Built-in browser support
- ❑ Keys must be surrounded with double quotes

```
var json = '{"id": 1, "name": "Ori", "email": "ori@gmail.com"}';
```

Built-In Browser Support

14

- Modern browsers offer a global object named JSON

- ▣ For older browser consider using JSON2

<https://github.com/douglascrockford/JSON-js>

- Serialization

```
var obj = {  
  id: 123,  
  name: "Ori",  
  email: "ori@gmail.com",  
};  
  
var json = JSON.stringify(obj);
```

- Deserialization

```
var json = '{"id":1,"name":"Ori","email":"ori@gmail.com}';  
var obj = JSON.parse(json);
```

JSON Limitations

15

❑ Cannot serialize cyclic references

```
var ori = { id: 1, name: "Ori" };  
var roni = { id: 2, name: "Roni" };  
  
ori.sibling = roni;  
roni.sibling = ori;  
  
JSON.stringify(ori); // exception is thrown
```

❑ Object type is lost when serialization/deserializing

```
function Contact(name) {  
    this.name = name;  
}  
  
var ori = new Contact("Ori");  
var clone = JSON.parse(JSON.stringify(ori));  
  
console.log(clone instanceof Contact); // prints false
```

toJSON

16


- JSON.stringify checks for **toJSON** method on the method being serialized
- If found, the method is executed and the returned value is serialized into JSON
- The method should return an object not a string

```
var obj = {  
  id: 123,  
  name: "Ori",  
  toJSON: function () {  
    return {  
      name: this.name,  
    };  
  }  
};
```


Receiving JSON

17

- jQuery analyzes the response Content-Type
- If equals to **application/json** it automatically parses the returned data and pass it to the success handler
- Old servers might not specify a Content-Type
- Use **dataType** option



```
$.ajax({  
  type: "GET",  
  url: "/Home/Get",  
  dataType: "json",  
  success: function (data) {  
    console.log(data);  
  }  
});
```

Sending JSON

18

- By default when sending AJAX request the content type is **application/x-www-form-urlencoded**
- jQuery does not parse the data string, thus, it has no idea that we are sending JSON
- The misleading Content-Type might confuse the back end server
- Fix it



```
$.ajax({  
  type: "POST",  
  url: "/api/contact",  
  contentType: "application/json",  
  data: JSON.stringify({id:1, name: "Ori"}),  
});
```

Same Origin Policy

19

- AJAX request sent to different domain is prohibited
 - ▣ Security reason
- The following is considered different domain
 - ▣ **Schema** - http vs. https
 - ▣ **Host** – g.com vs. goole.com
 - ▣ **Port** – http://localhost:1 23 vs. http://localhost:1 24
- This means that one web site cannot share its data with other web sites
 - ▣ For example, Twitter would like any site to get a list of the 10 latest most popular twitts

JSONP

20

- A technique to overcome the same origin policy limitation
- It involves both server and client side modification
- Server
 - ▣ Instead of JSON string
 - ▣ Returns a JavaScript code which call an arbitrary method and passes the relevant JSON string
- ▣ The name of the method can be specified by the client

```
clientMethod({"id": 1, "name": "Ori"});
```

```
http://twitter.com/latest?callback=clientMethod
```

JSONP - Client

21

- Dynamically appends script tag into the HTML
 - ▣ src attribute should point to the server side URL which returns the JSONP content
- The browser downloads the script and executes it
- Need to remove the script tag
- Error handling it tricky

```
$("#head").append('<script src="/Home/Get" />');  
  
function clientMethod(str) {  
    console.log(JSON.parse(str));  
}
```

JSONP - jQuery

22

- jQuery supports JSONP invocation through \$.ajax
- This is a totally different mechanism since there is no use of XMLHttpRequest object
- Allow us to consume JSONP content as if it was plain HTTP service which returns JSON

```
$.ajax({  
  url: "/Home/Get",  
  dataType: "jsonp",  
  success: function (data) {  
    console.log(data);  
  },  
  error: function () {  
    console.log("ERROR");  
  }  
});
```

CORS

23

- HTML5 introduces the concept of CORS
 - ▣ Cross Origin Resource Sharing
 - ▣ A challenge-response mini protocol
- Old servers
 - ▣ Do not support the new protocol
 - ▣ Therefore the returned response is detected as invalid
 - ▣ The browser rejects the response
 - ▣ Application does not get a chance to process the response

CORS

24

- Assuming new browser and server
- The browser allows the request to be sent
 - ▣ Appends an HTTP header named **Origin**
 - ▣ Contains the URL of the requesting domain
- The response includes an HTTP header named **Access-Control-Allow-Origin**
 - ▣ Contains a list of allowed domains
- If the requesting domain is included inside the allowed list the browser let the application process the response as if it was a plain AJAX response

Access-Control-Allow-Credentials

25

- By default the browser does not send any cookie alongside the request
- To change this behavior
 - ▣ Specify `xhrFields.withCredentials = true`
 - ▣ Server must return `Access-Control-Allow-Credentials` HTTP header with value equals `true`

```
$.ajax({  
  url: "http://localhost:10659/api/contact",  
  type: "GET",  
  success: function (contacts) { },  
  error: function (jqXHR, text, error) { },  
  xhrFields: {  
    withCredentials: true,  
  }  
});
```

Summary

26

- AJAX manipulation is very common
- JSON is the web preferred data format
- Sending/Receiving JSON with \$.ajax is easy
- Same origin policy limits access to different domains
- Use JSONP to bypass same origin policy