Web Technologies — Week 10

Mikheil Rukhaia

International Black Sea University,
Tbilisi, Georgia

December 10, 2015

Outline

Ajax

Ajax

- Ajax (Asynchronous JavaScript and XML) is a technology to retrieve more data without refreshing a page.
- Before Ajax, similar effect was accomplished through a number of hacks, using iframes.
- Cornerstone of Ajax is `XMLHttpRequest` (XHR) object, making server requests and evaluating the responses.
- Despite the mention of XML, Ajax is format-independent.

XHR usage

- There are three main steps to retrieve data asynchronously from the server:
  1. Create a XHR object.
  2. Open the connection by specifying request type, URL and a boolean value indicating whether request is asynchronous.
  3. Send the request by specifying request body.

- Note: synchronous requests are deprecated; and for security reasons, requests can be made within the same domain only.

- Example:

```
xhr = new XMLHttpRequest();
xhr.open("get", "index.php", true);
xhr.send(null);
```

XHR usage (ctd.)

- When a response is received, the XHR object properties are filled with data.

Property	Description
`responseText`	contains the text that was returned as the body of the response.
`responseXML`	contains an XML document with the response data if the response has a content type of "text/xml" or "application/xml".
`status`	contains the HTTP status of the response (200s and 304 are considered successful).
`statusText`	contains the description of the HTTP status (unreliable across browsers).

XHR usage (ctd.)

- Asynchronous requests allow JavaScript code execution to continue without waiting for the response.
- XHR object has a `readyState` property indicating what phase of the request/response cycle is currently active:

Value	Description
0	Uninitialized.
1	Open.
2	Sent.
3	Receiving.
4	Complete.

XHR usage (ctd.)

- Example:

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){
  if (xhr.readyState == 4){
    if ((xhr.status >= 200 && xhr.status < 300)
        || xhr.status == 304) {
      alert(xhr.responseText);
    } else {
      alert("unsuccessful: " + xhr.status);
    }
  }
};
xhr.open("get", "index.php", true);
xhr.send(null);
```

GET and POST requests

- GET request is the most common to communicate with the server.
- If necessary, query-string arguments can be appended to the URL to pass information to the server.
- Usually, GET requests are two times faster than POST requests sending the same amount of data.
- Example:

```
xhr.open("post", "index.php?name=value", true);
xhr.send(null);
```

GET and POST requests (ctd.)

- POST request is expected to have data submitted as the body of the request.
- POST request also expects `Content-Type` header to be set to `application/x-www-form-urlencoded` value.
- The later can be specified via `setRequestHeader()` property of XHR.
- Example:

```
xhr.open("post", "index.php", true);
xhr.setRequestHeader("Content-Type",
                     "application/x-www-form-urlencoded");
xhr.send("name=value");
```

`FormData` object

- In the recent browsers there is a `FormData` object to serialize a form.
- You can pass a form directly to the constructor of `FormData`.
- Example:

```
data = new FormData(document.forms[0]);
xhr.open("post", "index.php", true);
xhr.send(data);
```

- Note: `FormData` takes care to set proper HTTP header information.

FormData object (ctd.)

- FormData can be used to create new data in the same format as a form for easy transmission via XHR.
- For this purpose use default constructor and append property of FormData.
- Example:

```
data = new FormData();
data.append("name", "value");
xhr.open("post", "index.php", true);
xhr.send(data);
```

CORS

- CORS (Cross-Origin Resource Sharing) is a security policy how the browser and server must communicate when accessing sources across origins.
- XHR object can be used with an absolute URL to make a request to a resource on another domain (has some limitations, like `setRequestHeader` cannot be used, etc.).
- But: XHR cannot be used for CORS in IE, for thi purpose there is `XDomainRequest` object, having even more restrictions (like no `status` property, etc.).

CORS (ctd.)

- `XMLHttpRequest` object has similar interfaces to the `XDomainRequest` object.

Property	Description
`abort()`	used to stop a request that is already in progress.
`onerror`	used instead of `onreadystatechange` to detect errors.
`onload`	used instead of `onreadystatechange` to detect successes.
`responseText`	used to get contents of response.
`send()`	used to send the request.

CORS (ctd.)

- Example

```
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
  if ("withCredentials" in xhr) {
    xhr.open(method, url, true);
  } else if
      (typeof XDomainRequest != "undefined") {
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    xhr = null;
  }
  return xhr;
}
```

CORS (ctd.)

- Example (ctd.)

```
request = createCORSRequest("get",
                            "http://www.example.com/");
if (request){
  request.onload = function() {
    alert(request.responseText);
  };
  request.onerror = function() {
    alert("An error occurred");
  };
  request.send();
}
```

Image pings

- Image pings are simple, cross-domain, one-way communication with the server.
- The data is sent via query-string arguments and the response can be anything, but it is enough to know the response has been received.
- Example:

```
img = new Image();
img.onload = img.onerror = function() {
  alert("Request was sent successfully!");
};
img.src = "http://www.example.com/
                        index.php?name=value";
```

Comet

- Comet is a term for a more advanced Ajax technique sometimes referred to as server push.
- Comet is described as the server pushing data to the page, allowing information to come in a manner closer to real time (ideal for live streaming).
- There are two popular approaches to Comet: long polling and streaming.
- The main difference is that while long polling establishes new connection on each HTTP request, streaming uses a single HTTP connection for the entire lifetime of the page.

Comet (ctd.)

- Example: server side (`streaming.php`)

```
<?php
  $i = 0;
  while(true){
    //output data and flush the buffer
    echo "Streaming. Lapsed $i seconds";
    ob_flush();
    flush();

    sleep(1); //wait for one second
    $i++;
  }
?>
```

Comet (ctd.)

- Example: client side (script only)

```
xhr = new XMLHttpRequest();
received = 0;
xhr.open("get", "streaming.php", true);
xhr.onreadystatechange = function() {
  var result;
  if (xhr.readyState == 3){
    result=xhr.responseText.substring(received);
    received += result.length;
    document.getElementById("live").
                                  innerHTML=result;
  } else if (xhr.readyState == 4){
    alert("Done!");
  }
};
xhr.send(null);
```

JSON

- JSON (JavaScript Object Notation) is the most popular for use in Ajax communication due to its speed of evaluation and easy data access for JavaScript code.
- JSON is preferred over XML because XML manipulation in JavaScript is quite different from browser to browser.
- There is a global `JSON` object with two methods: `parse()` and `stringify()`.

JSON (ctd.)

- Example:

```
xhr=createCORSRequest("post", "contacts.php");
xhr.onload = function () {
  contacts = JSON.parse(xhr.responseText);
  for (i=0; i < contacts.length; i++) {
    alter(contacts[i].name + ": " +
          contacts[i].email);
  }
}
contact = {
  name: "Jonh Smith",
  email: "smith@example.com"
};
xhr.send(JSON.stringify(contact));
```

Ajax with jQuery

Ajax with jQuery

- One of the strongest features of jQuery is its Ajax implementation.
- jQuery Ajax implementation is built on top of the method `jQuery.ajax()`.
- `ajax()` takes a JavaScript plain object as argument, which contains HTTP request information and a function that runs on successful answer.
- Example:

```
jQuery.ajax({
  "url": "index.php",
  "success" : function(data,status,jqXHR) {
    console.log(data,status,jqXHR);
                                        }});
```

Ajax with jQuery (ctd.)

- The default format to transfer data using jQuery Ajax is JSON, because it can be manipulated directly in JavaScript.

- You can use text or XML as well by specifying additional argument `dataType`.

- Example:

```
jQuery.ajax({
  "url": "index.php",
  "dataType" : "text",
  "success" : function(data) {
    $("body").html(data);
  }
});
```

Ajax with jQuery (ctd.)

- If you work with XML, you can use `$().find()` method to filter `data`.

- Example:

```
jQuery.ajax({
  "url": "users.xml",
  "dataType" : "xml",
  "success" : function(data) {
    name=$(data).find("name:first").text();
    console.log(name);
  }
});
```

Ajax with jQuery (ctd.)

- The default action in jQuery is to use a GET request.
- The POST request is made by specifying `type` and `data` arguments.
- Example:

```
jQuery.ajax({
  "url": "index.php",
  "type" : "post",
  "data" : { "submit" : "ok"},
  "success" : function(data) {
    $("body").html(data);
  }
});
```

Ajax with jQuery (ctd.)

- There are also `$.get()` and `$.post()` methods, but they are more restricted than `$.ajax()`.
- `$.get()` takes two parameters: the URL and the success function.
- `$.post()` takes additionally third parameter – the data that has to be sent.

Ajax with jQuery (ctd.)

- Example:

```
$("form").on("submit", function(event) {
  $form = $(this);
  event.preventDefault();
  jQuery.post(
    $form.attr("action"),
    $form.serialize(),
    function(data) {
      $("#submitted").html(data);
    }
  );
});
```

Laboratory Work

Exercises

- Create image pings and retrieve total number of clicks from server via Ajax (i.e. implement something like FB likes).

Discussion?!