Responsive Web Applications with Asynchronous JavaScript Remote Procedure Calls

Sivan Toledo
Tel-Aviv University

The XMLHttpRequest Object

A JavaScript object that the browser should support (modern browsers do)

Allows a client-side JavaScript program to send requests to a remote server

When the server responds to the request, the browser invokes a JavaScript function that uses the server's response to update the user interface

Creating the XMLHttpRequest Object

```
<script type="text/javascript">

var request = false;

try {
  request = new XMLHttpRequest();
} catch (now_try_MSXML2) {
  try {
    request = new
      ActiveXObject("Msxml2.XMLHTTP");
  } catch (now_try_MS) {
    try {
```

Creating the XMLHttpRequest Object (cont)

```
      request = new ActiveXObject(
                        "Microsoft.XMLHTTP");
    } catch (failed) {
      request = false;        we signal failure
    }
  }
}

if (!request)
  ... do something else,
      the browser does not support XMLHttpRequest
</script>
```

Sending a Request

```
function handleResp() {
   ...  we'll examine this later
}

var url = ...;  we'll examine this later

request.open("GET",    http method
             url,      where to send, arguments
             true);    yes, make it asynchronous

request.onreadystatechange = handleResp;
request.send(null);  no actual data to send
```

The Arguments of XMLHttpRequest.open

Between 2 and 5 arguments

- The HTTP method
- The URL; for GET requests, the URL includes the arguments (key-value pairs of strings)
- (optional) Is the request asynchronous? True by default
- (optional) Username for authentication
- (optional) Password for authentication

Constructing the URL

```
var url = "/MyWebApp/servlet/XYZ?" +
          "param1=" + escape(value1) +
    "&" + "param2=" + escape(value2);
```

The URL is relative: the browser will only honor requests to communicate with the server from which the script arrived

GET requests specify the parameters in the URL

Escape strings to avoid forbidden URL characters

Handling the Request

```
function handleResp() {
  if (request.readyState == 4) {
    We received a response; otherwise we have not
    if (request.status == 200) {
      ... do something useful with the response
    } else {
      ... processing is completed, but the browser
      ... or the server reported some error: URL
      ... not found, unauthorized, etc
    }
  }
}
```

What To Do with a Bad Status Code

If the user is expecting something (e.g. a response to a search query), tell her it's not coming; don't stop the application, maybe she just moved into a bad wireless spot and will reconnect soon, etc

If the script was trying to do something in the background (cache data for later), keep quiet

The server is **your** server (sandbox), so really unexpected codes indicate that you have a bug

Using the Data Sent by the Server

The reply of the server is represented by one of two properties of the request object

`request.responseText` *if the server sent text/plain*
`request.responseXML` *if the server sent text/xml*

Plain-text responses can be formatted arbitrarily: comma separated (or any other separatator), etc

Use string and regular-expression methods to parse plain-text responses

Taking a Nonstandard Shortcut

```
var b = document.getElementById('body');
b.innerHTML = request.responseText;
```

We just replace the contents of body (or of any other document element) by the HTML that the server sent

Setting innerHTML parses the right-hand side and changes the document tree accordingly

Not part of the DOM! Some find it objectionable

Using responseXML

This field contains the servers response **only if the server sent text/xml** data; data in other formats, including text/html, is only available as a string in responseText

The type of responseXML is a DOM node of type "document" (nodeType==9)

Use DOM methods to extract the information in the response

Other XMLHttpRequest Methods

`abort()` — *if your script decides that the request timed out or if it is no longer relevant (e.g., the user clicked another button)*

```
setRequestHeader(
  "field_name",
  "value")
```
to set a specific HTTP header

```
getResponseHeader(
  "field_name"
  )
```
to read a specific HTTP header

Using responseXML (Example)

```
var r =
  request.responseXML.documentElement;

var header =
  r.getElementsByTagName('header')[0].
    firstChild.data;
...
```

This uses standard DOM methods on the response