INT222 - Internet Fundamentals

Week 12: AJAX and Canvas

Agenda

- Ajax
  - simulating a call to a Web service
- HTML5 Canvas

What is AJAX

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML.
- Coined in 2005 by Jesse James Garrett.
- Not a technology , but a "new" way to use existing technologies.
- AJAX is a group of interrelated web techniques used on the client-side for creating fast and dynamic web pages.

Technologies used in AJAX

- HTML
- CSS
- JavaScript
- The Document Object Model (DOM)
- XML/JSON, XSLT
- XMLHttpRequest object

Ajax Examples

- Google Maps
  http://maps.google.com/

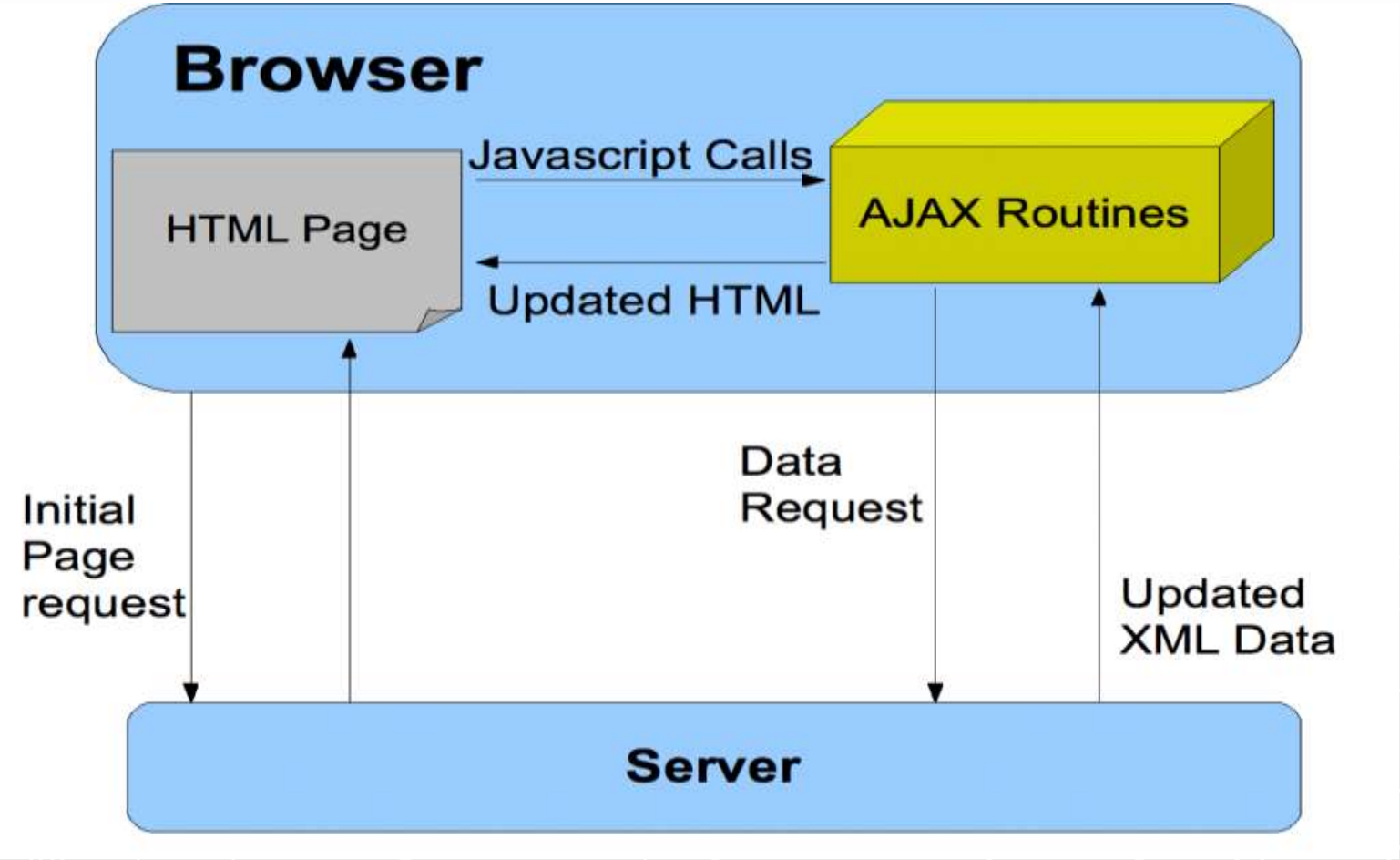
- Google Suggest
  http://www.google.com/webhp?complete=1&hl=en

- Gmail
  http://gmail.com/

How does it works?

- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

The AJAX model



Comparing with classic apps

- Conventional web application transmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.

Packaging data in AJAX model

- XML was originally used as the format in AJA**X**.
- JSON is used more than XML nowadays .
  - Advantages to use JSON: being lighter and a part of JavaScript.
- Any format, including plain text, can be used.

Features

- Make requests to the server without reloading the page
- Receive and work with data from the server
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background
- Intuitive and natural user interaction. No clicking required only Mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven

XMLHttpRequest object

- It is the most important part of AJAX.
- It is a JavaScript object
  - Designed by MS and adopted by Mozilla, Apple, …
- It provides an easy way to retrieve data from a URL without having to do a full page refresh.
- Creation:

```
var myRequest = new XMLHttpRequest();
```

XMLHttpRequest Methods

- **abort()**
- **getResponseHeader()**
- **open()** // Initializes a request.
- **overrideMimeType()**
- **send()** // Sends the request.
- **...**

XMLHttpRequest properties

- Onreadystatechange
- readyState
- responseText
- responseType // set to change the response type
- responseXML

Writing AJAX

Step 1 – makes an HTTP request object

```
// creating a cross-browser instance
var httpRequest;
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
  httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 8 and older
  httpRequest = new
  ActiveXObject("Microsoft.XMLHTTP");
}
```

Writing AJAX

- **Step 2 – register a request listener**

```
httpRequest.onreadystatechange = reqListener;

function reqListener () {
    // process the server response
};
```

- **Or:**

```
httpRequest.onreadystatechange = function(){
  // process the server response
};
```

Writing AJAX

- **Step 3 - make the request**

```
// Specifies the type of request
httpRequest.open('GET',
    'http://www.example.org/some.file', true);

// Sends the request off to the server.
httpRequest.send(null);
```

Parameters of open() and send()

- Parameters of the open(*method, url, async*) method:
  - 1st para: HTTP request method - GET, POST, …
  - 2nd para: the URL of the page this requesting
  - 3rd pare: sets whether the request is asynchronous
- Parameters of the send(*string*) method:
  - The *"string"* para: Only used for POST requests.
    - any data you want to send to the server if using POST method. This can be as a query string, like:

name=value&anothername="+encodeURIComponent(myVar)+"&so=on"

Writing AJAX

- **Step 4 – handling the server response**

```
// check for the state of response
if (httpRequest.readyState === 4) {
  // everything is good, the response is received
} else {
  // still not ready
}
```

  - the readyState values:
    - 0 (uninitialized), 1 (loading), 2 (loaded), 3 (interactive), 4 (complete)

Writing AJAX

- **Step 4 – handling the server response (cont')**

```
// check is the response code of the HTTP server response
if (httpRequest.status === 200) {
    // perfect!
} else {
    // there was a problem with the request,
    // for example the response may contain a 404 (Not Found)
    // or 500 (Internal Server Error) response code
}
```

Writing AJAX

- Set the button to start

```
<button type="button" onclick="makeRequest();">
 Make a request</button>
```

Working with the XML response

- Suppose server response is in XML format:

```
<?xml version="1.0" ?>
<root>
    I'm a test.
</root>
```

- Parsing XML data

```
var xmldoc = httpRequest.responseXML;
 var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);
```

Working with the JSON response

- Suppose server response is in JSON format:

  {"Name": "Kevin", "Age": 22 }

- Parsing JSON data:

```
// Javascript function JSON.parse to parse JSON data
var jsonObj = JSON.parse(http_request.responseText);
Var name = jsonObj.Name;
```

About JSON

- JSON stands for JavaScript Object Notation.
- JSON is a lightweight text based data-interchange format.
- JSON filename extension is **.json**
- JSON Internet Media type is **application/json**
- JSON is built on two structures:
  - JSON object: a collection of name/value pairs.
  - JSON array: an ordered list of values.
- Example (https://zenit.senecac.on.ca/~wei.song/int222/ajax/student.json)

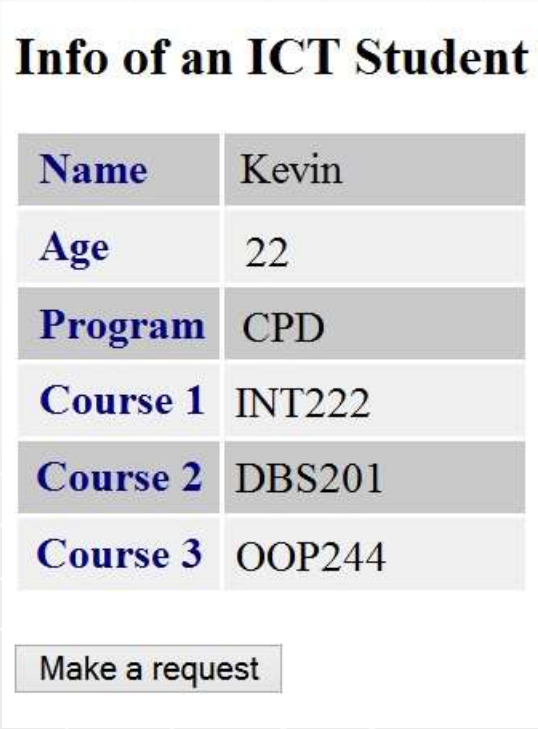
```
{“name”: “Kevin”, “age“: 22, “program”: “CPD”,
“courses”, [“INT222”, “DBS201”, “OOP244”]}
```

JavaScript: simulating AJAX calls to Web Services

- Example 1: calling a web service with JSON object response
  - Code: https://zenit.senecac.on.ca/~wei.song/ajaxjson.html
- Example 2: calling a web service with JSON array response
  - Code: https://zenit.senecac.on.ca/~wei.song/ajaxjson2.html

Exercise

- Create an HTML5 page that loads data with an AJAX call to a web service.
- The web service is simulated by a static JSON file: https://zenit.senecac.on.ca/~wei.song/int222/ajax/student.json
- You have to HTML page to the ZENIT server to make the AJAX code work.
  - ATTN: usually, Cross-Domain AJAX call is not allowed for security reason.
- The HTML page is showed at right.



HTML5:
The <canvas> Element

<canvas>

- <canvas> - an HTML5 element to give you a drawing space in JavaScript on your Web pages.
- It allows for dynamic, scriptable rendering of 2D shapes and bitmap images.
- It is only a container for graphics. You must use a script to actually draw the graphics.
- Canvas consists of a drawable region defined in HTML code with *height* and *width* attributes.

The <canvas> Element

- Example:

```
<canvas id="myCanvas" width="150" height="150"></canvas>
```

- Always specify an id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas.
- You can have multiple <canvas> elements on one HTML page.

By default, the <canvas> element has no border and no content

Fallback content

- Providing alternate content inside the <canvas> element, in case of browsers don't support <canvas>.
  - text description

```
<canvas id="myCanvas" width="200" height="100"
  style="border:10px solid #000000;">
      Your browser does not support the HTML5 canvas tag.
  </canvas>
```

  - static image

```
<canvas id="clock" width="150" height="150">
   <img src="images/clock.png" width="150" height="150"
alt=""/>
</canvas>
```

Checking for support in Scripts

- By testing for the presence of the getContext() method.

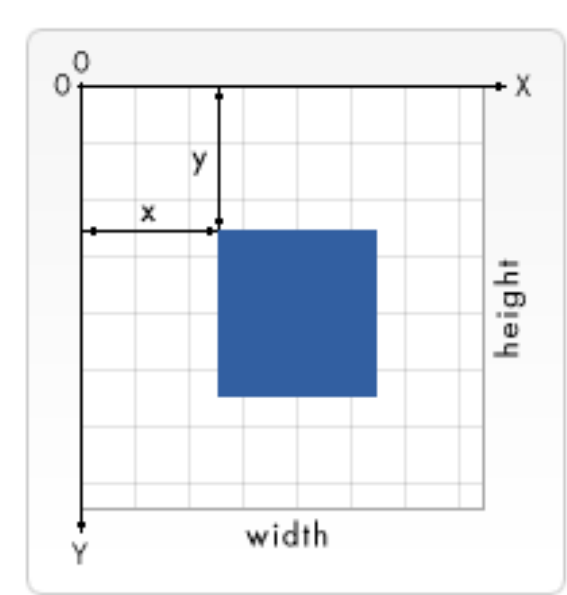
```
var canvas = document.getElementById('myCanvas');
if (canvas.getContext){
    var ctx = canvas.getContext('2d');
    // drawing code here
} else {
    // canvas-unsupported code here
}
```

A skeleton template

```
<!DOCTYPE html>
<html>
 <head>
 <title>Canvas Test</title>
  <script type="application/javascript">
    function draw() {
      var canvas = document.getElementById("my-canvas");
      if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        ctx.....// drawing code goes here.
      }
    }
  </script>
  <style type="text/css">
                canvas { border: 1px solid black; }
  </style>
 </head>
 <body onload="draw();">
   <canvas id="my-canvas" width="150" height="150"></canvas>
 </body>
</html>
```

Drawing rectangles

- Three functions
  - fillRect(*x*, *y*, *width*, *height*)
    - Draws a filled rectangle.
  - strokeRect(*x*, *y*, *width*, *height*)
    - Draws a rectangular outline.
  - clearRect(*x*, *y*, *width*, *height*)
    - Clears the specified rectangular area, making it fully transparent.



Drawing paths

beginPath()

- Creates a new path. Once created, future drawing commands are directed into the path and used to build the path up.

closePath()

- Closes the path so that future drawing commands are once again directed to the context.

stroke()

- Draws the shape by stroking its outline.

fill()

- Draws a solid shape by filling the path's content area.
- When you call fill(), any open shapes are closed automatically, so you don't have to call closePath().

Moving the Pen & Drawing Lines

moveTo(x, y)

- Moves the pen to the coordinates specified by x and y.

lineTo(x, y)

- Draws a line from the current drawing position to the position specified by x and y.

Drawing a Triangle

```
function draw() {
    var canvas = document.getElementById('canvas');
    if (canvas.getContext){
       var ctx = canvas.getContext('2d');
       ctx.beginPath();
       ctx.moveTo(75,50);
       ctx.lineTo(100,75);
       ctx.lineTo(100,25);
       ctx.fill();
    }
}
```

Drawing Arcs

arc(*x*, *y*, *radius*, *startAngle*, *endAngle*, *anticlockwise*)

- Draws an arc.
- x, y: coordinate
- Start: starting angle (e.g. 0)
- Stop: stopping angle (e.g., 2 * Matho.PI)

➢ To actually draw the circle, use stroke() or fill().

Using images

- One of the more exciting features of <canvas> is the ability to use images.
  - These can be used to do dynamic photo compositing or as backdrops of graphs, for sprites in games, and so forth.
- Drawing images

drawImage(*image*, *x*, *y*)

- Draws the CanvasImageSource specified by the image parameter at the coordinates (x, y).

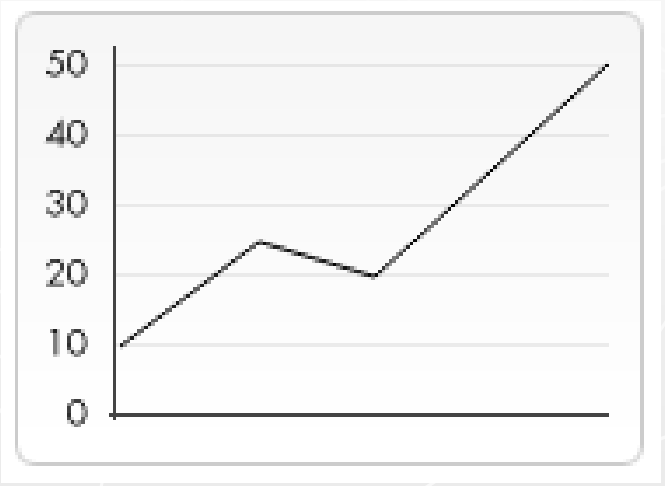
drawImage(*image*, *x*, *y*, *width*, *height*)

- This adds the width and height parameters, making the image scalable.

Using images

```
function draw() {
    var canvas = document.getElementById("my-
  canvas");
    if (canvas.getContext) {
      var ctx = canvas.getContext("2d");

      var img = new Image();
      img.onload = function(){
                ctx.drawImage(img,10,10);
                ctx.beginPath();
                ctx.moveTo(40,106);
                ctx.lineTo(80,76);
                ctx.lineTo(113,86);
                ctx.lineTo(180,25);
                ctx.stroke();
      };
      img.src = 'backdrop.png';
    }
  }
```



Filling Text

- To draw text on a canvas, the most important property and methods are:
  - font
    - defines the font properties for text
  - fillText(*text,x,y*)
    - Draws "filled" text on the canvas
  - strokeText(*text,x,y*)
    - Draws text on the canvas (no fill)
- Example

```
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);
```

Gradients

- Gradients can be used to fill rectangles, circles, lines, text, etc. Shapes on the canvas are not limited to solid colors.
- There are two different types of gradients:

createLinearGradient(*x,y,x1,y1*)

- Creates a linear gradient

createRadialGradient(*x,y,r,x1,y1,r1*)

- Creates a radial/circular gradient

Canvas Examples

- The Canvas Examples are at the location:
  https://github.com/wsong18/front-end-web-demos/tree/master/multimedia
- Include:
  - canvas_test_rect.html
  - canvas_test_tri.html
  - canvas_test_arcs.html
  - canvas_test_using_img.html
  - canvas_test_text.html
  - canvas_test_grad.html

Thank You!