University of Regina

CS 215 Web Oriented Programming

JavaScript & DOM Manipulation

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs215/

Readings

- Continue reading in Chapters 13 - 16

- Midterm Exam: Tuesday Oct 24 @ 2:30 PM
  - covering material and readings up to the end of this topic
- Assignment 3: Thursday Oct 26 @ 11:55 PM
  - doing the assignment will help you to study for the exam

Dynamic HTML

- Dynamic HTML is not a new version of HTML
  - it is a collection of technologies that allow for an HTML document to be dynamically manipulated **after** it has been loaded by the browser
  - many things can be changed
    - style
    - content
    - insert & delete elements
    - position of floating elements
  - changes are made by an embedded script (JavaScript) that has access to manipulate the DOM
  - unfortunately, support for dynamic HTML is a bit different across different browsers → test before you release

Changing Colours & Fonts

- Dynamically modifying colours and fonts is simply a matter of overwriting the style properties
  - example 2:
    - dynFont.html
  - example 1:
    - dynColors.html
    - dynColors.js

- The second example shows how we can use the same JavaScript function for two purposes

Changing the Style Class

- These examples changed the specific elements of the style dynamically
- In many cases, it is better to define multiple styles in CSS, and use JavaScript to change which style is used
    - avoids the need to specify style information within the JavaScript code
    - allows the styles to be defined and tested independently

```
function mouseDown (event){
  event.currentTarget.className = “pressed”;
}
```

- exam practice: do this for the examples on the previous slide

Dynamic Content

- Changing the content of an existing HTML element can be done by modifying the `value` or `innerHTML` property of the DOM object (depending on whether it is a form element or a regular tag)
  - the browser will detect such a change, and update the page
  - example
    - dynValue.html
    - dynValue.js

JavaScript Traversal of the DOM

- The DOM tree structure can be easily traversed
  - starting at a particular node, we can
    - see if there are any child nodes
      ```
      var node = document.getElementById(“myList”);
      var children = node.childNodes.length;
      ```
    - access the previous sibling
      ```
      var prev = node.previousSibling;
      ```
    - access the next sibling
      ```
      var next = node.nextSibling;
      ```
    - access the parent
      ```
      var parent= node.parentNode;
      ```
    - access an array of the children
      ```
      var children_array = node.childNodes;
      ```
    - access the first child
      ```
      var first = node.firstChild;
      ```
    - access the last child
      ```
      var last = node.lastChild;
      ```

DOM Tree Modification

- There are a few built-in methods that allow us to easily manipulate the DOM
  - insertBefore(new_child, ref_child);
  - replaceChild(new_child, old_child);
  - removeChild(old_child);
  - appendChild(new_child);

  - can use createElement(element_type) to create a new HTML element
    - need to assign the parameters programmatically to this element

DOM Tree Modification Example

- http://www.w3schools.com/jsref/met_node_insertbefore.asp

  - note the need to create the HTML element and the text within the element in separate steps

```
var h=document.createElement(“h1");
var t=document.createTextNode("Hello World");
h.appendChild(t);
document.body.appendChild(h);
```

  - another option is to simply edit the content of the HTML tag using the innerHTML property

```
var h=document.createElement(“h1");
h.innerHTML = “Hello World”;
document.body.appendChild(h);
```

Positioning

- For some elements, their ability to be manipulated in dynamic HTML depends on how they are defined
  - we have already discussed one example of this
    - event registration
  - another example is positioning
    - we initially discussed three options for positioning in the CSS lectures
    - the default value for the `position` property is `static`
      - statically positioned elements cannot be moved with dynamic HTML
    - if you want an element to be moveable, set position to `absolute` or `relative`

Absolute Positioning

- The frame of reference for absolute positioning is either the document itself, or the upper-left corner of an enclosing element if it is also absolutely positioned

```
div.indent {position:absolute; left:100px; top:200px;}
p.inside {position:absolute; left:100px; top:200px;}
p.outside {position:absolute; left:-50px; top:-100px;}

<div class="indent">
<p>This is a paragraph.</p>
<p class="inside">This is another paragraph.</p>
<p class="outside">This one is outside.</p>
</div>
```

Relative Positioning

- Relative positioning adjusts the positioning in relation to where the element would have normally been placed by the browser
  - if left and top have no values, this has no effect but makes the element available for dynamic re-positioning via JavaScript

```
span.givemesomespace {position: relative; left:15px;}

.specialtext {font:2em Times; color:red; position:relative;
top:15px;}
```

Moving Elements

- Moving elements is simpler than you might think
  - simply change the top and left properties of the style with JavaScript
    - access the style properties through the style object associated with the object
    - property names are the same in the DOM as in CSS, with the exception of CSS styles that have a dash in their name (those are specified in camel-case: border-color= borderColor property)
    - must still adhere to the rules of CSS (i.e., positions require a unit of measurement)

```
function moveIt (it, new_top, new_left) {
  var style = document.getElementById(it).style;
  style.top = new_top + “px”;
  style.left = new_left + “px”;
}
```

Element Visibility

- Manipulating element visibility is as easy as moving elements
  - change the CSS property `visibility` between the values of `visible` and `hidden`

```
function flipIt(it) {
  var style = document.getElementById(it).style;
  if (style.visibility == “visible”) {
    style.visibility = “hidden”;
  } else {
    style.visibility = “visible”;
  }
}
```

Element Stacking

- In addition to changing the position of elements, we can also change how the system treats elements that are overlapping
  - CSS attribute z-index
    - integer values
    - higher valued elements are placed in front of lower valued elements
    - can be changed dynamically in JavaScript (zIndex)
  - Example
    - stacking.html
    - stacking.js

Accessing Mouse Events

- When an event occurs that is a result of a mouse action, the Event object includes a set of properties that are mouse specific
  - clientX and clientY
    - coordinates from the upper-left corner of the browser window
  - screenX and screenY
    - coordinates from the upper-left corner of the screen
  - button
    - which button was pressed
  - altKey, ctrlKey, shiftKey, metaKey
    - whether specific control keys on the keyboard were down

Example

- If we capture onmousedown and onmouseup events in the body, we can detect where the mouse is clicked, and perform some action

  - example:
    - anywhere.html
    - anywhere.js

Animating the Movement of Elements

- When we move HTML elements, we can simply change the top and left properties
  - causes the element to instantly be moved to the new place
  - sometimes, this isn't what we want
    - a smooth animation allows the user to see where the element came from and where it is going
    - the effect of the move is much less of a surprise
- The simplest way to do animation in JavaScript is to use stop-gap animation (also called stop motion animation)
  - incrementally move the object from its source to destination, showing the location at each step
    - less time between steps = smoother animation

Animation

- There are two methods that can be used to implement stop-gap animation
  - setTimeout
    - two parameters: the code to be executed, and the number of seconds to wait before executing it

      ```
      setTimeout("mover()", 20);
      ```
  - setInterval
    - same two parameters
    - executes the code repeatedly, waiting the time interval before the next execution

      ```
      setInterval("mover()", 20);
      ```
    - returns an id value that can be used as the parameter to a `clearInterval()` function call that stops the timer

Animation

- The easiest way to move the objects is to do so on a straight line between the source and destination
  - example:
    - moveText.html
    - moveText.js
  - note that this example assumes a diagonal line (incrementing the top and left one pixel at a time)
  - a more flexible method would be to determine how many steps to take, and then calculate the increment of the x and y dimensions independently

Dragging and Dropping Elements

- Using the mousedown, mouseup, and mousemove events, we can drag and drop HTML elements with just a few lines of code
  - example:
    - dragNDrop.html
    - dragNDrop.js
  - this example dynamically adds and removes event handlers
    - handling the mousemove and mouseup events is only needed after a mousedown is performed

Homework

- Catch up on your textbook reading
- Study for Midterm Exam

- Upcoming deadlines:
  - Midterm Exam: Tuesday Oct 24 @ 2:30 PM
  - Assignment 3: Thursday Oct 26 @ 11:55 PM

  - tip: you should plan to have most of the assignment done by the time of the midterm