University of Regina

CS 215
Web Oriented Programming

JavaScript, DOM, & Event Handling

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

Readings

- Continue reading in Chapters 13 - 16

Client-Side JavaScript

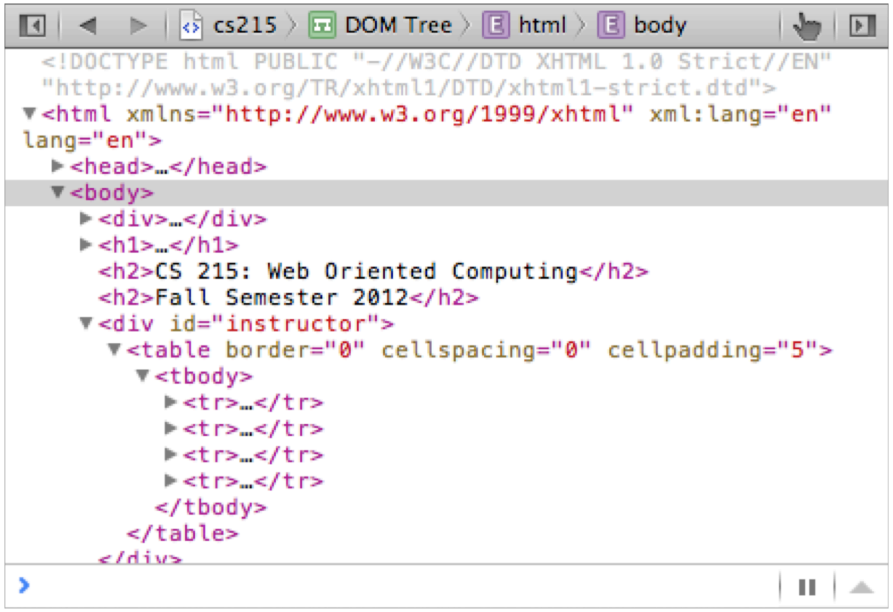
- Client-side JavaScript extends core JavaScript by adding a collection of objects, methods, and properties
  - these allow the JavaScript code to interact with HTML documents on the browser
    - write data to the document as it is being rendered
      - document.write()
    - open a modal window
      - alert(), confirm(), and prompt()
    - access data within a form
    - access HTML element properties
    - change HTML element properties

JavaScript Execution Environment

- `Window` object
  - represents the browser window that displays the document
  - properties and methods are visible to all JavaScript (implicit or explicit) within the document
    - treated as global variables and methods
    - all new variables and methods are added to the Window object
- `Document` object
  - represents the displayed HTML document
  - a property of the Window object

Document Object Model (DOM)

- DOM is an abstract model that defines the interface between HTML documents and application programs
  - essentially, it is an API
  - tree structure of the HTML and CSS information
  - accessed through the document object
  - four levels:
    - DOM 0 – supported by all
    - DOM 1
    - DOM 2 – added CSS
    - DOM 3 – latest approved version (2004)



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
▼<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
  ▶<head>…</head>
  ▼<body>
    ▶<div>…</div>
    ▶<h1>…</h1>
     <h2>CS 215: Web Oriented Computing</h2>
     <h2>Fall Semester 2012</h2>
    ▼<div id="instructor">
      ▼<table border="0" cellspacing="0" cellpadding="5">
        ▼<tbody>
          ▶<tr>…</tr>
          ▶<tr>…</tr>
          ▶<tr>…</tr>
          ▶<tr>…</tr>
         </tbody>
       </table>
     </div>
```

DOM & JavaScript

- HTML elements are represented as objects
- HTML element attributes are represented as properties

```
<input type =“text” name = “address”>
```

  - represented as an object with two explicitly defined properties
  - in most cases, the property names in JavaScript/DOM are the same as their corresponding attribute names in HTML
  - can use dot notation to access properties or other embedded objects

Element Access

- There are several ways to access the elements of an HTML document from JavaScript

- DOM Address Indexing
  - indexing within the forms and elements arrays

```
<form action="">
    <input type="button" name="pushme" />
</form>

var button = document.forms[0].elements[0];
```

  - if the document changes, the indices could change too

Element Access

- Element Names
  - require all elements in which the element is embedded to have names (except body, which can't be named)

```
<form name="myForm" action="">
   <input type="button" name="pushMe" />
</form>

var button = document.myForm.pushMe;
```

  - valid for HTML5, but not XHTML (the form tag cannot be named)

Element Access

- getElementById method
  - because the element's id is unique within the document, this approach will search the DOM to find the required object

```
<form action="">
    <input type="text" name="username" id="username" />
    <input type="button" name="pushMe" id="pushMe" />
</form>

var button = document.getElementById("pushMe");
var username = document.getElementById("username").value;
```

  - most flexible approach
  - can even use variables (Strings) in the parameter of getElementById
  - not just limited to form elements – can access anything with an id

Checkboxes and Radio Buttons

- Accessing individual checkboxes and radio buttons are a bit trickier
  - we don’t want to have to assign an id to each element
  - better to assign an id to the form, and access the implicit array for the group of checkboxes or radio buttons

```
<form id = ”theForm">
  <input type = "checkbox"  name = "toppings"
         value = "olives" />
  ...
  <input type = "checkbox"  name = "toppings"
         value = "tomatoes" />
</form>

var numChecked = 0;
var theForm = document.getElementById(”theForm");
for (index = 0; index < theForm.toppings.length; index++){
  if (theForm.toppings[index].checked){
    numChecked++;
  }
}
```

Events & Event Handling

- Much of what we do in JavaScript is **event-driven programming**
  - detect activities in the browser
  - perform actions/computation initiated by these actions
- Two levels of event handling
  - DOM 0
    - supported by all browsers that support JavaScript
  - DOM 2
    - supported only by the latest versions of browsers

Events and Event Handlers

- Event
  - notification that something specific has occurred
  - in JavaScript, an object is implicitly created by the browser in response to something having happened
- Event Handler
  - a script (function) that is implicitly executed in response to the appearance of an event
- Registration
  - the process of connecting an event handler to an event
  - two approaches
    - assign handler via tag attributes
    - assign handler function to a DOM object property

Events and Associated Tag Attributes

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

Registration Methods (DOM0)

- Assign the event handler code to an event tag attribute

```
<input type=“button” id=“myButton” onclick=“alert(‘you
clicked the button’);” />

<input type=“button” id=“myButton”
onclick=“myButtonHandler();” />
```

- Assign the event handler function to the event property of the object

```
document.getElementById(“myButton”).onclick =
myButtonHandler;
```

  - this statement must follow both the handler function and the form element so that the browser has seen both before assigning the property to the function name

Events from <body>

- The two most commonly used events for the body tag are load and unload
  - allow us to call functions when the document is finished loading or unloading
    - initialization of variables
    - initiate actions that occur as soon as the document is ready
    - clean up variables

```
<body onload="greeting();" onunload="goodbye();">


function greeting() {
    alert ("hello!");
}
function goodbye() {
    alert ("goodbye!");
}
```

Events from Buttons

- For simple buttons, use the onclick property

```
<input type="button" id="myButton" onclick="myButtonHandler();" />
```

- For radio buttons, things are much more complex
  - can assign an event handler for each button, passing in a parameter
    - <input type="radio" name="meal" value="chicken" onclick="processMealChoice('chicken');" />
  - assign an event handler that checks which button is "on"
  - see example
    - radio_click2.html
    - radio_click2.js
    - radio_click2r.js

Pros & Cons of Assigning Event Handlers to Object Properties

- Assigning event handlers to object properties has both good and bad aspects (in comparison to registering the event within the HTML attributes)
  - bad
    - can't pass in parameters
    - must add special registration code at the end of the page
  - good
    - somewhat clean separation of HTML from JavaScript
    - allows the handler to be changed at run-time

  - we'll see soon that there is a better way (DOM2 Event Registration)

DOM2 Event Model

- The DOM0 event model was the original method for access in the HTML elements from within JavaScript
- DOM2 added a more sophisticated and powerful method for event handling
  - separates the HTML events from the mouse events
    - HTML events
      - abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
    - mouse events
      - click, mousedown, mousemove, mouseout, mouseover, mouseup

Event Propagation

- When an event occurs, an event object is created at some node in the document tree (*target node*)
- Three-phase process for handling the event
  - capturing phase
    - starting at the document root node, the event propagates down the tree to the target node
    - looks for and executes “enabled” event handlers at any of these nodes
  - target mode phase
    - the event handlers registered for the event at the target node are executed
  - bubbling phase
    - the event “bubbles back up” the tree to the document root node
    - any handers for the event that are encountered are executed

Event Propagation

- Not all events bubble
  - e.g., load and unload events do not bubble; all mouse events do
  - general rule: if it makes sense to handle an event farther up the document tree than the target node, the event will bubble
- Propagation can be explicitly halted

```
event.stopPropagation();
```

- Can explicitly stop the default operation of an element (e.g., submitting a form)

```
event.preventDefault();
```

Registering Event Handlers

- Event handlers can be registered with addEventListener and removed with removeEventListener
  - take the same three parameters
    - name of the event (string literal)
    - name of the handler function
    - Boolean value that indicates whether the event is “enabled” for capturing by other event handlers (for our purposes this will always be false)

```
<input type=“text” id=“custName” />

var cust = document.getElementById(“custName”);
cust.addEventListener(“change”, chkName, false);
```

Accessing the Event

- When an event handler is added this way, the event handling function will receive the Event object as a parameter
  - Event.currentTarget references the object on which the handler is being executed
    - target object during the target node phase
    - other objects during the capturing or bubbling phases
  - Event.target always references the target node

- We can explicitly stop the default operation of an element (e.g., submitting a form) using this event object

```
event.preventDefault();
```

- If the event is a mouse event, a MouseEvent object is provided as the parameter of the event handling function instead
  - same methods and properties as Event
  - adds mouse-specific properties (e.g., clientX, clientY, button, etc.)

Events from Text-Based Elements

- For text-based input elements (input type of text or password, or textarea element) there are four common events that can be handled
  - focus
    - click or tab into the element
  - blur
    - click or tab to something else (out of the element)
  - change
    - change the text in the element
  - select
    - select any text within the element

Validating Form Input

- A common use of JavaScript is to ensure that the values provided on forms are valid
  - doing this server-side requires that the values be sent to the server, checked, and sent back if there is a problem
  - doing this on the client-side makes more sense
    - can be checked at different stages
      - as a field is filled out
      - when the user moves to the next field
      - when the submit button is pressed
    - provides quicker feedback to the user
    - saves on network resources
    - still need to check on the server-side in case the JavaScript validation is circumvented

Validating Form Input

- When an error is detected, there are a few different ways of giving feedback to the user
  - alert() function call, but this is not a good interface feature
  - changing the contents of the HTML document to show where the error is
    - this is a much better approach, since we can design how it will look and work
    - we'll talk about how to do this next week

- When providing an error message, make sure it
  - indicates where the error occurred
  - what the error is
  - what the user can do to fix it

Example

- A simple password typo check
  - a common method to verify whether a user entered some data correctly is to ask them to provide it twice
    - new passwords
    - email addresses
  - the form will have two input elements
  - the data can be checked at two times
    - when the user exits the second input element
    - when the user clicks the submit button
  - see example (using DOM0 event registration)
      - pswd_chk.html
      - pswd_chk.js
      - pswd_chkr.js

Validation Using Regular Expressions

- More complex validation of user input can be done with regular expressions
  - validate a properly formatted phone number

    ```
    /^\d{3}-\d{3}-\d{4}$/
    ```
  - validate the format of a name

    ```
    /^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.?$/
    ```
    - a capital letter, one or more letters, a comma, zero or one space, a capital letter, or more letters, a comma, zero or one space, a capital letter, zero or one period
  - these can be checked with the String.search() method

  - see example (using DOM2 event registration)
      - validator2.html
      - validator2.js
      - validatorr2.js

DOM2 Event Registration is Mandatory

- Even though we have seen an example of using the DOM0 event registration method, you will be expected to use DOM2 in all assignments and exam questions
- Why?
  - separation of JavaScript code from HTML
  - better support for code re-use
  - the event object eliminates the need to access the object on which the event occurred by its id
  - events can be dynamically registered and unregistered

Homework

- Keep up with your reading (Chapters 13-16)

- Next topic: JavaScript & DOM Manipulation

- 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