Intro to DOM and Timers

CSE 190 M (Web Programming) Spring 2008
University of Washington

Reading: Chapter 3 sections 3.2 - 3.3

Lecture Outline

- Introduction to the Document Object Model (DOM)
- JavaScript timers and animation
- Unobtrusive JavaScript

Introduction to the Document Object Model (DOM)

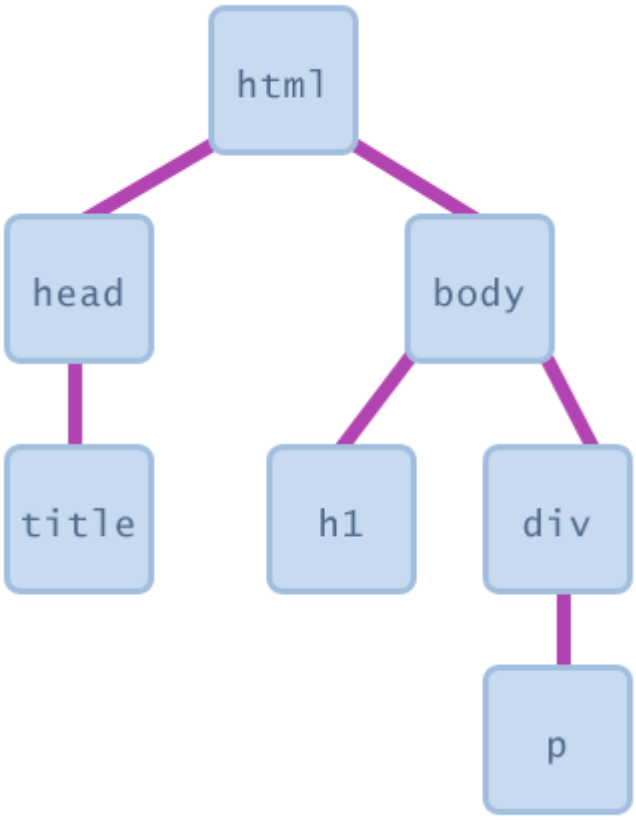
Used to manipulate XHTML page elements in your JS code

Basic idea

- most JS code manipulates elements on an XHTML page
  - example: clicking a button makes text bold
- in this section, we'll learn:
  - how to make our event handlers interact with elements on the page
  - how to *properly* attach event handlers to elements, without modifying the XHTML code (better style)

What is the DOM?

- Document Object Model (**DOM**): a representation of the current page as a set of JS objects
  - e.g. each tag is represented as an object
- we can access these objects in several ways:
  - by traversing the page as a tree-like structure
  - asking for an element's DOM object by its `id`
- script code can view/modify these DOM objects, which causes changes to appear on the web page



Accessing elements: The $ function

```HTML
<span id="sale">Blowout sale!</span>
<button onclick="makeRed();">Make Text Red</button>
```

```JS
function makeRed() {
  $("sale").style.color = "red";
}
```

Blowout sale! Make Text Red

- $ function returns the DOM object for an element with a given `id`
  - $ is actually short for the command `document.getElementById`

More about the $ function

```JS
function $(id) {
  return document.getElementById(id);
}
```

- the `document.getElementById` function returns the DOM object for an element with a given `id`
  - `$("foo") === document.getElementById("foo")`
  - `document` is one of several useful global JS objects we'll see later
- $ is not part of standard JavaScript, but we'll have it in our programs
  - it makes our DOM code much more readable and saves you typing
  - many JavaScript **libraries** define a $ function for convenience
    - later in this course, we'll use a library named Prototype

Manipulating DOM objects

```HTML
<input id="username" type="text" size="12" />
<button onclick="capitalize();">Capitalize It!</button>
```

```JS
function capitalize() {
  $("username").value = $("username").value.toUpperCase();
}
```

- you can get/set most attributes from the XHTML via the DOM object
  - `$("username").type` is `"text"`
  - `$("username").size` is `12`
  - `$("username").value` is whatever value the user has typed
    - `value` exists in most XHTML UI controls (`textarea`, `select`, ...)

More DOM object properties

```JS
<div id="main" class="foo bar">
  <p>Hello, <em>very</em> happy to see you!</p>
</div>
```

- `tagName`: the HTML tag of this element, capitalized
  - `$("main").tagName` is `"DIV"`
  - `$("main").className`: the CSS class(es) of this element, if any is `"foo bar"`
  - `$("main").innerHTML`: the HTML text content inside this element is `"\n  <p>Hello, <em>very</em> happy to see you!</p>\n"`

DOM `style` property

```
<button id="clickme" onclick="enlarge();">
Make me big!</button>
```
HTML

```
function enlarge() {
  $("clickme").style.fontSize = "42pt";
}
```
JS

Make me big!

- `style` property represents the combined CSS styles on this element
- contains identical properties to those set in CSS, but with `namesLikeThis` instead of `names-like-this`
  - examples: `backgroundColor`, `borderLeftWidth`, `fontFamily`

Common DOM styling errors

- many students forget to write `.style` when setting styles

```
$("somediv").color = "red";
$("somediv").style.color = "red";
```
JS

  - our **JSLint** checker will catch this mistake
- style properties are `likeThis`, not `like-this`

```
$("somediv").style.font-size = "14pt";
$("somediv").style.fontSize = "14pt";
```
JS

- style properties must be set as Strings, often with units at the end

```
$("somediv").style.width = 200;
$("somediv").style.width = "200px";
$("somediv").style.padding = "0.5em";
```
JS

  - write what you would have written in the CSS, but in quotes

JavaScript timers and animation

Repeatedly executing an event handler at timed intervals

Timer concepts

- **timer**: executes an action after a delay, or repeatedly at given intervals
- JavaScript's implementation of timers:
  - `setTimeout`, `setInterval`, `clearTimeout`, `clearInterval` functions
  - an event handler function and a delay (ms) are passed as parameters to the above functions
  - the function is called after the delay

Timer functions

- `setTimeout(`*function*`, `*delay*`, [`*param1, param2, ...*`]);`
  arranges to call the given function after the given delay in ms, optionally passing it the parameters provided
- `setInterval(`*function*`, `*delay*`, [`*param1, param2, ...*`]);`
  arranges to call the given function repeatedly, once every *delay* ms
  - both `setTimeout` and `setInterval` return an object representing the timer
- `clearTimeout(`*timer*`);`
  `clearInterval(`*timer*`);`
  stops the given timer object so it will not call its function any more

`setTimeout` example

```JS
function delayMsg() {
  setTimeout(booyah, 5000);
}
function booyah() {   // called when the timer goes off
  alert("Booyah!");
}
```

```HTML
<button onclick="delayMsg();">Click me!</button>
```

Click me!

- `setTimeout` returns instantly; `delayMsg` does not wait for the 5 sec to elapse

`setInterval` example

```JS
function repeatedMessage() {
  setInterval(rudyRudy, 1000);
}
function rudyRudy() {
  alert("Rudy!");
}
```

```HTML
<button onclick="repeatedMessage();">Click me!</button>
```

Click me!

- you may not actually want to click the button ...

Clearing a timer

```JS
var timer;
function repeatedMessage() {
  timer = setInterval(rudyRudy, 1000);
}
function rudyRudy() {
  alert("Rudy!");
}
function cancel() {
  clearInterval(timer);
}
```

```HTML
<button onclick="repeatedMessage();">Rudy chant</button>
<button onclick="cancel();">Make it stop!</button>
```

- `setInterval` returns an object representing the timer
  - can be stored in a global variable
- to cancel the timer, call `clearInterval` and pass the timer object

Passing parameters to timers

```JS
function delayedMultiply() {
  // 6 and 7 are passed to multiply when timer goes off
  var myTimer = setTimeout(multiply, 2000, 6, 7);
}
function multiply(a, b) {
  alert(a * b);
}
```

```HTML
<button onclick="delayedMultiply();">Click me</button>
```

- any parameters after the delay are passed to the timer function
  - (doesn't work in IE6)

Common timer errors

- many students mistakenly write `()` when passing the function

```
~~setTimeout(booyah(), 2000);~~
setTimeout(booyah, 2000);
```
JS

  - what does it actually do if you have the `()` ?

Unobtrusive JavaScript

Adding JavaScript to a page without editing the XHTML

onclick="badStyle();"

```HTML
<button onclick="makeRed();">Make Text Red</button>
```

- placing `onclick` and similar handlers in your XHTML file is actually bad style
- XHTML is for content, not program code or style information
- a page with many handlers becomes cluttered with `onclick`s
- better approach: **unobtrusive JavaScript**
  - the goal: **no JavaScript** in our .html file except the `script` tag link
  - give `id`s to all elements for which we want to handle events
  - attach the handlers to those events *in the JavaScript file itself*

Attaching event handler via DOM

```HTML
<button onclick="makeRed();">Make Text Red</button>
<button id="makeredbutton">Make Text Red</button>
```

```JS
element.event = handlerFunction;
```

```JS
$("makeredbutton").onclick = makeRed;
```

- instead of putting an `onclick` attribute in the XHTML,
  - put an `id` on that same XHTML element
  - in the JS code, grab the DOM object for that element and set its `.onclick` property

A failed attempt

```
// "global" code   (this example does not work!)
$("makeredbutton").onclick = makeRed;

function makeRed() {
  $("sale").style.color = "red";
}
```
JS

- The key question: Where in our JS code do we attach these event handlers?
- We'd like to attach them when the page first loads.
- The "global" area executes *too soon*, because it's in the page's `head`
  - (`body` hasn't been read or created yet by the browser)

The `window.onload` event

```
// "global" code
window.onload = pageLoad;

// runs when the page has completely finished loading
function pageLoad() {
  $("makeredbutton").onclick = makeRed;
}

function makeRed() {
  $("sale").style.color = "red";
}
```
JS

- global `window` object's `onload` event occurs when page is done loading
  - this is exactly when we want to attach our other event handlers

Common unobtrusive JS errors

- many students mistakenly write `()` when attaching the handler

```
~~window.onload = pageLoad();~~
window.onload = pageLoad;

~~$("makeredbutton").onclick = makeRed();~~
$("makeredbutton").onclick = makeRed;
```
JS

  - our **JSLint** checker will catch this mistake
  - what does it actually do if you have the `()` ?
- event names are all lowercase, not capitalized like most variables

```
~~window.onLoad = pageLoad;~~
window.onload = pageLoad;
```
JS

The keyword `this`

```
window.onload = pageLoad;
function pageLoad() {
  $("makeredbutton").onclick = makeRed;
}

function makeRed() {
  this.style.color = "red";
}
```
JS

Make Text Red

- event handlers attached unobtrusively are **bound** to the element
  - doesn't work if you attach it as an `onclick` attribute in the HTML
- inside the handler, the element can refer to itself as `this`
  - also useful when the same handler is shared on multiple elements