Lecture 6: JavaScript Programming Environment

Maharishi University of Management -Fairfield, Iowa © 2016

Outline

- Global DOM Objects -- the global context all js programs run in
- Unobtrusive Javascript -- separation of content (HTML) and behavior (Javascript)
- DOM Element Objects -- the main operatives of js programs
- JS Timers -- needed for today's lab, and common tool for js

The six global DOM objects

- Every JavaScript program can refer to the following global objects:

Name	Description
document	Current HTML page and its content
history	List of pages the user has visited
location	URL of the current HTML page
navigator	Info about the web browser you are using
screen	Info about the screen area occupied by the browser
window	The browser window

The window object

- *the entire browser window; the top-level object in DOM hierarchy*

- technically, all global code and variables become part of the window object
- properties:
  - document, history, location, name
- methods:
  - alert, confirm, prompt (popup boxes)
  - setInterval, setTimeout clearInterval, clearTimeout (timers)
  - open, close (popping up new browser windows)
  - blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo

Popup windows with window.open

```
window.open("http://foo.com/bar.html", "My
  Foo Window",
  "width=900,height=600,scrollbars=1");
```

- window.open pops up a new browser window
- This method is the cause of all the terrible popups on the web!
- some popup blocker software will prevent this method from running

The document object

- *the current web page and the elements inside it*

- properties:
  - anchors, body, cookie, domain, forms, images, links, referrer, title , URL
- methods:
  - getElementById
  - getElementsByName
  - getElementsByTagName
  - close, open, write, writeln
- complete list

The location object

- *the URL of the current web page*

- properties:
  - host, hostname, href, pathname, port, protocol, search
- methods:
  - assign, reload, replace
- complete list

The navigator object

- *information about the web browser application*

- properties:
  - appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent
  - complete list
- Some web programmers examine the navigator object to see what browser is being used, and write browser-specific scripts and hacks:
  - `if (navigator.appName === "Microsoft Internet Explorer") { ...`
  - (this is poor style; you should not need to do this)

The screen object

- *information about the client's display screen*

- properties:
  - availHeight, availWidth, colorDepth, height, pixelDepth, width
  - complete list

The history object

- *the list of sites the browser has visited in this window*

- properties:
  - length
- methods:
  - back, forward, go
- complete list
- sometimes the browser won't let scripts view history properties, for security

Main Point

JavaScript has a set of global DOM objects accessible to every web page. Every JavaScript object runs inside the global window object. The window object has many global functions such as alert and timer methods. At the level of the unified field, an impulse anywhere is an impulse everywhere.

Unobtrusive JavaScript

- JavaScript event code seen yesterday was *obtrusive*, in the HTML; this is bad style
- now we'll see how to write *unobtrusive* JavaScript code
  - HTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions
- allows separation of web site into 3 major categories:
  - **content** (HTML) - what is it?
  - **presentation** (CSS) - how does it look?
  - **behavior** (JavaScript) - how does it respond to user interaction?

Obtrusive event handlers(bad)

```
<button onclick="okayClick();">OK</button>

function okayClick() {
 alert("booyah");
}
```

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body

Unobtrusive JavaScript

```
// where element is a DOM element object
element.onevent = function;

<button id="ok">OK</button>
var okButton = document.getElementById("ok");
okButton.onclick = okayClick;
```

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do **not** put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

Linking to a JavaScript file: script

- JS code can be placed directly in the HTML file's body or head (like CSS)
  - but this is bad style (should separate content, presentation, and behavior)
- script tag should be placed in HTML page's head
- script code should be stored in a separate .js file (like CSS)

```
<script src="example.js" ></script>
```

When does my code run?

```
<html>
 <head>
 <script src="myfile.js"></script> </head>
 <body> ... </body> </html>

// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them
- at this point in time, the browser has not yet read your page's body
  - none of the DOM objects for tags on the page have been created yet

See example: lecture06_examples/runjs.html

A failed attempt at being unobtrusive

```
<html>
 <head>
 <script src="myfile.js" type="text/javascript"></script>
  </head>
 <body> ... </body> </html>
<div><button id="ok">OK</button></div>

// code in myfile.js
document.getElementById("ok").onclick = okayClick; //
  error: cannot set property onclick of null
```

- problem: myfile.js code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM
  - See lecture06_examples/failedattempt.html
- we need a way to attach the handler after the page has loaded...

The window.onload event

```
// this will run once the page has finished loading
function functionName() {
 element.event = functionName;
 element.event = functionName;
 ...
}
window.onload = functionName; // global code
```

- we want to attach our event handlers right after the page is done loading
  - there is a global event called window.onload event that occurs at that moment
- in window.onload handler we attach all the other handlers to run when events occur

See example: lecture06_examples/unobtrusivehandler1.html (does it work?)

Common unobtrusive JS errors

- many students mistakenly write () when attaching the handler

```
window.onload = pageLoad();
window.onload = pageLoad;
okButton.onclick = okayClick();
okButton.onclick = okayClick;
```

  - IMPORTANT FUNDAMENTAL CONCEPT !!!
    - Function reference versus evaluation

- event names are all lowercase, not capitalized like most variables

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

Anonymous functions

```
function(parameters) {
  statements;
}
```

- JavaScript allows you to declare **anonymous functions**
- creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.
- Important in JavaScript because of event handling nature and minimizing namespace clutter

Anonymous function example

```
window.onload = function() {
 var okButton = document.getElementById("ok");
 okButton.onclick = okayClick;
};

function okayClick() {
 alert("booyah");
}
```

or the following is also legal (though harder to read):

```
window.onload = function() {
 var okButton = document.getElementById("ok");
 okButton.onclick = function() {
   alert("booyah");
 };
};
```

See example: lecture06_examples/anonymous.html

Main Points

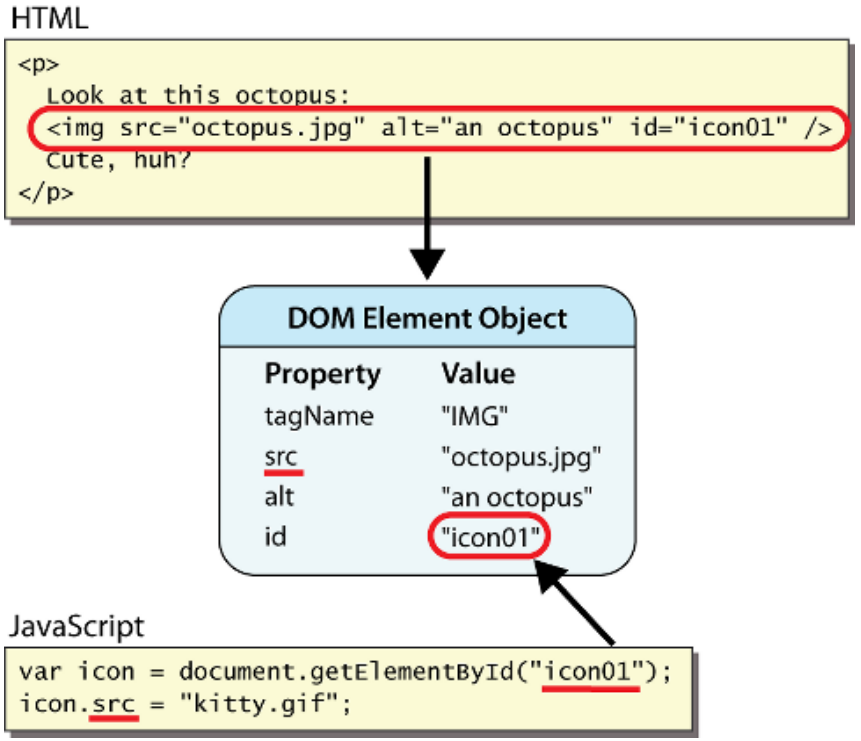
- Unobtrusive JavaScript promotes separation of web page content into 3 different concerns: content (HTML), presentation (CSS), and behavior(JS) (ala MVC, knower, known, process of knowing)

- JavaScript code runs when the page loads it. Event handlers cannot be assigned until the target elements are loaded. Event handlers often use anonymous functions. Creative intelligence proceeds in an orderly sequential manner.

Recall: DOM element objects

- every element on the page has a corresponding DOM object

- access/modify the attributes of the DOM object with *objectName.attributeName*



DOM object properties

```
<div id="main" class="foo bar">
  <p>Hello, <em>very</em> happy to see you!</p>
  <img id="icon" src="greatwall.jpg" alt="GREAT WALL" />
</div>
var mainDiv = document.getElementById("main");
var icon = document.getElementById("icon");
```

See example: lecture06_examples/objectproperties.html

Property	Description	Example
tagName	element's HTML tag	mainDiv.tagName is "DIV"
className	CSS classes of element	mainDiv.className is "foo bar"
innerHTML	Content in element	mainDiv.innerHTML is "\n <p>Hello…
src	URL target of an image	icon.src is "greatwall.jpg"

DOM properties for form controls

```
<input id="sid" type="text" size="7" maxlength="7"
  />
<input id="frosh" type="checkbox" checked="checked"
  /> Freshman?
var sid = document.getElementById("sid");
var frosh = document.getElementById("frosh");
```

Property	Description	Example
value	the text/value chosen by the user	sid.value could be "1234567"
checked	whether a box is checked	frosh.checked is true
disabled	whether a control is disabled (boolean)	frosh.disabled is false
readOnly	whether a text box is read-only	sid.readOnly is false

More about from controls

```
<select id="captain">
  <option value="kirk">James T. Kirk</option>
  <option value="picard">Jean-Luc Picard</option>
  <option value="cisco">Benjamin Cisco</option>
</select>
<label>
  <input id="trekkie" type="checkbox" /> I'm a
   Trekkie
</label>
```

- when talking to a text box or select, you usually want its value
- when talking to a checkbox or radio button, you probably want to know if it's checked (true/false)

Abuse of innerHTML

```
// bad style!
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "<p>text and <a
  href="page.html">link</a>";
```

- innerHTML can inject arbitrary HTML content into the page
- however, this is prone to bugs and errors and is considered poor style
  - innerHTML not part of DOM standard
  - issues involving DOM rebuilding tree and maintaining node and event handler references
- Best practice: inject plain text only
  - Do not use innerHTML to inject HTML tags;

DOM elements style property

```
<button id="clickme">Color Me</button>
window.onload = function() {
  document.getElementById("clickme").onclick =
  changeColor;
};
function changeColor() {
 var clickMe = document.getElementById("clickme");
 clickMe.style.color = "red";
}
```

Property	Description
style	lets you set any CSS style property for an element

- contains same properties as in CSS, but with camelCasedNames
  - examples: backgroundColor, borderLeftWidth, fontFamily

Common DOM styling errors

- many students forget to write .style when setting styles

```
var clickMe = document.getElementById("clickme");
clickMe.color = "red";
clickMe.style.color = "red";
```

- style properties are capitalized likeThis, not like-this

```
clickMe.style.font-size = "14pt";
clickMe.style.fontSize = "14pt";
```

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

```
clickMe.style.width = 200;
clickMe.style.width = "200px";
clickMe.style.padding = "0.5em";
```

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

Unobtrusive styling

```
function okayClick() {
  this.style.color = "red";
  this.className = "highlighted";
}
.highlighted { color: red; }
```

- well-written JavaScript code should contain as little CSS as possible
- use JS to set CSS classes/IDs on elements
- define the styles of those classes/IDs in your CSS file
- What about cssZenGarden?

Getting/Setting CSS classes

```
function highlightField() {
 // turn text yellow and make it bigger
 if (!document.getElementById("text").className) {
  document.getElementById("text").className = "highlight";
 } else if
  (document.getElementById("text").className.indexOf("inva
  lid") < 0) {
   document.getElementById("text").className += "
  highlight";
 }
}
```

- JS DOM's className property corresponds to HTML class attribute
- somewhat clunky when dealing with multiple space-separated classes as one big string

Problems with reading/changing styles

```
<button id="clickme">Click Me</button>
window.onload = function() {
  document.getElementById("clickme").onclick = biggerFont;
};
function biggerFont() {
 var size =
  parseInt(document.getElementById("clickme").style.fontSi
  ze);
  size += 4;
  document.getElementById("clickMe").style.fontSize = size
  + "pt";
}
```

- style property lets you set any CSS style for an element
- problem(??): you cannot (sometimes) read existing styles with it
  - E.g., older browsers when fontSize is inherited
- See example: lecture06_examples/style.html

Common bug: incorrect usage of existing styles

```
document.getElementById("main").style.top =
document.getElementById("main").style.top + 100 +
  "px"; // bad
```

- the above example computes e.g. "200px" + 100 + "px" , which would evaluate to "200px100px"

- a corrected version:

```
document.getElementById("main").style.top =
  parseInt(document.getElementById("main").style.top
  ) + 100 + "px"; // correct
```

Main Points

- The DOM is an API so the JavaScript programmer can conveniently access and manipulate the HTML elements in code.

- **Science of Consciousness:** The TM and TM-Sidhi programs are techniques that allow anyone to conveniently contact and act at the level of the Unified Field.

Timer events

method	description
setTimeout(*function*, *delayMS*);	arranges to call given function after given delay in ms
setInterval(*function*, *delayMS*);	arranges to call function repeatedly every *delayMS* ms
clearTimeout(*timerID*); clearInterval(*timerID*);	stops the given timer so it will not call its function

- both setTimeout and setInterval return an ID representing the timer
  - this ID can be passed to clearTimeout/Interval later to stop the timer

Asynchronous & Callbacks

- A callback function is a function you give to another function, to be invoked later when the other function is finished (desired).

- All callback functions do not execute immediately (even when called), instead they are queued in the browser event queue.
  - How JavaScript deals with being single threaded

setTimeout Example

```
<button onclick="delayMsg();">Click me!</button>
<span id="output"></span>

function delayMsg() {
  setTimeout(booyah, 5000);
  document.getElementById("output").innerHTML =
  "Wait for it...";
}
function booyah() {
 // called when the timer goes off
  document.getElementById("output").innerHTML =
  "BOOYAH!";
}
```

setInterval example

```
timer = null; // stores ID of interval timer
function delayMsg2() {
 if (timer === null) {
      timer = setInterval(rudy, 1000);
  } else {
      clearInterval(timer);
      timer = null;
 }
}

function rudy() { // called each time the timer goes
  off
 document.getElementById("output").innerHTML += "
  Rudy!";
}
```

Passing parameters to timers

```
function delayedMultiply() {
 // 6 and 7 are passed to multiply when
  timer goes off
 setTimeout(multiply, 4000, 6, 7);
}
function multiply(a, b) {
 alert(a * b);
}
```

- any parameters after the delay are eventually passed to the timer function
- why not just write this? setTimeout(**multiply(6, 7)**, 2000);

Common timer errors

- many students mistakenly write () when passing the function

```
setTimeout(booyah(), 2000);
setTimeout(booyah, 2000);
setTimeout(multiply(num1 * num2), 2000);
setTimeout(multiply, 2000, num1, num2);
```

- what does it actually do if you have the () ?
  - it calls the function immediately, rather than waiting the 2000ms!
- IMPORTANT!!!

First-class functions

- Functions can be assigned to variables

```
var myfunc = function(a, x) {
 return a * b;
};
```

- Functions can be passed as parameters

```
function apply(a, b, f) {
 return f(a, b);
}
var x = apply(2, 3, myfunc); // 6
```

- Functions can be return values

```
function getAlert(str) {
 return function() { alert(str); }
}
var whatsUpAlert= getAlert("What's up!");
whatsUpAlert(); // "What's up!"
```

JavaScript “strict” mode

```
"use strict"; your code...

(function() {
 "use strict";
 your code...
})();
```

- writing "use strict"; at the very top of your JS file turns on strict syntax checking:
  - shows an error if you try to assign to an undeclared variable
  - stops you from overwriting key JS system libraries
  - forbids some unsafe or error-prone language features
- You should always turn on strict mode for your code in this class
- IIFE syntax will be discussed further in next lesson