Web Programming Step by Step

Lecture 15
Unobtrusive JavaScript
Reading: 8.1 - 8.3

Except where otherwise noted, the contents of this presentation are Copyright 2009 Marty Stepp and Jessica Miller.

8.1: Global DOM Objects

- 8.1: Global DOM Objects
- 8.2: DOM Element Objects
- 8.3: The DOM Tree
The six global DOM objects

Every Javascript program can refer to the following global objects:

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>current HTML page and its content</td>
</tr>
<tr>
<td>history</td>
<td>list of pages the user has visited</td>
</tr>
<tr>
<td>location</td>
<td>URL of the current HTML page</td>
</tr>
<tr>
<td>navigator</td>
<td>info about the web browser you are using</td>
</tr>
<tr>
<td>screen</td>
<td>info about the screen area occupied by the browser</td>
</tr>
<tr>
<td>window</td>
<td>the browser window</td>
</tr>
</tbody>
</table>

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
The **document** object

*the current web page and the elements inside it*

- properties:
  - anchors, body, cookie, domain, forms, images, links, referrer, title, URL
- methods:
  - getElementsById
  - 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:

```javascript
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

---

**Unobtrusive JavaScript (8.1.1)**

- JavaScript event code seen previously 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)

 threaten event handlers (bad)

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

// called when OK button is clicked
function okayClick() {
  alert("booyah");
}

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

Attaching an event handler in JavaScript code

- 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?
When does my code run?

• 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

A failed attempt at being unobtrusive

• problem: global 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
• we need a way to attach the handler after the page has loaded...
The **window.onload** event (8.1.1)

```javascript
// 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

---

An unobtrusive event handler

```html
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>

// called when page loads; sets up event handlers
function pageLoad() {
    $('#ok').onclick = okayClick;
}

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

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

OK
Common unobtrusive JS errors

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

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

- our JSLint checker will catch this mistake
- event names are all lowercase, not capitalized like most variables

```javascript
window.onload = pageLoad;
window.onload = pageLoad;
```

Anonymous functions (8.1.2)

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

- JavaScript allows you to declare anonymous functions
- quickly creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.
Anonymous function example

```javascript
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 and bad style):

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

The keyword **this** (8.1.3)

```javascript
this.fieldName // access field
this.fieldName = value; // modify field
this.methodName(parameters); // call method
```

- all JavaScript code actually runs inside of an object
- by default, code runs inside the global `window` object
  - all global variables and functions you declare become part of `window`
- the `this` keyword refers to the current object
The keyword `this` (8.1.3)

function pageLoad() {
    $\text{"ok"}.onclick = okayClick;  \quad \text{// bound to okButton here}
}

function okayClick() {
    \text{this}.innerHTML = \text{"booyah"}; \quad \text{// it was called on}
}

window.onload = pageLoad;

- event handlers attached unobtrusively are `bound` to the element
- inside the handler, that element becomes `this` (rather than the `window`)

Fixing redundant code with `this`

```html
<fieldset>
    <label><input type="radio" name="ducks" value="Huey" /> Huey</label>
    <label><input type="radio" name="ducks" value="Dewey" /> Dewey</label>
    <label><input type="radio" name="ducks" value="Louie" /> Louie</label>
</fieldset>
```

```js
function processDucks() {
    if ($("huey").checked) {
        alert("Huey is checked!");
    } else if ($("dewey").checked) {
        alert("Dewey is checked!");
    } else {
        alert("Louie is checked!");
    }
    alert(\text{this}.value + \text{" is checked!"});
}
```

- if the same function is assigned to multiple elements, each gets its own bound copy