Unobtrusive JavaScript

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

Lecture Outline

- unobtrusive JavaScript
  - writing web pages where there is no JS code in your XHTML body
- Accessing groups of DOM elements
  - processing groups of DOM element objects for events or styling purposes

Unobtrusive JavaScript

writing web pages where there is no JS code in your XHTML body

Unobtrusive JavaScript idea

- JavaScript event code seen previously was *obtrusive* (in the XHTML)
  - this is bad style (mixes content and behavior)
- now we'll see how to write *unobtrusive* JavaScript code
  - XHTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions
  - clean XHTML code, clear separation of content, presentation, behavior

Obtrusive event handlers (bad)

```
<body>
  <button id="ok" onclick="okayClick();">Click me</button>
  ...
```
HTML

```
// called when OK button is clicked
function okayClick() {
  $("ok").style.color = "red";
}
```
JS

Click me

- this is considered bad style in modern web programming (HTML is cluttered with JavaScript calls)

Attaching an event handler in JavaScript code

```JS
element.event = functionName;
```

```JS
$("ok").onclick = okayClick;
```

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
- this is better style than attaching them in the XHTML
- Where should we put the above code?

A failed attempt at being unobtrusive

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

```JS
// global code
$("ok").onclick = okayClick;   // error: $("ok") is undefined
```

- 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 just as the page finishes loading

Browser/page events

- `onerror` : an error occurs when loading a document or an image
- **`onload`** : the browser loads the page
- `onresize` : the browser window is resized
- `onunload` : the browser exits the page

- generally handlers for these are attached to the global `window` object or the document's `body`

The `window.onload` event

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

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

- we want to attach our event handlers right after the page is done loading
  - this is exactly when the `window.onload` event occurs, so we'll handle that event
- in `window.onload` handler we attach all the other handlers, which in turn run when those controls are interacted with

An unobtrusive event handler

```
<body>
  <button id="ok">Click me</button>
...
```
HTML

```
window.onload = pageLoad;

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

function okayClick() {
  $("ok").style.color = "red";
}
```
JS

Click me

Why is unobtrusive JavaScript better?

- allows separation of web site into 3 major categories:
  - **content** (XHTML) - what is it?
  - **presentation** (CSS) - how does it look?
  - **behavior** (JavaScript) - how does it respond to user interaction?
- page isn't cluttered with event code or stylistic information

Common unobtrusive JS errors

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

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

~~$("ok").onclick = okayClick();~~
$("ok").onclick = okayClick;
```
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() {
  $("ok").onclick = okayClick;   // bound to $("ok") here
}

function okayClick() {           // okayClick knows what DOM object
  this.style.color = "red";      // it was called on
}
```
JS

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

Fixing redundant code with `this`

```HTML
<fieldset>
  <label><input id="Huey" type="radio" name="ducks" /> Huey</label>
  <label><input id="Dewey" type="radio" name="ducks" /> Dewey</label>
  <label><input id="Louie" type="radio" name="ducks" /> 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(this.id + " is checked!");
}
```

Anonymous functions

```JS
function(parameters) {
  the function's code;
}
```

- sometimes you want to quickly create a function without giving it a name or explicit declaration
- JavaScript allows you to declare **anonymous functions**
- an anonymous function can be stored as a variable, attached to an event handler, etc.

Anonymous function example

```JS
window.onload = function() {
  $("ok").onclick = okayClick;
};

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

Click me

or, the following is even legal (though harder to read and bad style):

```JS
window.onload = function() {
  $("ok").onclick = function() {
    this.style.color = "red";
  };
};
```

Unobtrusive styling

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

```CSS
.highlighted { color: red; }
```

- well-styled JavaScript code should contain as little CSS as possible
- whenever you can, you should instead use JS to set CSS classes/IDs on elements, and then define the styles of those classes/IDs in your CSS file
- Prototype methods for setting CSS classes:
  - `addClassName`, `classNames`, `hasClassName`, `removeClassName`
- non-Prototype way of dealing with classes/IDs:
  `className`, `id` properties

Accessing groups of DOM elements

processing groups of DOM element objects for events or styling purposes

Motivation for grouping DOM elements

How would we do each of the following in our JavaScript code?

- When the Go button is clicked, reposition all the `div`s of class `puzzle` to random x/y locations.
- When the user hovers over the maze boundary, turn all maze walls red.
- Change every other item in the `ul` list with `id` of `TAs` to have a gray background.

Each task involves modifying a group of elements to have a common new feature or style...

Accessing DOM element objects

methods in `document` object for getting DOM elements (* = Prototype):

- `document.getElementById` (a.k.a. `$` *) : DOM element that uses the given `id`
- `document.getElementsByTagName` :
  returns array of DOM elements with the given XHTML element, such as `"div"`
- `document.getElementsByName` :
  returns array of DOM elements with given `name` attribute (e.g. radio buttons in a group)
- `document.getElementsByClassName` * :
  returns array of DOM elements that use the given `class` attribute
- `document.getElementsBySelector` * (a.k.a. `$$` *) :
  returns array of DOM elements that match the given CSS selector string, such as `"div#sidebar ul.news > li"`

Getting all elements of a certain type

highlight all paragraphs in document

```JS
var allParas = document.getElementsByTagName("p");
for (var i = 0; i < allParas.length; i++) {
  allParas[i].style.backgroundColor = "yellow";
}
```

```HTML
<body>
  <p>This is the first paragraph</p>
  <p>This is the second paragraph</p>
  <p>You get the idea...</p>
</body>
```

Combining with $

highlight all paragraphs inside of the section with ID `"footer"`

```JS
var footerParas = $("footer").getElementsByTagName("p");
for (var i = 0; i < footerParas.length; i++) {
  footerParas[i].style.backgroundColor = "yellow";
}
```

```HTML
<p>This won't be returned!</p>
<div id="footer">
  <p>1234 Street</p>
  <p>Atlanta, GA</p>
</div>
```

Simplifying things with Prototype

highlight all paragraphs inside of the section with ID `"footer"`

```
var footerParas = $$("#footer p");
for (var i = 0; i < footerParas.length; i++) {
  footerParas[i].style.backgroundColor = "yellow";
}
```
JS

- Prototype's `$$` function will return the array of DOM elements that matches *any* CSS selector
- this is a very powerful way to select exactly the elements on the page that you want

$$ and event handlers

listen to clicks on all `button`s with class `control` directly inside of the section with ID `"game"`

```
window.onload = function() {
  var gameButtons = $$("#game > button.control");
  for (var i = 0; i < gameButtons.length; i++) {
    gameButtons[i].onclick = gameButtonClick;
  }
};

function gameButtonClick() {
  ...
}
```
JS

- you can use `$$` and other DOM walking methods to unobtrusively attach event handlers to a group of related elements in your `window.onload` code

Common $$ errors

- many students forget to write `.` or # in front of a `class` / `id`

```
var gameButtons = $$("control");
var gameButtons = $$(".control");
```
JS

- `$$` returns an array, not a single element; must loop over the results and process each one

```
$$(".control").style.color = "red";
var gameButtons = $$(".control");
for (var i = 0; i < gameButtons.length; i++) {
  gameButtons[i].style.color = "red";
}
```
JS

- Common question: Yes, you can select a group of elements using `$$` even if your CSS file has no style rule for that same group

Combining with `$` : `Element.select`

select all `button`s with class `control` directly inside of the section with ID `"game"`

```
var gameArea = $("game");
var gameButtons = gameArea.select("button.control");
for (var i = 0; i < footerParas.length; i++) {
  gameButtons[i].style.color = "yellow";
}
```
JS

- the `select` method returns an array of DOM element objects matching a given CSS selector within a particular root element
  - much like `$$`, but only within part of the page
- the above code grabs all buttons with class of `"control"` that are inside the element with `id` of `"game"`