Javascript Events

CSE 190 M (Web Programming), Spring 2007
University of Washington

Reading: Sebesta Ch. 5 sections 5.2, 5.4.2, 5.7 - 5.8.3

Lecture summary

- in these slides, we'll see how to respond to several events
- Javascript event code seen previously was *obtrusive* (in the HTML)
  - this is bad style (mixes content and behavior)
- this week we'll see how to write *unobtrusive* Javascript code
  - HTML with minimal Javascript inside
  - uses the DOM to attach and execute all Javascript functions
  - one drawback: significant browser compatibilities

Old-style event handlers

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

// called when OK button is clicked
function okay() {
    var button = document.getElementById("ok");
    button.style.color = "red";
}
```

Click me

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

Event handlers using DOM

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

window.onload = initializeBody;   // global code

// called when page loads; sets up all event handlers
function initializeBody() {
    document.getElementById("ok").onclick = okay;
}
function okay() {
    this.style.color = "red";
}
```

Click me

Why is unobtrusive Javascript better?

- 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?
- page isn't cluttered with event code or stylistic information

The `window.onload` event

```
window.onload = name;

// called when page loads; sets up all event handlers
function name() {
    set up any necessary event handlers using the DOM
}
```

- `window` is one of several global DOM objects
- its `onload` event property represents the loading of the web page
  - we assign a function to run when the page loads (no parentheses)
  - that function can do all attachment of event handlers to page elements

Proper event handlers

```
window.onload = initializeBody;

function initializeBody() {
    var name = document.getElementById("ok");
    name.event = function;
    ...
}
```

- to set up an event handler, get the element's DOM object and set the appropriate event property to the function to call (no parentheses)

Mouse events

DOM objects for HTML elements have the following properties:

- clicking
  - `onmousedown` : user presses down mouse button on this element
  - `onmouseup` : user releases mouse button on this element
  - `onclick` : user presses/releases mouse button on this element
  - `ondblclick` : user presses/releases mouse button *twice* on this element
- movement
  - `onmouseover` : mouse cursor enters this element's box
  - `onmouseout` : mouse cursor exits this element's box
  - `onmousemove` : mouse cursor moves within this element's box

```
myElement.onmousemove = myFunction;
```

Mouse event example

```
<div id="dare">Click me ... I dare you!</div>

function initializeBody() {
    document.getElementById("dare").onmousedown = colorIt;
}
function colorIt() {
    this.style.backgroundColor = "red";
}
```

Click me ... I dare you!

- in event handler, element can refer to itself using keyword `this` (don't have to call `document.getElementById` again)

Multiple mouse event variation

```
<div id="dare">Click me ... I dare you!</div>

function initializeBody() {
    var dareDiv = document.getElementById("dare");
    dareDiv.onmousedown = colorIt;
    dareDiv.onmouseup = uncolorIt;
}
function colorIt() {
        this.style.backgroundColor = "red";
}
function uncolorIt() {
        this.style.backgroundColor = "transparent";
}
```

Click me ... I dare you!

Examining the mouse event

```
function colorIt(event) {
    this.style.backgroundColor = "red";
    this.innerHTML = "You clicked (" + event.screenX +
        ", " + event.screenY + ")");
}
```

Click me ... I dare you!

- a handler can accept an optional parameter representing the event
- this object holds several properties about the event that occurred

Event object properties (reference)

- `type` : what kind of event, such as `"click"` or `"mousedown"`
  - same as event property name without `on` prefix
  - useful if you use the same handler to handle multiple events
- `clientX`, `clientY` : coordinates from top/left of *page*
- `screenX`, `screenY` : coordinates from top/left of *screen*

Browser incompatibilities: events

- fuzzy W3C specs and browser wars have led to event differences between browsers
- IE6 sucks and doesn't support accepting `event` as a parameter
  - instead uses non-standard property `window.event`
  - some properties inside this object are non-standard
- even mighty Firefox is missing some standard properties (gasp!)
- a cross-browser script can handle both

Browser incompatibilities: properties

- `offsetX`, `offsetY` : coordinates from top/left of *element*
  - Firefox uses non-standard `layerX`, `layerY` properties instead
- `button` : which mouse button was pressed/released, if any
  - IE returns 1/2/4 for left/right/middle button; Firefox returns 0/1/2 (standard)
  - Firefox also uses non-standard `which` property instead
- `srcElement` : element that fired the event
  - Firefox uses non-standard `target` property instead
- more incompatibilities

Click me: Which properties are supported?

One workaround for incompatibilities

```
function handleClick(event) {
    event = standardizeEvent(event);
    ...
}

// Repairs various browser incompatibilities.
function standardizeEvent(event) {
    var e = event || window.event;
    e.srcElement = e.srcElement || e.target;
    return e;
}
```

- many web developers write a function to fix event incompatibilities
- NOT required for this course (assume Firefox)

Practice problem: Draggable map

One of the coolest features of Google Maps is the ability to drag the map to move it around. Write a program with a draggable map of Middle Earth using Javascript mouse event handlers. (See the background CSS properties from the end of the CSS slides.)

Event nesting

```
<div id="gum"><img src="gum.png" alt="gum" />
Double your pleasure, <em>double your fun</em>!</div>

function initializeBody() {
    document.getElementById("gum").ondblclick = doBorder;
}
function doBorder(event) {
    event = standardizeEvent(event);
    event.srcElement.style.border = "2px dashed blue";
}
```

Double your pleasure, *double your fun*!

- placing event handler on the `div` intercepts events on anything inside it
- when multiple event handlers overlap, capture/bubble rules apply

----- Other events -----

Other events

Keyboard events

DOM objects for HTML elements have the following properties:

- `onkeydown` : user presses a key while this element has keyboard focus
- `onkeyup` : user releases a key while this element has keyboard focus
- `onkeypress` : user presses and releases a key while this element has keyboard focus
- `onfocus` : this element gains keyboard focus
- `onblur` : this element loses keyboard focus

Key event object properties

- `keyCode`: ASCII value of key pressed
  - convert to letter: `String.fromCharCode(event.keyCode)`
  - list of key values
- `altKey` : `true` if Alt key is being held
- `ctrlKey` : `true` if Ctrl key is being held
- `shiftKey` : `true` if Shift key is being held

Which key event properties does your browser support?

Text box events

these are supported by `<input type="text">`, `<textarea>`

- `onselect` : text within a text box is selected
- `onchange` : content of a text box changes

Practice problem: numbers only

Write Javascript code so that a text input box will only accept numbers as input. Any other characters typed will be removed from the box immediately.

Problem: multiple scripts

```
// script 1
window.onload = init1;

// script 2
window.onload = init2;
```

- some HTML pages link to multiple scripts that use `window.onload`
- problem: each overrides the other's `window.onload` handler
  - last script loaded "wins"; others fail to work

Using `addEventListener`

```
// script 1
window.addEventListener("load", init1, false);

// script 2
window.addEventListener("load", init2, false);
```

- solution: one event can have multiple handlers
- W3C standard method `addEventListener` achieves this

  ```
  element.addEventListener("event", function, false);
  ```

- IE6 sucks and requires non-standard `attachEvent` instead
  - a script can work in both by testing for which one to use

Browser/page events

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

- generally handlers for these are attached to the `window` object