Javascript: DOM Events

Some of these notes courtesy of M. Hall coreservlets.com. Used by permission

Event Handling

Most HTML elements have onXXX attributes

- Designates function to run when specified event occurs

General event handlers

- `onclick, ondblclick`
  - User single-clicks or double-clicks over element
- `onkeydown, onkeypress, onkeyup`
  - User presses key when element has the focus
- `onmousedown, onmouseup`
  - User presses mouse over element
- `onmouseover, onmouseout`
  - Mouse moves over or leaves being over element
- `onmousemove`
  - Mouse moves while over element

***HTML5 is introducing new touch-oriented events* for mobile devices**

Event Handlers (*general-events.html*)

```
<head><title>General Event Handlers</title>
<script src=”general-events.js” type="text/javascript"></script>
</head>
<body>
<h2 onclick="alert('Ouch!')">
Here is a heading.
What happens when you click on it?
</h2>
<h2 onmouseover="on()" onmouseout="off()">
Here is a heading.
<span id="messageRegion"></span>
What happens when you move over it?
</h2>

function on() {   // general-events.js
  var span = document.getElementById("messageRegion");
  span.innerHTML =
    "<br/><font color='red'>Wow!</font><br/>";
}
function off() {
  var span = document.getElementById("messageRegion");
  span.innerHTML = "";
}
```

Specialized Event Handlers

input

- `onclick`
  - For pushbuttons and toggle buttons. Also fires when button is invoked via keyboard.
- `onchange`
  - For textfields, when change is committed (focus leaves)
    - Use `onkeyup` for individual characters
- `onblur, onfocus`

form

- `onsubmit, onreset`
  - Return false to prevent form from really being submitted. Widely used for client-side validation of form fields.

body

- `onblur, onerror, onfocus, onload, onresize, onunload`

img

- `onabort, onerror, onload`

Page Load Events

When an HTML page loads…

- The browser's parser executes each script as it encounters it
- Couple problems with this default behavior:

1. What if the script does something computationally expensive?
   Problem here is the page will pause rendering, creating an awkward UX for the end user
2. What if the script performs DOM manipulation as a side-effect?
   The script will have a race condition between the parser/renderer and the Javascript engine
   - Best practice was to put your scripts at the end of the page. But even then some events will not be handled until that Javascript loads

See JSLoadRace1.html, JSLoadRace1Sleepy.html, JSLoadRace2.html (server), JSLoadRace2Sleepy.html

Page Load Events

See JSLoadRace3*.html examples

Event Handlers for the Load Sequence

1. <Body **onload**="*code*" …>
   - Executes *code* after the parser completes loading the *page and all resources on the page*
2. <script src="*URL-to-js*" **defer** />
   - "defer" is a boolean attribute that ensures an external script will not execute (even though it may load) until the page has been parsed
3. <script src="*URL-to-js*" **async** />
   - "async" is like "defer" in that it is a Boolean attribute that allows the parser to continue while a script is fetched
   - But a little different in that the script is executed as soon as loaded
   - Addresses concern #1 but not #2, but is faster than defer
4. Add an event listener to the **DOMContentLoaded** event
   - This event fires on the document object when the DOM is available in its entirety to the Javascript runtime.
   - A subtle difference from "defer"; it fires after parsing but before rendering may be completed (all resources retrieved and transformed).

Event Listeners

Pre-HTML5:

- You attached event handlers to objects directly by manipulating a property on the object
- Side effect was that if you wanted multiple independent behaviors to execute you still had to roll them into one encapsulating function
- “onclick” from our buttons, for example

HTML5:

- Add and remove listeners dynamically, programmatically
- addEventListener(eventType, eventHandler, useCapture flag)
  - eventType depends on the widget but could be ‘click’
  - eventHandler is an object, usually a function, that optionally takes an event object as a param, and does something
  - useCapture flag – a true/false that you should set to false

See JSLoadRace3DOMReplaceDeferEl.html and JSLoadRace3DOMContentLoadedEl.html

Summary on events

This talk presumes you have worked with events before in some other programming model, such as

- Systems programming where an event comes in from a sensor
- NodeJS programming (remember those event emitters?)
- GUI programming
- Lifecycle management on the state of an object or component

- Elements of the last 2 are present here
  - *GUI events* on HTML5 UI widgets
  - *Page load events* during the lifecycle of the page load/parse/render that happens in the browser (recall Garsiel's paper).
- The DOM itself is consider the state of the UI
  - That is, both types of events above can manipulate it
  - You can in fact hide additional types of state in the DOM, but we have better mechanisms for that, which we will discuss next.