

Yet More Javascript

CS174

Chris Pollett

Sep 26, 2007.

Outline

- Element Access in Javascript
- Events and Event Handling
- Validating Forms
- DOM 2
- Element Positioning
- Moving Elements
- Element Visibility

Element Access in Javascript

- The DOM 0 way of associating Javascript objects with form elements was to use the *forms* array of the Document Object, as well as its sub *elements* array.
- For example, if there was only one form on a page with the following button on it:

```
<input type="button" name="turnItOn" />
```

then it could be accessed with `document.forms[0].elements[0]`.
- This is awkward if we start to add elements as this address could change. Instead, we could do `document.forms[0].turniton`
- Still, having `forms[0]` is awkward. Another approach is to give this button an id:

```
<input type="button" id="turnItOn" name="turnItOn"/>
```

We could then look up the button using:

```
button = document.getElementById("turnItOn");
```
- We need the attribute *name* still for the button above because that is what gets sent to the server as the name/value pair when the form is submitted.

More on Element Access

- Checkboxes in a group of checkboxes often share the same name.
- Radio buttons in a group of radio buttons *always* share the same name.
- In the DOM, these object get reflected as an array rather than an element of that name.
- For example suppose we had a radio button group with name vehicles. on a form with id bob. We could access this by doing

```
myForm = document.getElementById("bob");
numButtons = myForm.vehicles.length;
// if we want to we could cycle over this array for values.
for( i =0 ; i < numButtons; i++)
{
    oneVehicle = myForm.vehicles[i].
    // do something
}
```

Events and Event Handling

- In XHTML, there are a collection of attributes beginning with “on” that can be used to give a handler for different kinds of events:
 - onblur -- can be used in <a>, <button>, <input>, <textarea>, <select>
 - onchange -- can be used in <input>, <textarea>, <select>
 - onclick -- can be used in <a>, <input>
 - onfocus -- can be used in <a>, <input>, <textarea>, <select>
 - onload/ onunload -- can be used in <body>
 - onmousedown, onmousemove, onmouseout, onmouseover, onmouseup -- can be used in most elements.
 - onselect -- can be used in <input>, <textarea>
- An example, of using one of these is:
`<input type = “button” id=“b” name=“b” onclick=“alert(‘b tapped’);” />`

More on Event Handling

- Besides the method of the last slide to register a handler for an event, you can also set the handler from Javascript:

```
document.getElementById("b").onclick =  
    myNewHandler;
```

- We can set the values of other form elements within Javascript and we can also generate events within Javascript.

```
<input type="text" id="cost"  
    onfocus="this.value=10; this.blur();" />
```

Validating Forms

- As we said before it useful to check that the data entered into a form is valid before sending it to the server.
- We saw last class we can associate a handler with a form using a syntax-like:

```
<form ... onsubmit="return checkSubmit()" >
```

- If this function returns true; the form is submitted; otherwise it is not.
- Typically, if the form is not correct you want to display some message.
 - You could either use an alert
 - Or you could set the style of some element from `display: none` to `display: inline` or `display: block`. This would allow you to put stars by missing data:

```
phoneStar = document.getElementById("phoneStar");  
phoneStar.style="display:inline";
```
 - One can also dynamically insert content such as an error message into div tags by setting its `innerHTML` attribute.
 - Finally, if only one element is missing in the form it doesn't to generate `focus()` and `select()` events on it.

DOM 2

- The DOM 2 Event model is supported by Firefox and other recent browsers but is not supported by IE6.
- In this model events are split into HTMLEvents like blur and MouseEvents like click and all events beginning with mouse.
- Event registration is done using addEventListener:
document.myelement.addEventListener(“change”, myhandler, false).
- You can add multiple handlers to the same node.
- There is also a removeEventListener method.
- Events are targeted on some node in the doc tree. In the event *capturing* phase, one starts at the root node and goes down the tree toward the target calling all events whose onCapture flag is true. When the target node is reached a DOM 0 like event handling is done for handlers that are specifically attached to this target. Then a *bubbling* phase back up the tree happens and all listeners whose onCapture flag is false are called.

Whether called incapture or bubbling phase

The Navigator Object

- The navigator object can be used to determine browser type.
- It has two useful properties, `appName` and `appVersion`

Element Positioning

- Positioning of elements can be done using styles.
- It come in two types absolute and relative:

```
<p style="position: absolute; left:100px; top: 200px">some text  
</p>  
<p style="position: relative; left:10px; top: -20px; width: 50">some  
  other text  
</p>
```
- Both positionings can be used to set text on top of other text.
- Since you can change the style property of an element in Javascript, you can use positioning to move objects around on the screen dynamically.

Moving Elements

- Consider the following Javascript function:

```
function moveIt(id, newTop, newLeft)
{
    myStyle = document.getElementById(id).style;
    myStyle.top = newTop + "px"; /* notice how CSS properties are
        properties of the style object*/
    myStyle.left = newLeft + "px";
}
```

- This could be used to move an element to a specific location on the screen.

Element Visibility, Colors and Fonts

- You can control the visibility of an XHTML element using the CSS property.

```
<div id="test" style="visibility: hidden">hi there</div>
```

```
<input type="button" onclick='show("test")' />
```

```
<script type="text/javascript">
```

```
    function show(id)
```

```
    {
```

```
        myStyle = document.getElementById(id).style;
```

```
        myStyle.visibility = "visible";
```

```
    }
```

```
</script>
```

- In a similar fashion one can change other CSS properties in response to events. For instance, if you like you could change the color or font type or size.

