CSE3026: Web Application Development
Events

Scott Uk-Jin Lee

11.1: Event-Handling

- 11.1: Event-Handling
  - 11.1.1 The Event Object
  - 11.1.2 Mouse Events
  - 11.1.3 Keyboard and Text Events
  - 11.1.4 Form Events
  - 11.1.5 Page Events
  - 11.1.6 Timer Events

JavaScript events

abort	blur	change	click	dblclick	error	focus
keydown	keypress	keyup	load	mousedown	mousemove	mouseout
mouseover	mouseup	reset	resize	select	submit	unload

- the `click` event (`onclick`) is just one of many events that can be handled
- **problem**: events are tricky and have incompatibilities across browsers
  - reasons: fuzzy W3C event specs; IE disobeying web standards; etc.
- **solution**: Prototype includes many event-related features and fixes

Attaching event handlers the Prototype way

```
element.onevent = function;
element.observe("event", function);
```
JS

```
// call the playNewGame function when the Play button is clicked
$("play").observe("click", playNewGame);
```
JS

- to use Prototype's event features, you must attach the handler using the DOM element object's `observe` method (added by Prototype)
- pass the **event name** as a string, and the **function name** to call
- handlers *must* be attached this way for Prototype's event features to work

`observe` substitutes for `addEventListener` (not supported by IE)

The `event` object

```JS
function name(event) {
    // an event handler function ...
}
```

- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

method / property name	description
`type`	what kind of event, such as `"click"` or `"mousedown"`
`element()` *	the element on which the event occurred
`stop()` **	cancels an event
`stopObserving()`	removes an event handler

* replaces non-standard `srcElement` and `which` properties

** replaces non-standard `return false;`, `stopPropagation`, etc.

Mouse events

`click`	user presses/releases mouse button on the element
`dblclick`	user presses/releases mouse button twice on the element
`mousedown`	user presses down mouse button on the element
`mouseup`	user releases mouse button on the element

clicking

`mouseover`	mouse cursor enters the element's box
`mouseout`	mouse cursor exits the element's box
`mousemove`	mouse cursor moves around within the element's box

movement

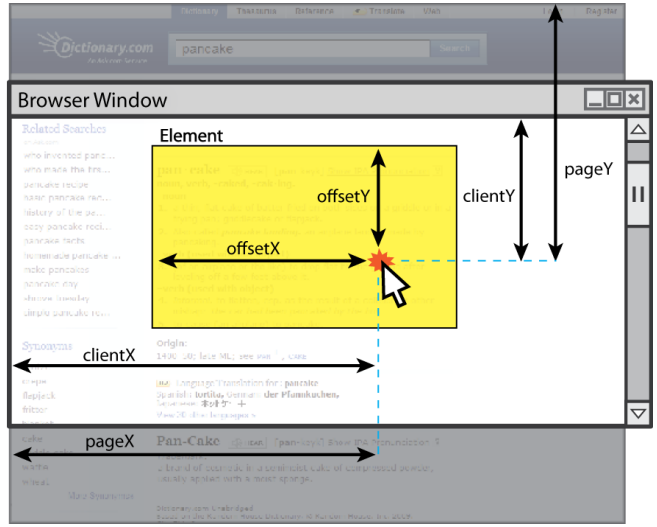
Mouse event objects

The `event` passed to a mouse handler has these properties:

property/method	description
`clientX`, `clientY`	coordinates in *browser window*
`screenX`, `screenY`	coordinates in *screen*
`offsetX`, `offsetY`	coordinates in *element* (non-standard)
`pointerX()`, `pointerY()` *	coordinates in *entire web page*
`isLeftClick()` **	`true` if left button was pressed

* replaces non-standard properties `pageX` and `pageY`

** replaces non-standard properties `button` and `which`



Mouse event example

```HTML
<pre id="target">Move the mouse over me!</pre>
```

```JS
window.onload = function() {
    $("target").observe("mousemove", showCoords);
};

function showCoords(event) {
    $("target").innerHTML =
          "pointer: (" + event.pointerX() + ", " + event.pointerY() + ")\n"
        + "screen : (" + event.screenX + ", " + event.screenY + ")\n"
        + "client : (" + event.clientX + ", " + event.clientY + ")";
}
```

```output
Move the mouse over me!
```

The keyword `this`

```
this.fieldName                    // access field
this.fieldName = value;           // modify field

this.methodName(parameters);      // call method
```
JS

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

Event handler binding

```
window.onload = function() {
    $("textbox").observe("mouseout", booyah);    // bound to text box here
    $("submit").observe("click", booyah);        // bound to submit button here
};

function booyah() {             // booyah knows what object it was called on
    this.value = "booyah";
}
```
JS

Save

output

- 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(this.value + " is checked!");
}
```

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

Page/window events

name	description
load, unload	the browser loads/exits the page
resize	the browser window is resized
error	an error occurs when loading a document or an image
contextmenu	the user right-clicks to pop up a context menu

- The above can be handled on the window object. An alternative to window.onload:

```
window.onload = function() { ... };
document.observe("dom:loaded", function() {
    // attach event handlers, etc.
});
```
JS

Keyboard/text events

name	description
`keydown`	user presses a key while this element has keyboard focus
`keyup`	user releases a key while this element has keyboard focus
`keypress`	user presses and releases a key while this element has keyboard focus
`focus`	this element gains keyboard focus
`blur`	this element loses keyboard focus
`select`	this element's text is selected or deselected)

- **focus**: the attention of the user's keyboard (given to one element at a time)

Key event objects

property name	description
`keyCode`	ASCII integer value of key that was pressed (convert to char with `String.fromCharCode`)
`altKey`, `ctrlKey`, `shiftKey`	`true` if Alt/Ctrl/Shift key is being held

`Event.KEY_BACKSPACE`	`Event.KEY_DELETE`	`Event.KEY_DOWN`	`Event.KEY_END`
`Event.KEY_ESC`	`Event.KEY_HOME`	`Event.KEY_LEFT`	`Event.KEY_PAGEDOWN`
`Event.KEY_PAGEUP`	`Event.KEY_RETURN`	`Event.KEY_RIGHT`	`Event.KEY_TAB`
`Event.KEY_UP`			

Prototype's key code constants

- issue: if the event you attach your listener to doesn't have the focus, you won't hear the event
  - possible solution: attach key listener to entire page body, outer element, etc.

Form events

event name	description
submit	form is being submitted
reset	form is being reset
change	the text or state of a form control has changed

- Prototype adds the following methods to form controls' DOM objects:

activate	clear	disable	enable
focus	getValue	present	select

Stopping an event

```HTML
<form id="exampleform" action="http://foo.com/foo.php">...</form>
```

```JS
window.onload = function() {
    $("exampleform").observe("submit", checkData);
};

function checkData(event) {
    if ($F("city") == "" || $F("state").length != 2) {
        alert("Error, invalid city/state.");  // show error message
        event.stop();
        return false;
    }
}
```

- to abort a form submit or other event, call Prototype's stop method on the event

Timer events

method	description
`setTimeout(`*function*`, `*delayMS*`);`	arranges to call given function after given delay in ms
`setInterval(`*function*`, `*delayMS*`);`	arranges to call function repeatedly every *delayMS* ms
`clearTimeout(`*timerID*`);` `clearInterval(`*timerID*`);`	stops the given timer so it will not call its function

- both `setTimeout` and `setInterval` return an ID representing the timer
  - this ID can be passed to `clearTimeout/Interval` later to stop the timer

`setTimeout` example

```
<button onclick="delayMsg();">Click me!</button>
<span id="output"></span>
```
HTML

```
function delayMsg() {
    setTimeout(booyah, 5000);
    document.getElementById("output").innerHTML = "Wait for it...";
}

function booyah() {   // called when the timer goes off
    document.getElementById("output").innerHTML = "BOOYAH!";
}
```
JS

Click me!

output

`setInterval` example

```
var timer = null;  // stores ID of interval timer

function delayMsg2() {
    if (timer == null) {
        timer = setInterval(rudy, 1000);
    } else {
        clearInterval(timer);
        timer = null;
    }
}

function rudy() {   // called each time the timer goes off
    document.getElementById("output").innerHTML += " Rudy!";
}
```

JS

Click me!

output

Passing parameters to timers

```
function delayedMultiply() {
    // 6 and 7 are passed to multiply when timer goes off
    setTimeout(multiply, 2000, 6, 7);
}
function multiply(a, b) {
    alert(a * b);
}
```
JS

Click me

output

- any parameters after the delay are eventually passed to the timer function
  - doesn't work in IE6; must create an intermediate function to pass the parameters
- why not just write this?

```
setTimeout(multiply(6 * 7), 2000);
```
JS

Common timer errors

- many students mistakenly write `()` when passing the function

```
setTimeout(booyah(), 2000);
setTimeout(booyah, 2000);

setTimeout(multiply(num1 * num2), 2000);
setTimeout(multiply, 2000, num1, num2);
```
JS

  - what does it actually do if you have the `()` ?
  - it calls the function immediately, rather than waiting the 2000ms!