Web Programming Step by Step

Chapter 9
Events and the Prototype Library

9.1: The Prototype JavaScript Library

- **9.1: The Prototype JavaScript Library**
- 9.2: Event-Handling

Problems with JavaScript

JavaScript is a powerful language, but it has many flaws:

- The DOM can be clunky to use
- The same code doesn't always work the same way in every browser
    - code that works great in Firefox, Safari, ... will fail in IE and vice versa
- Many web developers work around these problems with hacks:

```
// check if browser is IE (bad style!)
if (navigator.appName === "Microsoft Internet Explorer") { ...
```

JS

Prototype

```
<script src="http://www.cs.washington.edu/education/courses/cse190m/09sp/prototype.js"
 type="text/javascript"></script>

<!-- or, -->
<script src="http://prototypejs.org/assets/2008/1/25/prototype-1.6.0.2.js"
 type="text/javascript"></script>
```

JS

- Prototype JavaScript library adds many useful features to JavaScript:
    - many useful extensions to the DOM
    - added methods to String, Array, Date, Number, Object
    - improves event-driven programming
    - many cross-browser compatibility fixes
    - makes Ajax programming easier (seen later)

Prototype's new methods (9.1.2)

Prototype adds new methods to many existing JavaScript types:

Array	clear	clone	compact	each	first	flatten	from	**indexOf**
	inspect	last	reduce	**reverse**	size	toArray	uniq	without

Number	abs	ceil	floor	round	succ	times	toColorPart	toPaddedString

Object	clone	extend	**inspect**	isArray	isElement	isFunction	isHash
	isNumber	isString	isUndefined	keys	toHTML	toQueryString	values

String	blank	camelize	capitalize	dasherize	empty	**endsWith**	**escapeHTML**
	include	inspect	interpolate	parseQuery	scan	**startsWith**	**strip**
	sub	**stripTags**	toQueryParams	times	toArray	underscore	**unescapeHTML**

The $ function (9.1.3)

```
$("id")
```
JS

- returns the DOM object representing the element with the given `id`
- short for `document.getElementById("id")`
- often used to write more concise DOM code:

```
$("footer").innerHTML = $("username").value.toUpperCase();
```
JS

DOM element methods

absolutize	**addClassName**	**classNames**	cleanWhitespace	clonePosit
cumulativeOffset	cumulativeScrollOffset	empty	extend	firstDesce
getDimensions	getHeight	getOffsetParent	**getStyle**	getWidth
hasClassName	**hide**	identify	insert	inspect
makeClipping	makePositioned	match	positionedOffset	readAttrib
recursivelyCollect	relativize	**remove**	**removeClassName**	replace
scrollTo	select	setOpacity	setStyle	**show**
toggle	toggleClassName	undoClipping	undoPositioned	update
viewportOffset	visible	wrap	writeAttribute	

- categories: CSS classes, DOM tree traversal/manipulation, events, styles

Styles and CSS classes (9.1.4)

```
function makeFontBigger() {
  // turn text yellow and make it bigger
  if (!$("text").hasClassName("highlight")) {
    $("text").addClassName("highlight");
  }
  var size = parseInt($("text").getStyle("font-size"));
  $("text").style.fontSize = (size + 2) + "pt";
}
```

- `getStyle` function added to DOM object allows accessing existing styles
- `addClassName`, `removeClassName`, `hasClassName` manipulate CSS classes

Common bug: incorrect usage of existing styles

```
~~this.style.top = this.getStyle("top") + 100 + "px";~~            // bad!
```

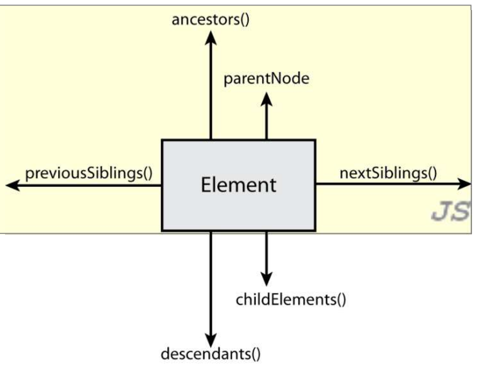
- the above example computes e.g. `"200px" + 100 + "px"` , which would evaluate to `"200px100px"`
- a corrected version:

```
this.style.top = parseInt(this.getStyle("top")) + 100 + "px";   // correct
```

DOM tree traversal methods

method(s)	description
`ancestors`, `up`	elements above this one
`childElements`, `descendants`, `down`	elements below this one (not text nodes)
`siblings`, `next`, `nextSiblings`, `previous`, `previousSiblings`, `adjacent`	elements with same parent as this one (not text nodes)

```
// remove elements in "main" that do not contain "Sun"
var sibs = $("main").siblings();
for (var i = 0; i < sibs.length; i++) {
  if (sibs[i].innerHTML.indexOf("Sun") < 0) {
    sibs[i].remove();
  }
}
```



- notice that these are methods, so you need `()`

Methods for selecting elements

Prototype adds methods to the `document` object (and all DOM element objects) for selecting groups of elements:

`getElementsByClassName`	array of elements that use given `class` attribute
`select`	array of elements that match given CSS selector, such as `"div#sidebar ul.news > li"`

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

The $$ function (9.1.5)

```
var arrayName = $$("CSS selector");
```
JS

```
// hide all "announcement" paragraphs in the "news" section
var paragraphs = $$("div#news p.announcement");
for (var i = 0; i < paragraphs.length; i++) {
  paragraphs[i].hide();
}
```
JS

- `$$` returns an array of DOM elements that match the given CSS selector
  - like `$` but returns an array instead of a single DOM object
  - a shorthand for `document.select`
- useful for applying an operation each one of a set of elements

Common $$ issues

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

```
// get all buttons with a class of "control"
var gameButtons = ~~$$("control");~~
var gameButtons = $$(".control");
```
JS

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

```
// set all buttons with a class of "control" to have red text
~~$$(".control").style.color = "red";~~
var gameButtons = $$(".control");
for (var i = 0; i < gameButtons.length; i++) {
  gameButtons[i].style.color = "red";
}
```
JS

- Q: Can I still select a group of elements using `$$` even if my CSS file doesn't have any style rule for that same group? (A: Yes!)

Prototype and forms (9.1.6)

```
$F("id")
```
JS

- `$F` returns the value of a form control with the given id

```
var name = $F("username");
if (name.length < 4) {
  $("username").clear();
  $("login").disable();
}
```
JS

- other form control methods:

activate	clear	disable	enable
focus	getValue	present	select

9.2: Event-Handling

More about 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 of interest and the function to use as the handler
- handlers *must* be attached this way for Prototype's event features to work

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

Attaching multiple event handlers with `$$`

```
// listen to clicks on all buttons with class "control" that
// are directly inside the section with ID "game"
window.observe("load", function() {
  var gameButtons = $$("#game > button.control");
  for (var i = 0; i < gameButtons.length; i++) {
    gameButtons[i].observe("click", 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
- notice that the observe syntax can also be used for `window.onload`

The `Event` object

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

JS

- 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 (9.2.2)

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

clicking

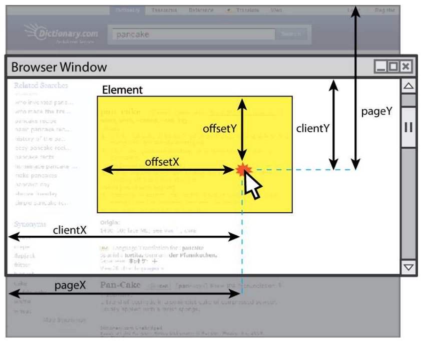
`mouseover`	mouse cursor enters this element's box
`mouseout`	mouse cursor exits this element's box
`mousemove`	mouse cursor moves around within this element's box

movement

Mouse event objects

The `event` parameter passed to a mouse event handler has the following properties:

property/method	description
`clientX`, `clientY`	coordinates in *browser window*
`screenX`, `screenY`	coordinates in *screen*
`offsetX`, `offsetY`	coordinates in *element*
`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>
```
HTML

```javascript
window.observe("load", function() {
  $("target").observe("mousemove", showCoords);
});

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

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

Keyboard/text events (9.2.3)

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 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

Form events (9.2.4)

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

```
window.observe("load", function() {
  $("orderform").observe("submit", verify);
});

function verify(event) {
  if ($F("zipcode").length < 5) {
    event.stop();        // cancel form submission unless
  }                      // zip code is 5 chars long
}
```
JS

Page/window events (9.2.5)

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

- The above events can be handled on the global `window` object. Also:

```
// best way to attach event handlers on page load
~~window.observe("load", function() {~~
document.observe("dom:loaded", function() {
  $("orderform").observe("submit", verify);
});
```
JS

Timer events (9.2.6)

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

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

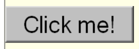
`setTimeout` example

```
<button id="clickme">Click me!</button>
<span id="output"></span>
```
HTML

```
document.observe("dom:loaded", function() {
  $("clickme").observe("click", delayMsg);
});

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

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



output

`setInterval` example

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

document.observe("dom:loaded", function() {
  $("clickme").observe("click", delayMsg2);
});

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

function rudy() {   // called each time the timer goes off
  $("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 passed to the timer function
    - doesn't work in IE6; must create an intermediate function to pass the parameters

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!