CSE3026: Web Application Development Prototype

Scott Uk-Jin Lee

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 developers work around these problems with hacks (checking if browser is IE, etc.)

Prototype framework

```
<script src="prototype.js" type="text/javascript"></script>
```

```
<script src="http://ajax.googleapis.com/ajax/libs/prototype/1.7.3.0/prototype.js"
  type="text/javascript"></script>
```

- the 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)
- In order to use Prototype JavaScript library :
  - download Prototype library from Prototype homepage
  - link to the downloaded Prototype.js in web page or the Google hosted Prototype JavaScript library

The $ function

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

- 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();
```

Prototype's DOM element methods

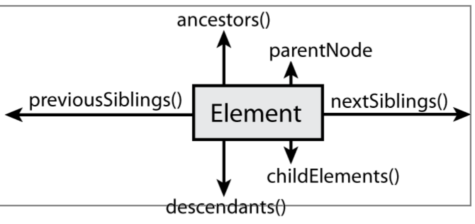
absolutize	**addClassName**	**classNames**	cleanWhitespace	clonePosition
cumulativeOffset	cumulativeScrollOffset	empty	extend	firstDescendant
getDimensions	getHeight	getOffsetParent	**getStyle**ㅊㅊㅊㅊ	getWidth
hasClassName	**hide**	identify	insert	inspect
makeClipping	makePositioned	match	positionedOffset	readAttribute
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

Prototype's 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)

```
// alter siblings of "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].innerHTML += " Sunshine";
    }
}
```



- Prototype strips out the unwanted text nodes
- notice that these are methods, so you need `()`

Selecting groups of DOM objects

- methods in `document` and other DOM objects for accessing descendents:

name	description
`getElementsByTagName`	returns array of descendents with the given tag, such as `"div"`
`getElementsByName`	returns array of descendents with the given `name` attribute (mostly useful for accessing form controls)
`querySelector` *	returns the first element that would be matched by the given CSS selector string
`querySelectorAll` *	returns an array of all elements that would be matched by the given CSS selector string

* = HTML5 : older browsers did not support `querySelectorAll` methods

Prototype's methods for selecting elements

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

현재 노드에서 querySelect를 할 때 사용한다.

`select`	array of descendants that match given CSS selector, such as `"div#sidebar ul.news > li"`
`$$`	equivalent to `document.querySelectorAll`

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

The `$$` function

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

```
// 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();
}
```

- `$$` 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`
  - essentially equivalent to `document.querySelectorAll`
- 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");
```

- `$$` 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";
}
```

- 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!)

Problems with reading/changing styles

```
<button id="clickme">Click Me</button>
```

```
window.onload = function() {
    $("clickme").onclick = biggerFont;
};
function biggerFont() {
    var size = parseInt($("clickme").style.fontSize);
    size += 4;
    $("clickMe").style.fontSize = size + "pt";
}
```

Click Me

- `style` property lets you set any CSS style for an element
- problem: you cannot (usually) read existing styles with it

Accessing styles in Prototype

```
function biggerFont() {
    // turn text yellow and make it bigger
    var size = parseInt($("clickme").getStyle("font-size"));
    $("clickme").style.fontSize = (size + 4) + "pt";
}
```

Click Me

- `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
```

Setting CSS classes in Prototype

```
function highlightField() {
    // turn text yellow and make it bigger
    if (!$("text").hasClassName("invalid")) {
        $("text").addClassName("highlight");
    }
}
```

- `addClassName`, `removeClassName`, `hasClassName` manipulate CSS classes
- similar to existing `className` DOM property, but don't have to manually split by spaces

Prototype form shortcuts

`$("formID")["name"]`

```
<form id="contract">
  <input name="company">
  </input>
</form>
```

- gets parameter with given **name** from form with given **id**

`$F("controlID")`

$("contract")["company"].getValue();

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

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

$(input).getValue(); == $F(input)