Web Programming Step by Step

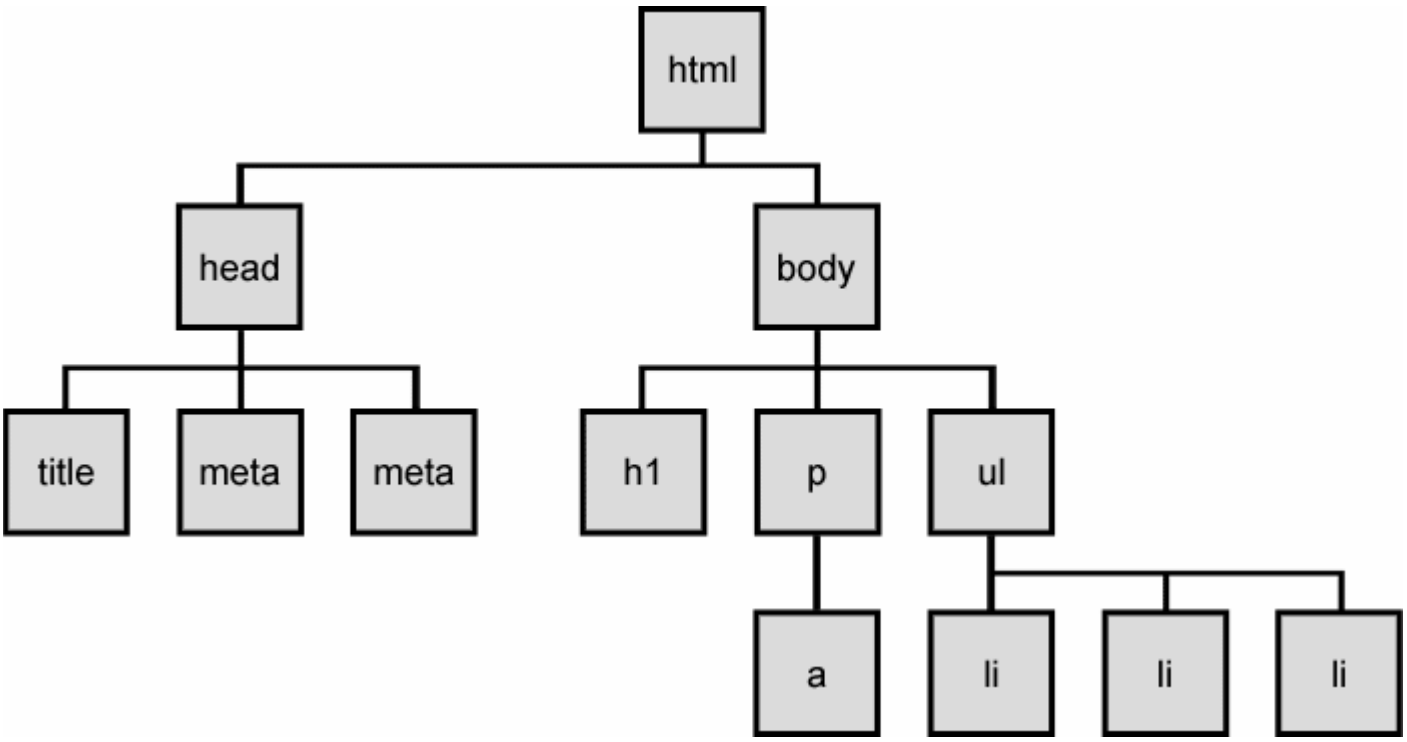
Lecture 16
The DOM Tree
Reading: 8.3, 9.1

Complex DOM manipulation problems

How would we do each of the following in JavaScript code? Each involves modifying each one of a group of elements ...

- When the Go button is clicked, reposition all the `divs` of class `puzzle` to random x/y locations.
- When the user hovers over the maze boundary, turn all maze walls red.
- Change every other item in the `ul` list with `id` of `TAs` to have a gray background.

The DOM tree (8.3)

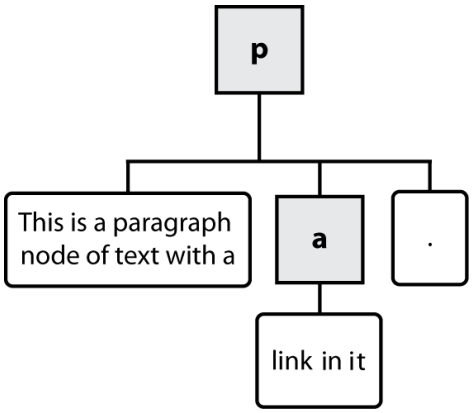


- The elements of a page are nested into a tree-like structure of objects
  - the DOM has properties and methods for traversing this tree

Types of DOM nodes (8.3.1)

```
<p>
  This is a paragraph of text with a
  <a href="/path/page.html">link in it</a>.
</p>
```
HTML

- **element nodes** (HTML tag)
  - can have children and/or attributes
- **text nodes** (text in a block element)
- **attribute nodes** (attribute/value pair)
  - text/attributes are children in an element node
  - cannot have children or attributes
  - not usually shown when drawing the DOM tree



Traversing the DOM tree (8.3.2 - 8.3.3)

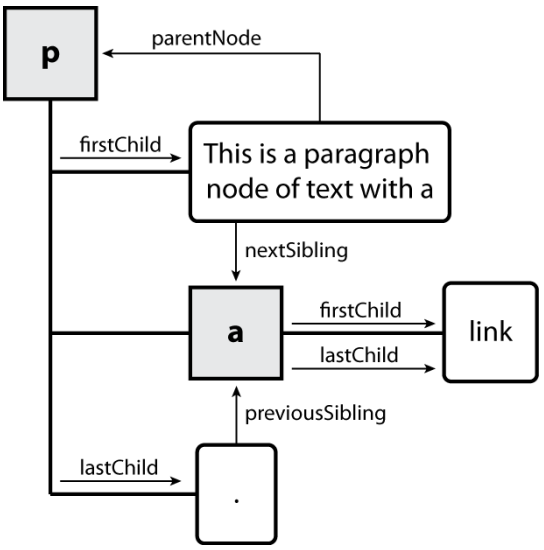
every node's DOM object has the following properties:

name(s)	description
`firstChild`, `lastChild`	start/end of this node's list of children
`childNodes`	array of all this node's children
`nextSibling`, `previousSibling`	neighboring nodes with the same parent
`parentNode`	the element that contains this node

- complete list of DOM node properties
- browser incompatiblity information (IE6 sucks)

DOM tree traversal example

```HTML
<p id="foo">This is a paragraph of text with a
  <a href="/path/to/another/page.html">link</a>.</p>
```



Element vs. text nodes

```HTML
<div>
  <p>
    This is a paragraph of text with a
    <a href="page.html">link</a>.
  </p>
</div>
```

- Q: How many children does the `div` above have?
- A: 3
  - an element node representing the <p>
  - two *text nodes* representing `"\n\t"` (before/after the paragraph)
- Q: How many children does the paragraph have? The `a` tag?

Prototype's DOM element methods (9.1.3)

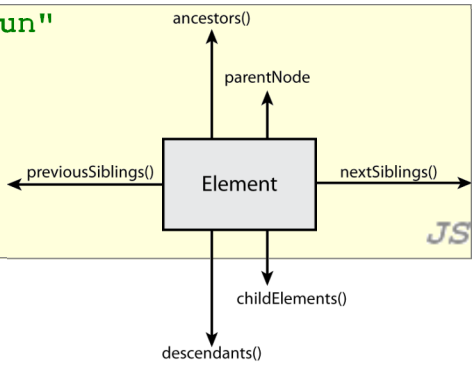
absolutize	**addClassName**	**classNames**	cleanWhitespace	
cumulativeOffset	cumulativeScrollOffset	empty	extend	
getDimensions	getHeight	getOffsetParent	**getStyle**	
hasClassName	**hide**	identify	insert	
makeClipping	makePositioned	match	positionedOffset	
recursivelyCollect	relativize	**remove**	**removeClassName**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

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

Prototype's DOM tree traversal methods (9.1.5)

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 (8.3.5)

- 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)

Getting all elements of a certain type

highlight all paragraphs in the document:

```
var allParas = document.getElementsByTagName("p");
for (var i = 0; i < allParas.length; i++) {
  allParas[i].style.backgroundColor = "yellow";
}
```
JS

```
<body>
  <p>This is the first paragraph</p>
  <p>This is the second paragraph</p>
  <p>You get the idea...</p>
</body>
```
HTML

Combining with `getElementById`

highlight all paragraphs inside of the section with ID `"address"`:

```
var addrParas = $("address").getElementsByTagName("p");
for (var i = 0; i < addrParas.length; i++) {
  addrParas[i].style.backgroundColor = "yellow";
}
```
JS

```
<p>This won't be returned!</p>
<div id="address">
  <p>1234 Street</p>
  <p>Atlanta, GA</p>
</div>
```
HTML

Prototype's 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 descendants 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!)

Creating new nodes (8.3.5)

name	description
`document.createElement("tag")`	creates and returns a new empty DOM node representing an element of that type
`document.createTextNode("text")`	creates and returns a text node containing given text

```
// create a new <h2> node
var newHeading = document.createElement("h2");
newHeading.innerHTML = "This is a heading";
newHeading.style.color = "green";
```
JS

- merely creating a node does not add it to the page
- you must add the new node as a child of an existing element on the page...

Modifying the DOM tree

Every DOM element object has these methods:

name	description
`appendChild(`*node*`)`	places given node at end of this node's child list
`insertBefore(`*new*`, `*old*`)`	places the given new node in this node's child list just before `old` child
`removeChild(`*node*`)`	removes given node from this node's child list
`replaceChild(`*new*`, `*old*`)`	replaces given child with new node

```
var p = document.createElement("p");
p.innerHTML = "A paragraph!";
$("main").appendChild(p);
```
JS

Removing a node from the page

```
function slideClick() {
  var bullets = document.getElementsByTagName("li");
  for (var i = 0; i < bullets.length; i++) {
    if (bullets[i].innerHTML.indexOf("children") >= 0) {
      bullets[i].remove();
    }
  }
}
```
JS

- each DOM object has a `removeChild` method to remove its children from the page
- Prototype adds a `remove` method for a node to remove itself

DOM versus `innerHTML` hacking

Why not just code the previous example this way?

```
function slideClick() {
  $("thisslide").innerHTML += "<p>A paragraph!</p>";
}
```
JS

- Imagine that the new node is more complex:
  - ugly: bad style on many levels (e.g. JS code embedded within HTML)
  - error-prone: must carefully distinguish " and '
  - can only add at beginning or end, not in middle of child list

```
function slideClick() {
  this.innerHTML += "<p style='color: red; " +
      "margin-left: 50px;' " +
      "onclick='myOnClick();'>" +
      "A paragraph!</p>";
}
```
JS

Problems with reading/changing styles

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

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

Click Me

output

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

Accessing styles in Prototype (9.1.4)

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

Click Me

output

- `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!
```
JS

- 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
```
JS

Setting CSS classes in Prototype (9.1.4)

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

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