中山大學
SUN YAT-SEN UNIVERSITY

Lecture 16 Manipulating DOM with JavaScript

SE-805 Web 2.0 Programming (supported by Google)

http://my.ss.sysu.edu.cn/courses/web2.0/

School of Software, Sun Yat-sen University

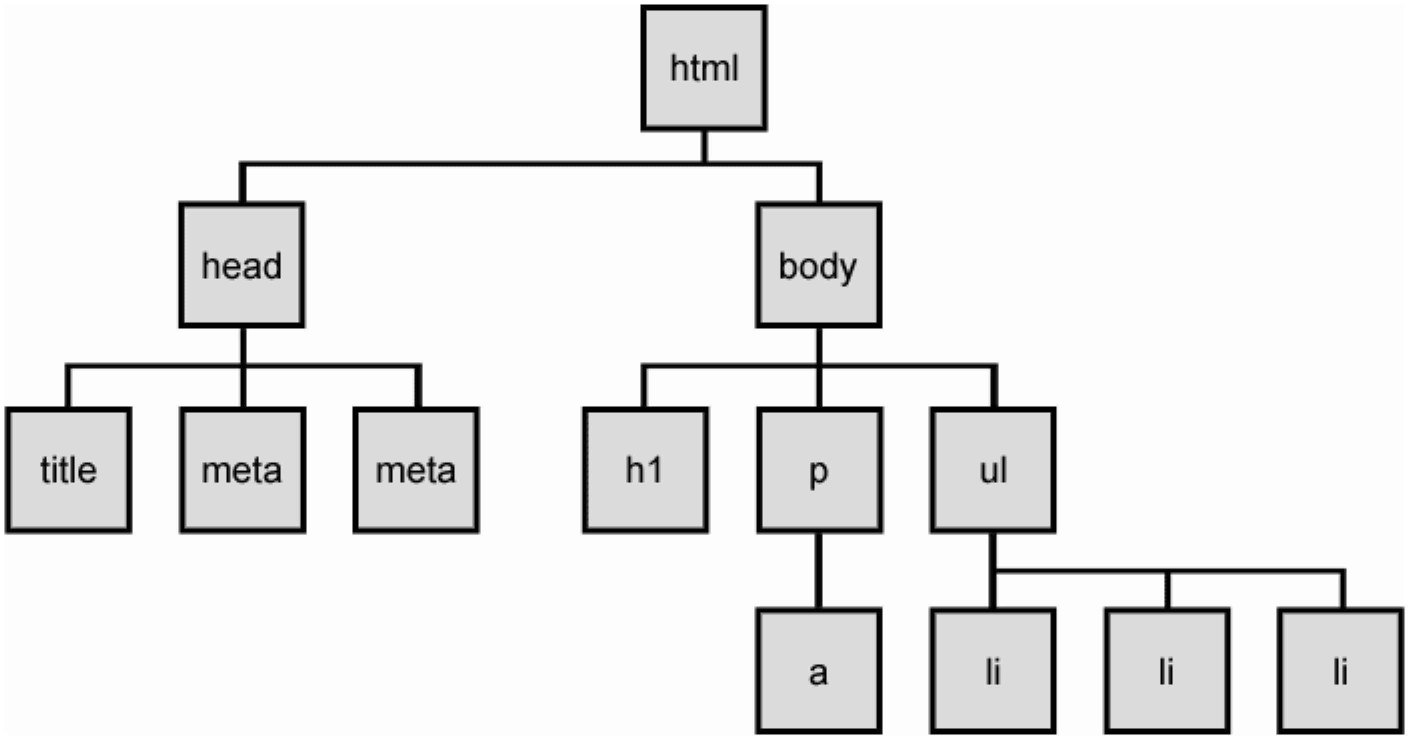
Outline

- **The DOM tree**
- Manipulating DOM

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

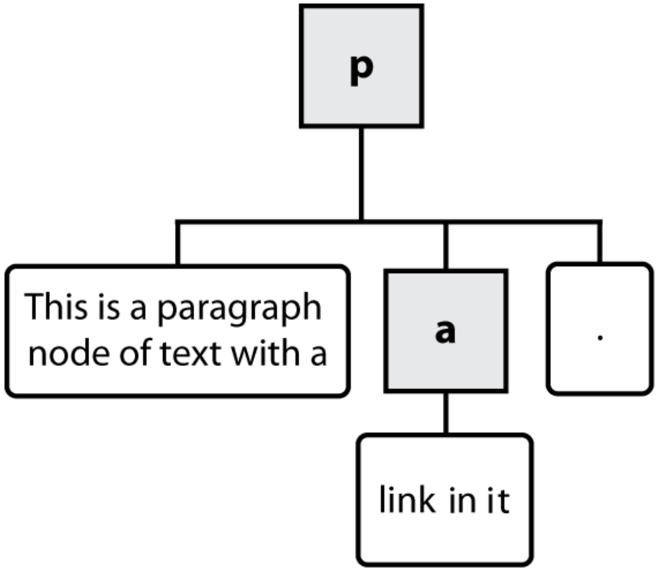


- The elements of a page are nested into a tree-like structure of objects – **DOM tree**
  - The **DOM** has properties and methods for traversing this tree

Type of DOM Nodes

```
<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

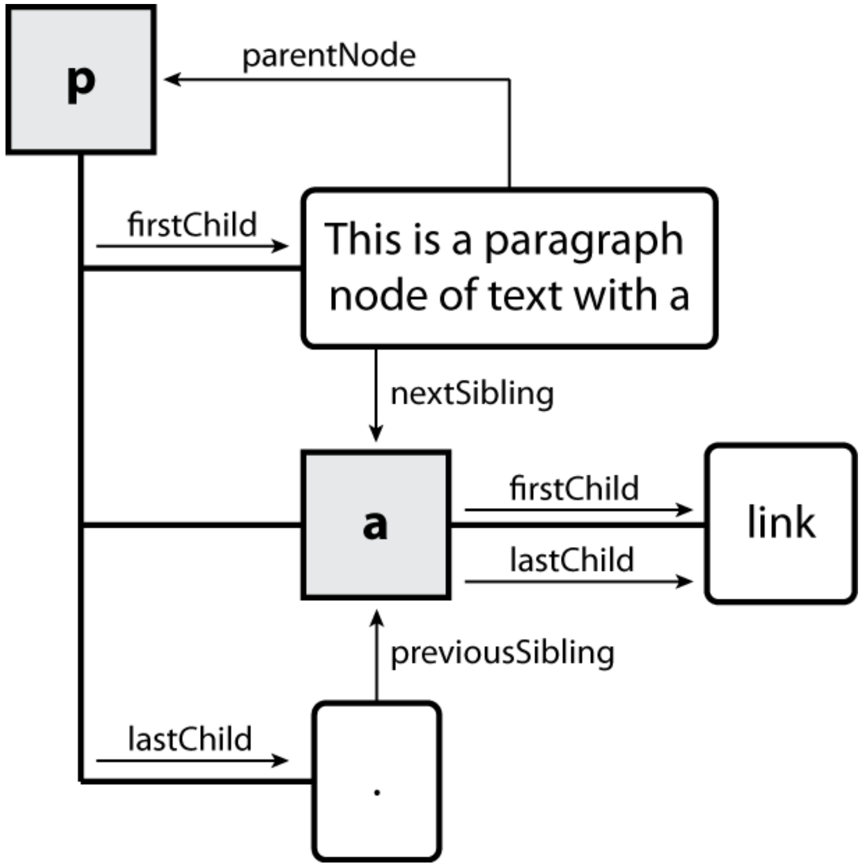
- Every node's DOM object has the following properties:

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 (IE 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>       HTML
```



Element vs. Text Nodes

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

- 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?

Outline

- The DOM tree
- **Manipulating DOM**

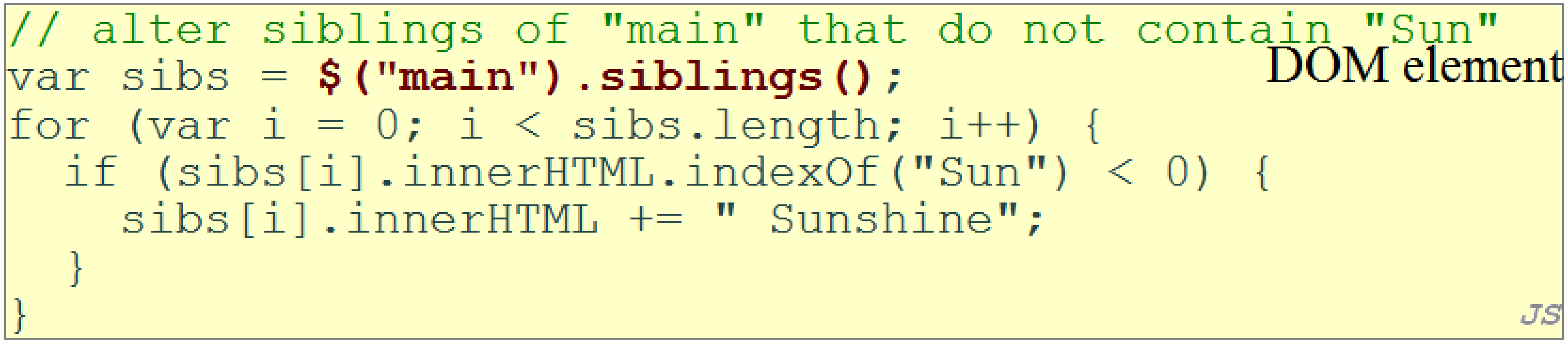
Prototype's DOM Element Methods

absolutize	**addClassName**	**classNames**	cleanWhitespace	clonePosition
cumulativeOffset	cumulativeScrollOffset	empty	extend	firstDescendant
getDimensions	getHeight	getOffsetParent	**getStyle**	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)

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

```
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 (in CSS 3, much more powerful than CSS 2)
  - 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
~~$$(".control").style.color = "red";~~
var gameButtons = $$(".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

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

```javascript
// 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

A paragraph!

A paragraph!

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 vs. 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

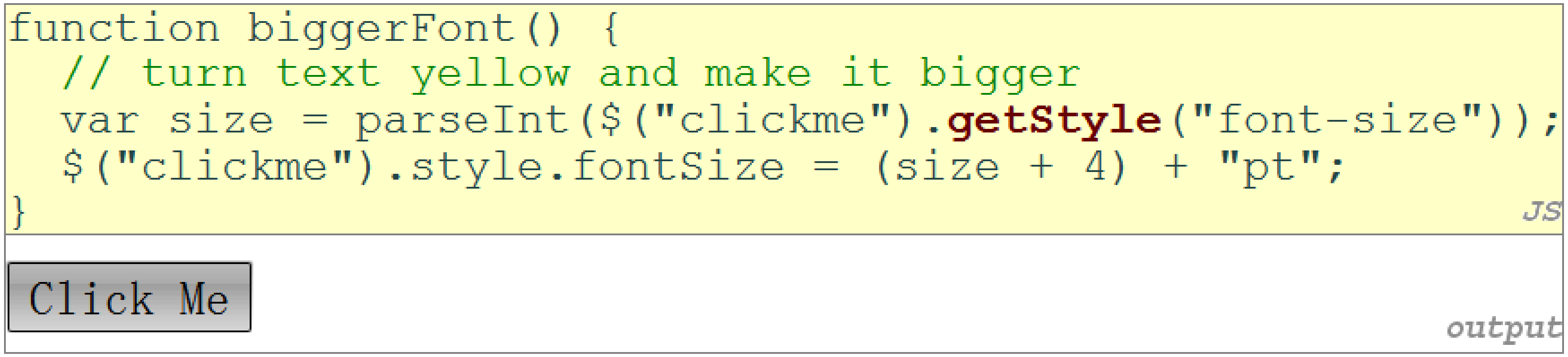
```html
<button id="clickme">Click Me</button>
```
HTML

```javascript
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



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

Setting CSS Classes in Prototype

```
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

Useful Prototype $□ Functions

- **`$()`**, **`$$()`**
- **`$w()`**: string to array
  - `$w('raspberries pears peaches kiwis' );`
    `// ['raspberries' , 'pears' , 'peaches' , 'kiwis' ]`
- **`$A()`**: an universal converter that turns just about anything roughly collection-like (or, if you prefer, “array-compatible”) into an actual Array object.
  - `var ps = $A(document.getElementsByTagName('p' ));`
    `ps.each(Element.hide);`
- **`$F( )`** takes a form field (or its ID) and returns the field’s value
  - Alleviating pains of different algorithms for `textarea, radio, checkbox, list` ….
- **`$H()`** for **Hash**, **`$F()`** for **Range**, both are useful objects in Prototype

Summary

- The DOM tree
  - DOM tree, nodes type
  - Traversing DOM, text nodes
- Manipulating DOM
  - Prototype's DOM methods
  - Select by tagName, name, className, CSS selector
  - $$
  - DOM vs. innerHTML
  - Access styles programmatically
  - $□ functions

Exercises

- Write a simple to-do list application as a web page.
  - A form with a textarea for specifying a new to-do item and a “add” button for adding it to the list
  - A list of current to-do items
  - Each item has a checkbox for select
  - Buttons “select all”, “deselect all”, “remove” (which removes all selected to-do items from the list)
  - When the “add” button is clicked the new to-do item will be inserted to the bottom of the list

Further Readings

- W3School DOM node reference
  http://www.w3school.com/dom/dom_node.asp/
- W3School DOM tutorial
  http://www.w3schools.com/htmldom/
- Quirksmode DOM tutorial
  http://www.quirksmode.org/dom/intro.html
- Prototype Learning Center
  http://www.prototypejs.org/learn
- How prototype extends the DOM
  http://www.prototypejs.org/learn/extensions

Thank you!