Document Object Model

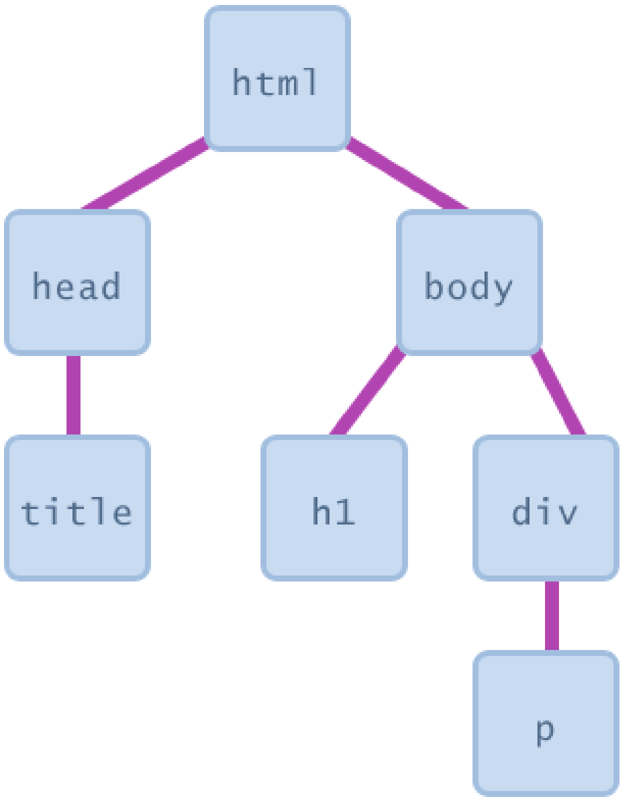
고급 웹 개발응용 실습
제주대학교 컴퓨터공학부

본 내용은 University of Washington, CSE 190 M (Web Programming), Spring 2007 수업에서 사용된 자료입니다.

Except where otherwise noted, the contents of this presentation are © Copyright 2007 Marty Stepp and are licensed under the Creative Commons Attribution 2.5 License.

Recall: Document Object Model (DOM)

- a representation of the current web page as a tree of Javascript objects
- allows you to view/modify page elements in script code after page has loaded
  - client side = highly responsive interactions
  - browser-independent
  - allows progressive enhancement of pages
- also used for **XML parsing**, which we'll do soon

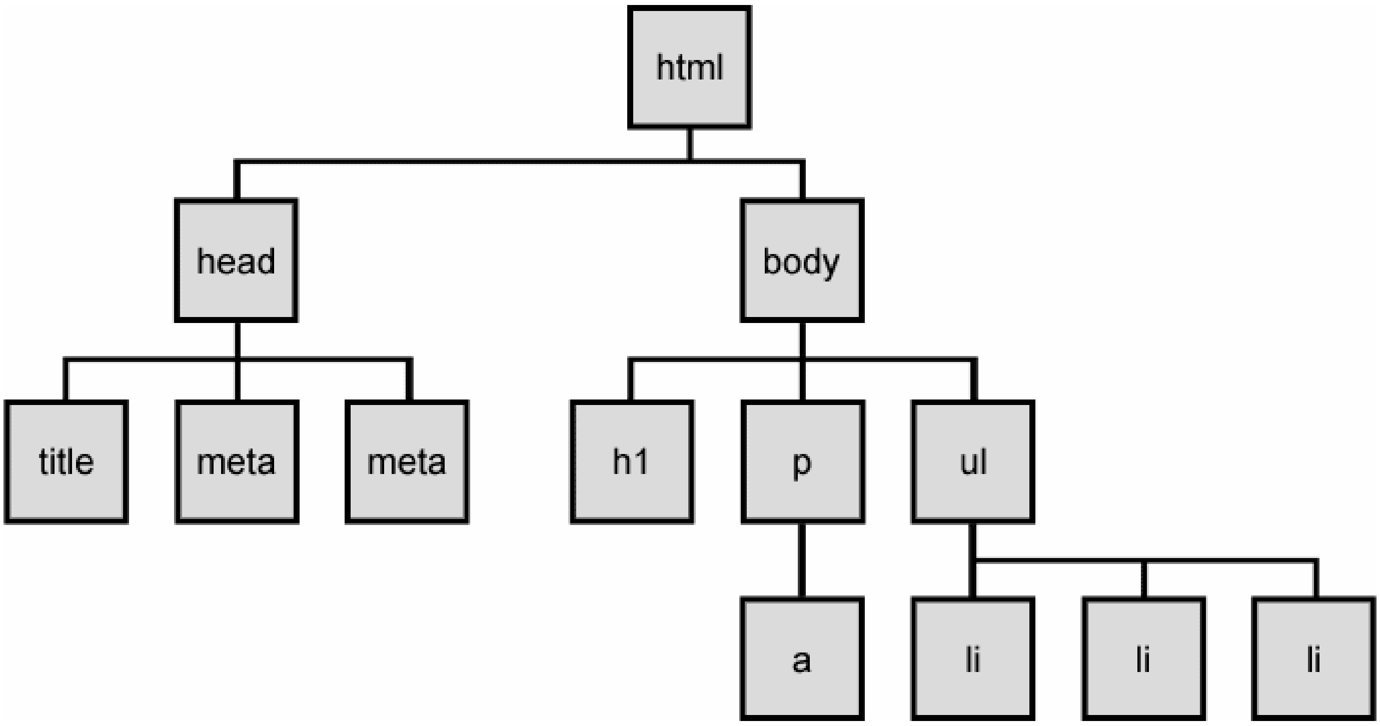


An example XHTML page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/XHTML1/DTD/XHTML1-strict.dtd">
<html xmlns="http://www.w3.org/1999/XHTML" xml:lang="en" lang="en">
<head>
<title>Page Title</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Language" content="en-us" />
</head>
<body>
```

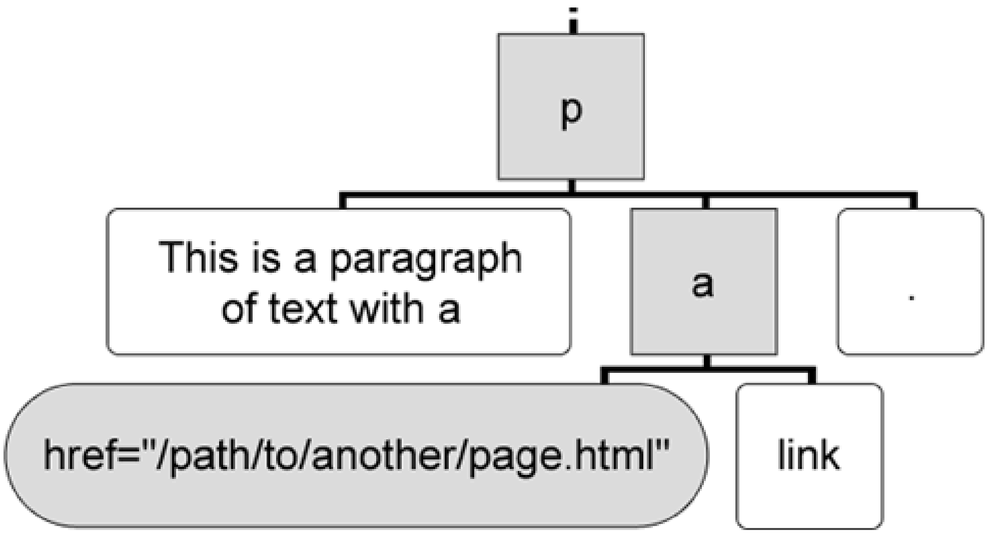
```
<h1>This is a heading</h1>
<p>A paragraph with a <a href="http://www.google.com/">link</a>.</p>
<ul>
<li>a list item</li>
<li>another list item</li>
<li>a third list item</li>
</ul>
</body>
</html>
```

The resulting DOM tree



Types of nodes

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


p
This is a paragraph of text with a
a
.
href="/path/to/another/page.html"
link

- ▨ **element nodes** (HTML tag)
  - can have children and/or attributes
- □ **text nodes** (text in a block element)
  - a child within an element node
  - cannot have children or attributes

- **attribute nodes** (attribute/value pair inside the start of a tag)
  - a child within an element node
  - cannot have children or attributes

Traversing the DOM tree

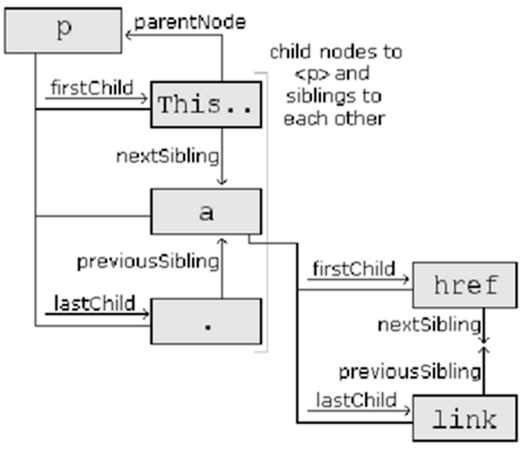
every node's DOM object has the following properties:

- `firstChild`, `lastChild` : start/end of this node's list of children
- `childNodes` : array of all this node's children
- `nextSibling`, `previousSibling` : neighboring nodes that have 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

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



- How would we change the word "link" in the above HTML to be "bunny"?

Modifying the DOM tree

Every DOM node object has these methods:

- `appendChild(node)` : places the given node at the end of this node's child list
- `insertBefore(newChild, oldChild)` : places the given new node in this node's child list just before `oldChild`
- `removeChild(node)` : removes the given node from this node's child list
- `replaceChild(newChild, oldChild)` : replaces the given child node with the given new node

Creating new nodes: `createElement`

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

// put it onto the page in the div with id "content"
var contentArea = document.getElementById("content");
contentArea.appendChild(newHeading);
```

- `document.createElement("tag")` constructs a new empty DOM node representing an element of that type
- this node's properties can be set just like any other DOM node's
- once appropriate properties are set, the node can be added to the page

Modifying tree example

```
    // in window.onload event handler,
    document.getElementById("thisslide").onclick = slideClick;

function slideClick() {
    var p = document.createElement("p");
    p.innerHTML = "A paragraph!";
    this.appendChild(p);
}
```

DOM versus `innerHTML`

Why not just code the previous example this way?

```
// equivalent to previous slide, but worse style
function slideClick() {
    this.innerHTML += "<p>A paragraph!</p>";
}
```

Ugly `innerHTML` code

Imagine that the new node is more complex:

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

- ugly as hell
- must carefully distinguish `"` and `'`
- bad style on many levels (style *and* JS code embedded within HTML)
- can only add at beginning or end, not in middle of child list

Benefits of DOM over `innerHTML`

```
function slideClick() {
    var p = document.createElement("p");
    p.style.color = "red";
    p.style.marginLeft = "50px";
    p.onclick = myOnClick;
    p.innerHTML = "A paragraph!";  // here innerHTML is okay
    this.appendChild(p);
}
```

- cleaner to attach event handlers to DOM object
- cleaner to set styles on DOM object
- still okay to use `innerHTML` if a node's inner contents are trivial

Practice problem: Rectangles

Click a rectangle to move it to the front. Shift-click a rectangle to delete it.

- Write Javascript code to create and manipulate random rectangles (HTML, CSS, JS).
- Hint: See absolute and relative positioning from the layout slides.
- Hint: Use `z-index` property to adjust which rectangles are on top.

----- More DOM -----

More DOM features

Accessing nodes by id or tag

```
document.getElementById("id")
```

- gets a specific element on the page

```
element.getElementsByTagName("tag")
```

- get an array of all children of the given type (`"p"`, `"div"`, etc.)
- can be called on the overall `document` or on a specific node

Getting all elements of a certain type

highlight all paragraphs in document

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

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

Combining with `getElementById`

highlight all paragraphs inside of the section with ID `"footer"`

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

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

Global DOM objects

Every Javascript program can refer to the following global objects:

- `window` : the browser window
- `navigator` : info about the web browser you're using
- `screen` : info about the screen area occupied by the browser
- `history` : list of pages the user has visited
- `location` : URL of the current HTML page
- `document` : current HTML page object model

The `window` object

- represents the entire browser window
- the top-level object in the DOM hierarchy
- technically, all global variables become part of the `window` object
- methods:
  - `alert`, `blur`, `clearInterval`, `clearTimeout`, `close`, `confirm`, `focus`, `moveBy`, `moveTo`, `open`, `print`, `prompt`, `resizeBy`, `resizeTo`, `scrollBy`, `scrollTo`, `setInterval`, `setTimeout`
- properties:
  - `document`, `history`, `location`, `name`

The `navigator` object

- information about the web browser application
- properties:
  - `appName`, `appVersion`, `browserLanguage`, `cookieEnabled`, `platform`, `userAgent`
  - complete list

The `screen` object

- information about the client's display screen
- properties:
  - `availHeight`, `availWidth`, `colorDepth`, `height`, `pixelDepth`, `width`
  - complete list

The `history` object

- list of sites the browser has visited in this window
- properties:
  - `length`
- methods:
  - `back`, `forward`, `go`
- complete list

The `location` object

- represents the URL of the current web page
- properties:
  - `host`, `hostname`, `href`, `pathname`, `port`, `protocol`, `search`
- methods:
  - `assign`, `reload`, `replace`
- complete list

The `document` object

- represents the URL of the current web page
- properties:
  - `anchors`, body, `cookie`, `domain`, `forms`, `images`, `links`, `referrer`, `title`, URL
- methods:
  - `close`, `getElementById`, `getElementsByName`, `getElementsByTagName`, `open`, `write`, `writeln`
- complete list