4

BOM DOM and Events

Topics

- HTML5 Documents
- HTML5 Forms
  - Reading Data from Forms
  - Validating HTML5 Form Input
  - Sending Data from Forms
- Controlling HTML5 Events with JavaScript
  - Writing `onClick` Events
- Document Object Model (DOM)

Controlling HTML5 Events with JavaScript

1. Event handling in HTML elements using attributes:

```
<section id="section-1" onclick="showSpeakersBanner()">
```

2. Event handling from JavaScript and DOM:

```
document.getElementById("section-2").onclick =
    showWorkshopBanner;
```

3. Event handling by adding an event listener:

```
window.addEventListener("load", function() {
  alert("Window load event listener fired!");
}, false);
```

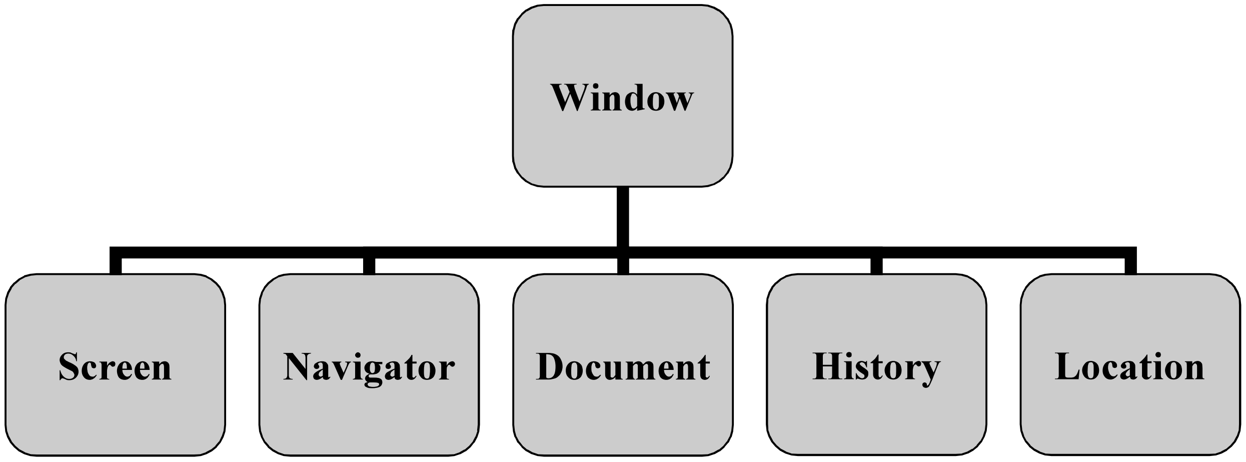
```
document.getElementById("headerProposal").addEventListener(
    "click", showAbstractBanner, false);
```

Browser Object

The Browser Object Model (BOM) is a collection of objects that interact with the browser window.

- Window Object
- History Object
- Location Object
- Navigator Object
- Screen Object
- Document Object

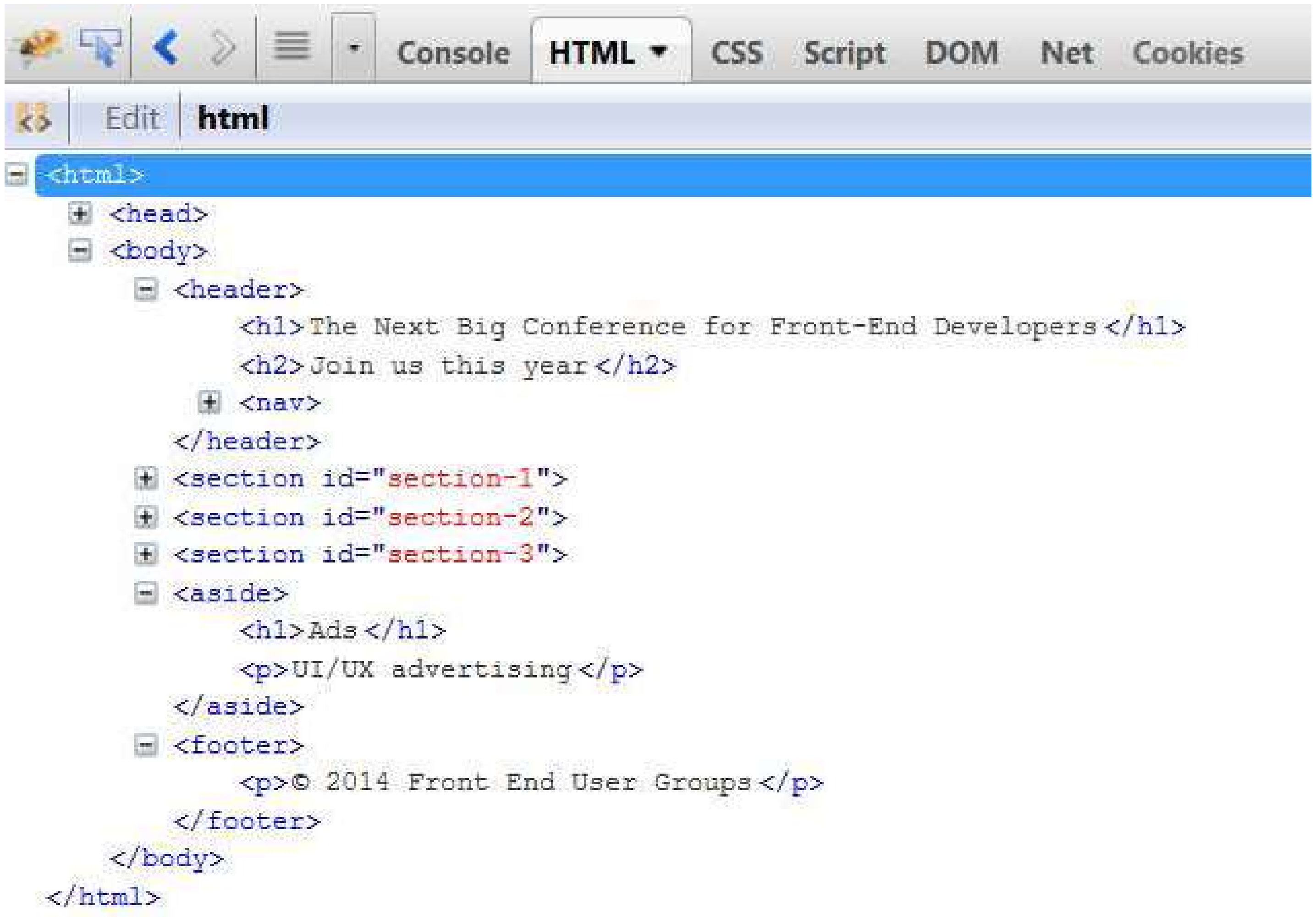
Browser Objects



Window Object

- Window Object is at the Top of object hierarchy.
- Acts as a master container for all the contents you view in the Web Browser.
- Window object has several properties and methods.

The `document` Object



```
<html>
    <head>
    <body>
        <header>
            <h1>The Next Big Conference for Front-End Developers</h1>
            <h2>Join us this year</h2>
            <nav>
        </header>
        <section id="section-1">
        <section id="section-2">
        <section id="section-3">
        <aside>
            <h1>Ads</h1>
            <p>UI/UX advertising</p>
        </aside>
        <footer>
            <p>© 2014 Front End User Groups</p>
        </footer>
    </body>
</html>
```

Location Object

- The Location object represents the URL loaded into the window.

Navigator Object

- The Navigator object is implemented in almost every scriptable browser, even though its name is reminiscent of the Netscape Navigator-branded browser.

Document Object

- The Document object holds the real contents of the page.
- Properties and methods of the document generally affect the look and content of the document that occupies the window.
- All W3C DOM compatible browsers allow script access to the text contents of a page when the document is loaded.

History Object

- The browser maintains a list of URL’s for the most recent stop.
- This list is represented in the scriptable object model by the ***History Object.***
- The History object is part of the window object and is accessed through the ***window.history*** property.

Screen Object

- This is another read-only object that lets the script learn about the physical environment in which the browser is running.
- This object reveals the number of pixels high and wide available in the monitor.

Topics

- HTML5 Documents
- HTML5 Forms
  - Reading Data from Forms
  - Validating HTML5 Form Input
  - Sending Data from Forms
- Controlling HTML5 events with JavaScript
  - Writing `onClick` Events
- Document Object Model (DOM)

Try this section in JSConsole.

HTML5 Sections
Divide the document by sections.

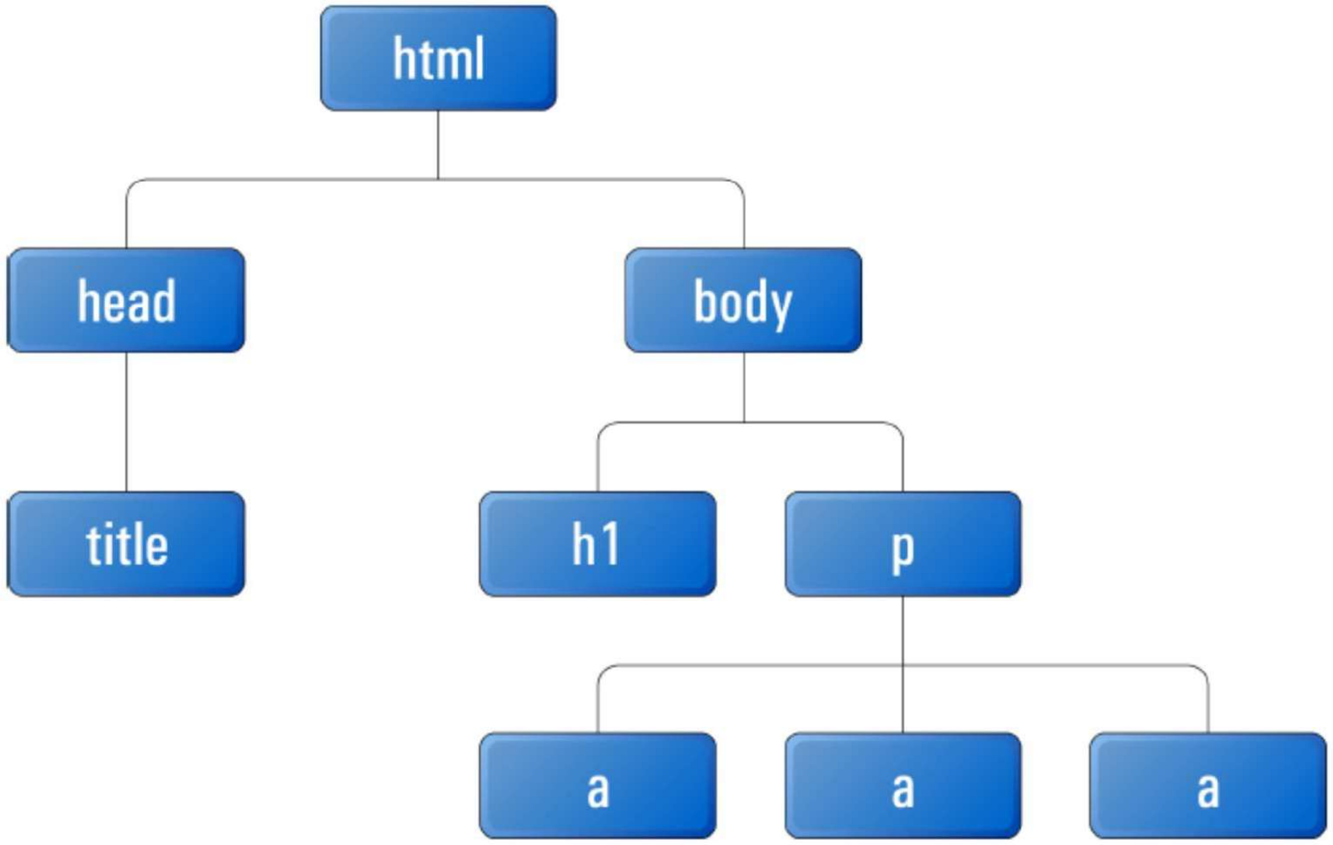
The Document Object Model:

- When an HTML document is downloaded to your browser, that browser has to do the job of turning what is essentially one long string of characters into a web page.
- To do this, the browser decides which parts are paragraphs, which parts are headings, which parts are text, and so on.
- The browser stores its interpretation of the HTML code as a structure of JavaScript objects, called the **Document Object Model**, or **DOM**.
- Within this model, each element in the HTML document becomes an object, as do all the attributes and text. JavaScript can access each of these objects independently

Example

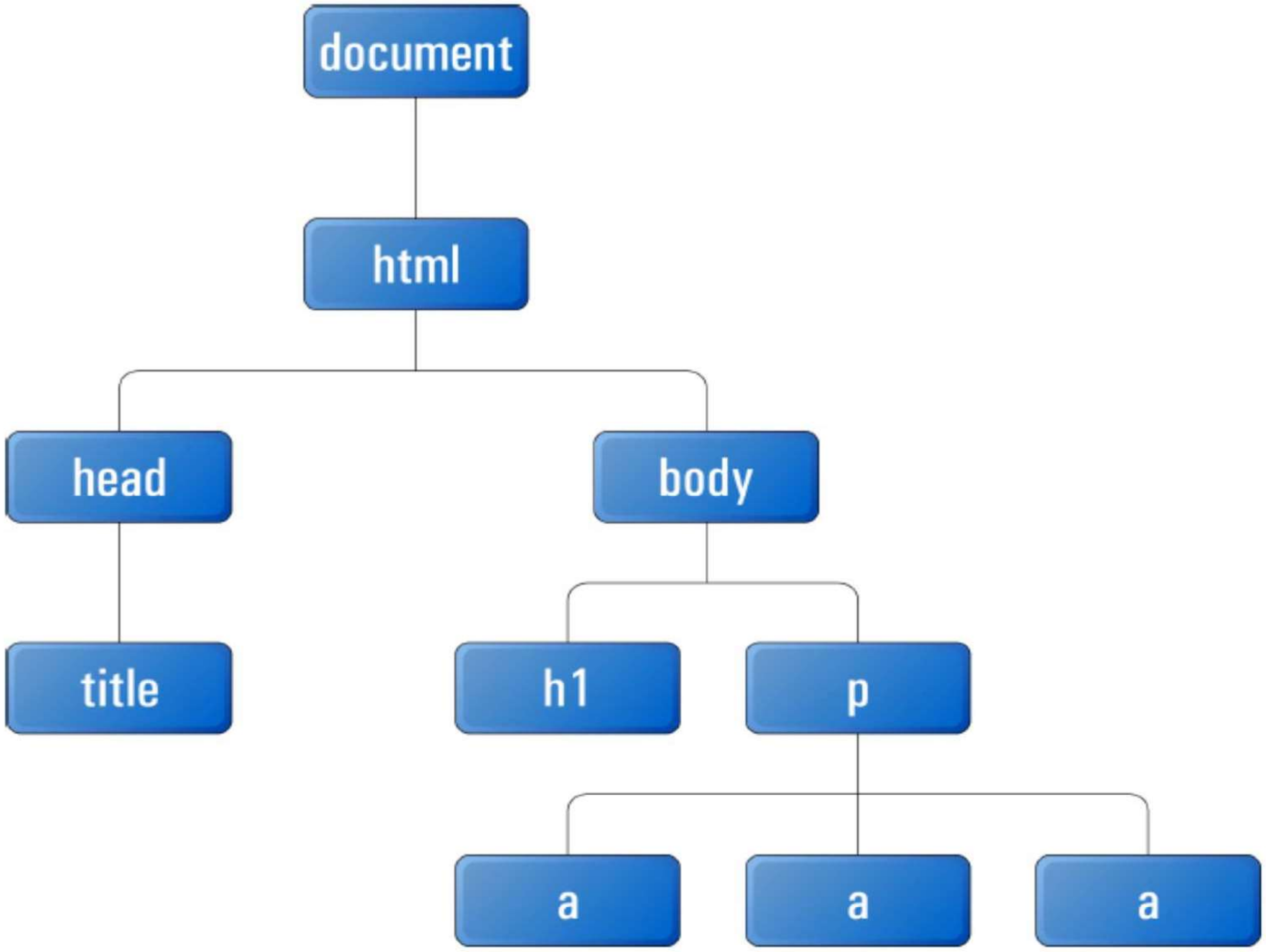
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US">
  <head>
    <title>DOMinating JavaScript</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8" />
  </head>
  <body>
    <h1>
      DOMinating JavaScript
    </h1>
    <p>
      If you need some help with your JavaScript, you might like
      to read articles from <a href="http://www.danwebb.net/"
          rel="external">Dan Webb</a>,
      <a href="http://www.quirksmode.org/" rel="external">PPK</a>
      and <a href="http://adactio.com/" rel="external">Jeremy
      Keith</a>.
    </p>
  </body>
</html>
```

These elements, as mapped out in the DOM



- To create the DOM for a document, each element in the HTML is represented by what’s known as a **node**.
- A node’s position in the DOM tree is determined by its parent and child nodes.
- An element node is distinguished by its element name (head, body, h1, etc.), but this doesn’t have to be unique. Unless you supply some identifying characteristic— like an **id attribute**

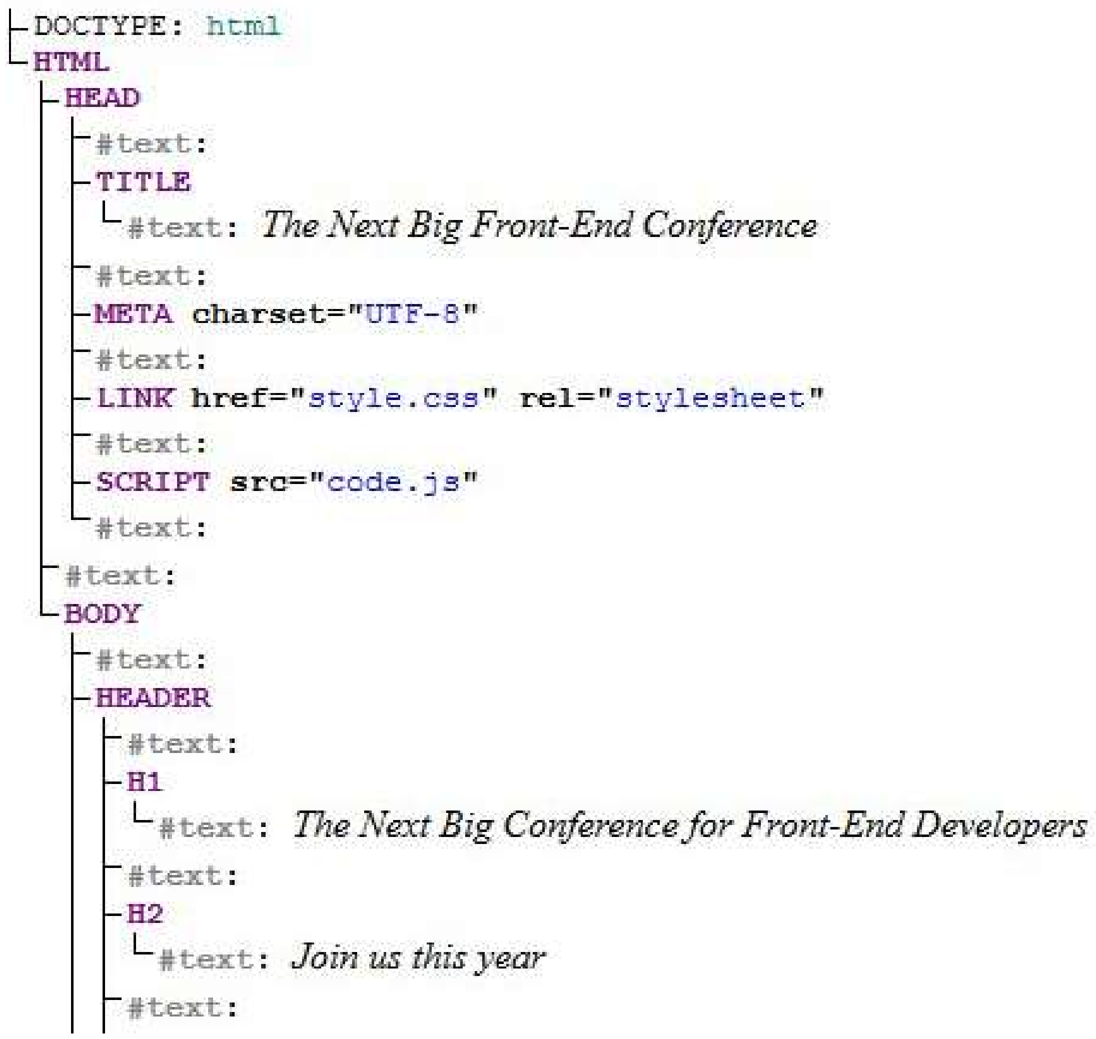
Root Node



Document Node

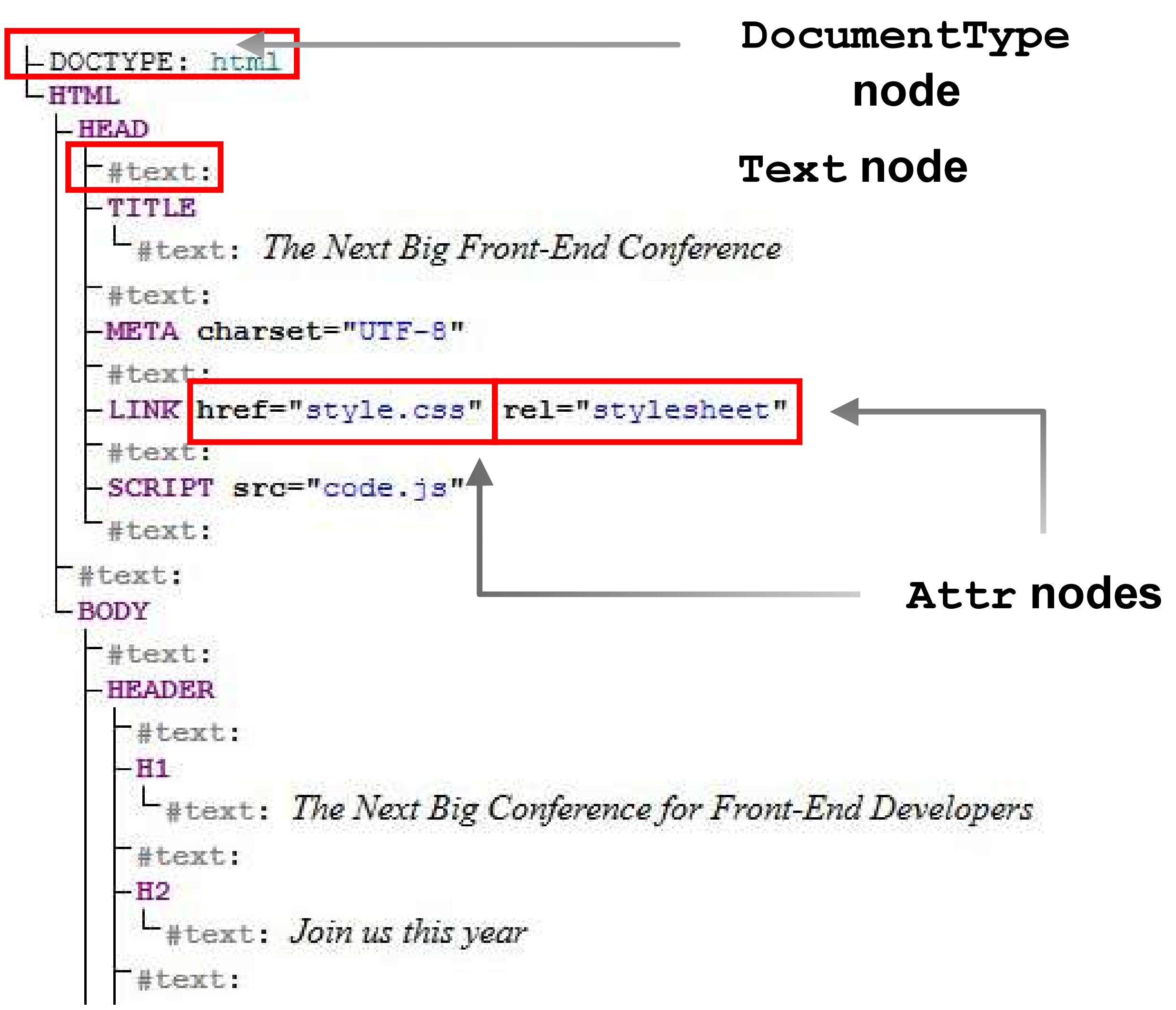
- *The content* of a document is contained in two other types of nodes:
  - text nodes
  - attribute nodes

The `Element` Node



```
├DOCTYPE: html
└HTML
  ├HEAD
  │ ├#text:
  │ ├TITLE
  │ │ └#text: The Next Big Front-End Conference
  │ ├#text:
  │ ├META charset="UTF-8"
  │ ├#text:
  │ ├LINK href="style.css" rel="stylesheet"
  │ ├#text:
  │ ├SCRIPT src="code.js"
  │ └#text:
  ├#text:
  └BODY
    ├#text:
    ├HEADER
    │ ├#text:
    │ ├H1
    │ │ └#text: The Next Big Conference for Front-End Developers
    │ ├#text:
    │ ├H2
    │ │ └#text: Join us this year
    │ ├#text:
```

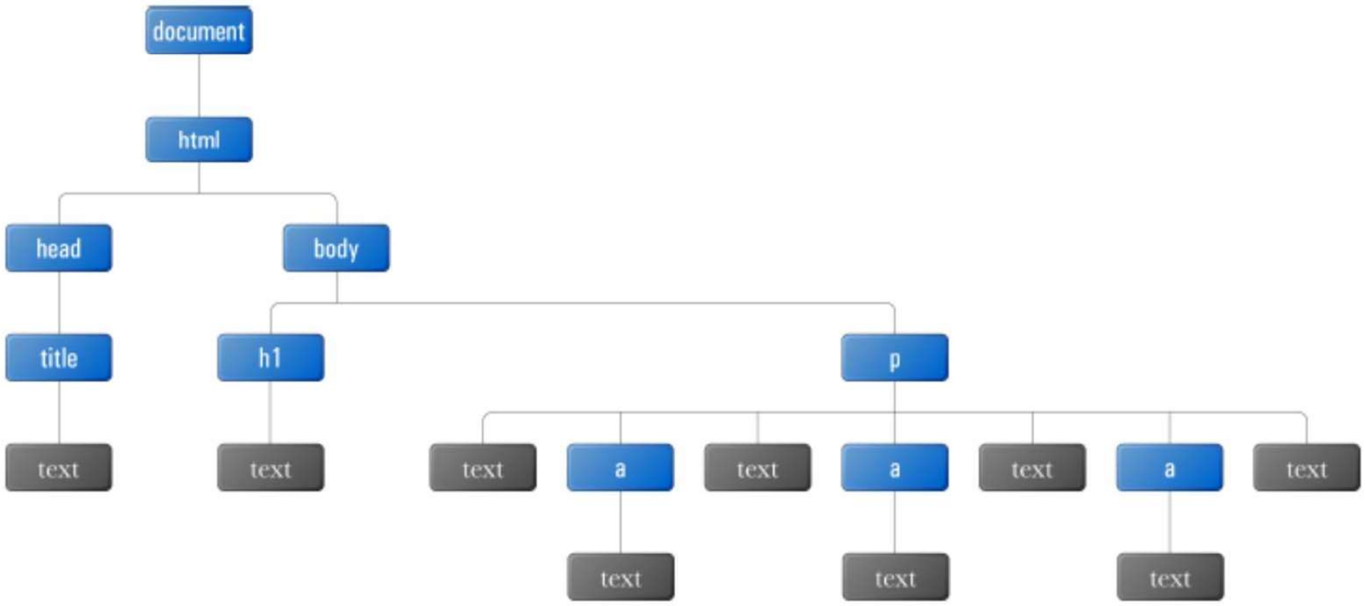
Other Element Nodes



```
├DOCTYPE: html
└HTML
  ├HEAD
  │ ├#text:
  │ ├TITLE
  │ │ └#text: The Next Big Front-End Conference
  │ ├#text:
  │ ├META charset="UTF-8"
  │ ├#text:
  │ ├LINK href="style.css" rel="stylesheet"
  │ ├#text:
  │ ├SCRIPT src="code.js"
  │ └#text:
  ├#text:
  └BODY
    ├#text:
    ├HEADER
    │ ├#text:
    │ ├H1
    │ │ └#text: The Next Big Conference for Front-End Developers
    │ ├#text:
    │ ├H2
    │ │ └#text: Join us this year
    │ ├#text:
```

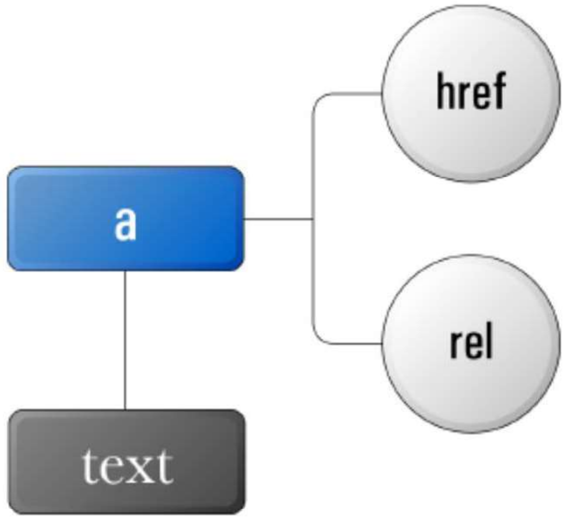
Text Nodes

- In HTML code, anything that’s not contained between angled brackets will be interpreted as a **text node** in the DOM.
- They cannot have children



Attribute Nodes

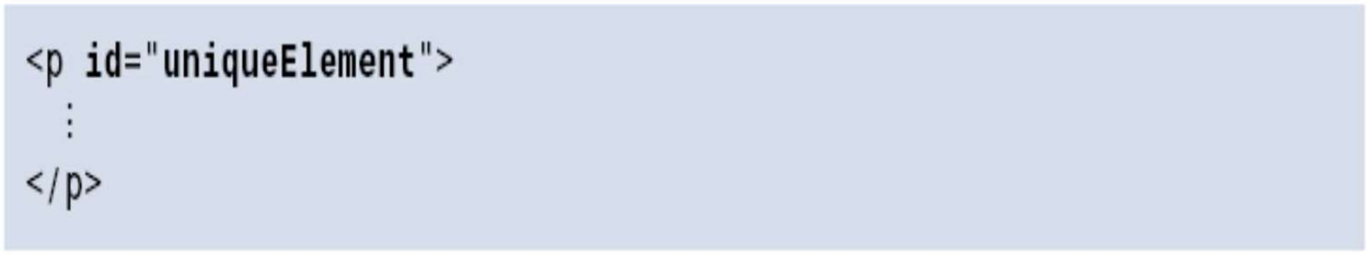
- With tags and text covered by element and text nodes, the only pieces of information that remain to be accounted for in the DOM are attributes.



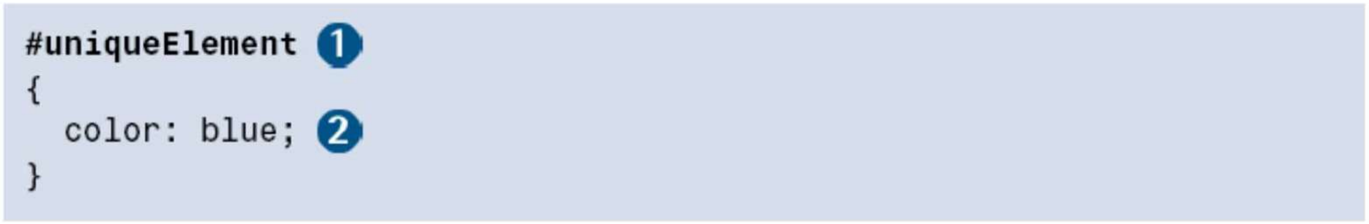
Accessing the Nodes you Want

•Finding an Element by ID

- The most direct path to an element is via its id attribute. id is an optional HTML attribute that can be added to any element on the page, but each ID you use has to be unique within that document:



```
<p id="uniqueElement">
  :
</p>
```



```
#uniqueElement ❶
{
  color: blue; ❷
}
```

Example

```
<h1>
  Sniper (1993)
</h1>
<p>
  In this cinema masterpiece,
  <a id="berenger" href="/name/nm0000297/">Tom Berenger</a> plays
  a US soldier working in the Panamanian jungle.
</p>
```

```
var target = document.getElementById("berenger");
alert(target.nodeName);
```

Finding Elements by Tag Name

- Using IDs to locate elements is excellent if you want to modify one element at a time, but if you want to find a group of elements, **getElementsByTagName**
- Unlike **getElementById, getElementsByTagName** can be executed as a method of any element node

Example

```
<!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" lang="en-US">
  <head>
    <title>Tag Name Locator</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8" />
  </head>
  <body>
    <p>
      There are 3 different types of element in this body:
    </p>
    <ul>
      <li>
        paragraph
      </li>
      <li>
        unordered list
      </li>
      <li>
        list item
      </li>
```

```
var listItems = document.getElementsByTagName("li");
```

Returns the Collection

```
var listItems = document.getElementsByTagName("li");

for (var i = 0; i < listItems.length; i++)
{
  alert(listItems[i].nodeName);
}
```

Looking at All the Elements

- To get all the elements in the document. We do this using
- getElementsByTagName, but we’re not going to look for a particular tag; instead, we’re going to pass this method the special value **"*"**

```
var elementArray = [];

if (typeof document.all != "undefined")
{
  elementArray = document.all;
}
else
{
  elementArray = document.getElementsByTagName("*");
}
```

Finding a Parent

- Every element node—except for the document node—has a parent. Consequently, each element node has a property called parentNode.
- When we use this property, we receive a reference to the target element's parent.

```
<p>
  <a id="oliver" href="/oliver/">Oliver Twist</a>
</p>
```

Once we have a reference to the anchor element, we can get a reference to its parent paragraph using `parentNode` like so:

```
var oliver = document.getElementById("oliver");
var paragraph = oliver.parentNode;
```

Finding Children

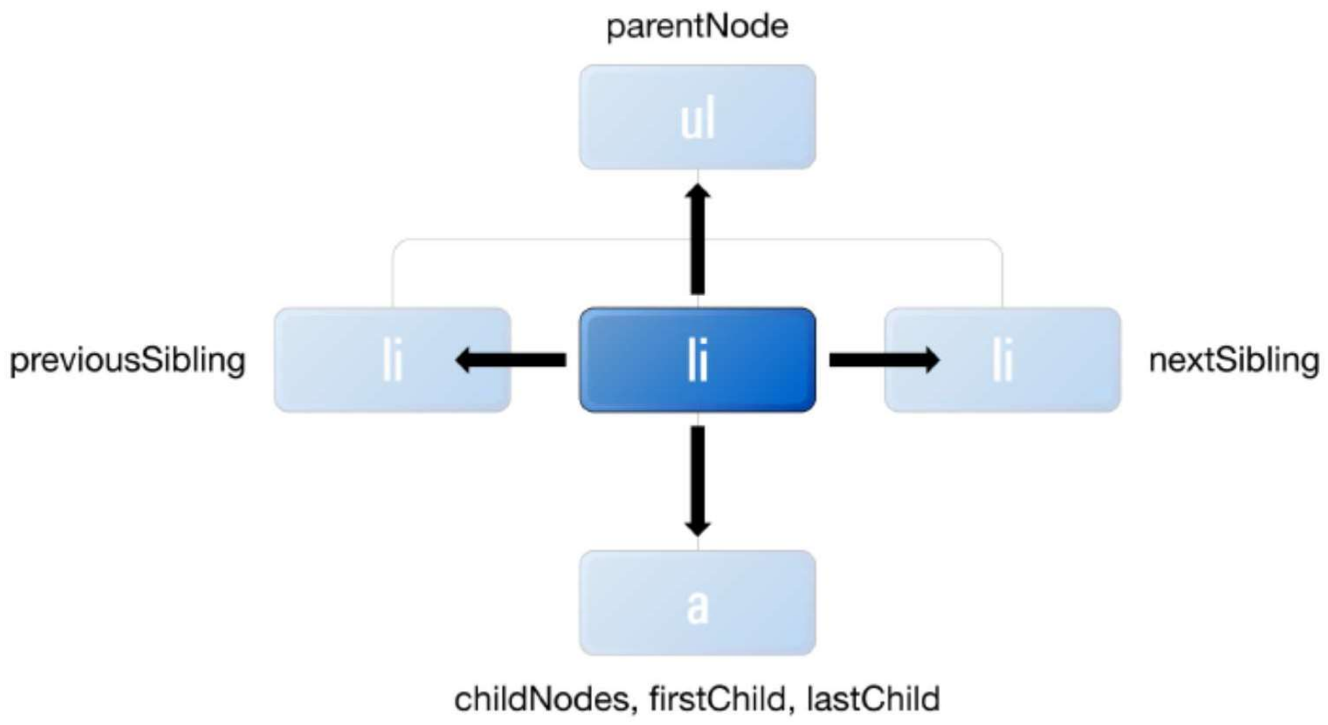
- The parent-child relationship isn't just one way. You can find all of the children of an element using the childNodes property.

```
<ul id="baldwins">
  <li>
    Alec
  </li>
  <li>
    Daniel
  </li>
  <li>
    William
  </li>
  <li>
```

```
var baldwins = document.getElementById("baldwins");
var william = baldwins.childNodes[2];
```

```
var alec = baldwins.firstChild;
```

```
var stephen = baldwins.lastChild;
```


parentNode
ul
previousSibling
li
li
li
nextSibling
a
childNodes, firstChild, lastChild

Getting an Attribute

- With a reference to an element already in hand, you can get the value of one of its attributes by calling the method **getAttribute** with the attribute name as an argument.

```
<a id="koko" href="http://www.koko.org/">Let's all hug Koko</a>
```

```
var koko = document.getElementById("koko");
var kokoHref = koko.getAttribute("href");
```

Setting an Attribute

- As well as being readable, all HTML attributes are writable via the DOM.
- To write an attribute value, we use the **setAttribute** method on an element, specifying both the attribute name we want to set and the value we want to set it to

```
var koko = document.getElementById("koko");
koko.setAttribute("href", "/koko/");
```

```
var koko = document.getElementById("koko");
koko.setAttribute("title", "Web site of the Gorilla Foundation");
```

Accessing and Manipulating Elements

- createElement:

```
var newParagraphElement = document.createElement("p");
var text = "This is the new Text";
var newTextElement = document.createTextNode(text);
newParagraphElement.appendChild(newTextElement);
```

- replaceChild:

```
parentNode.replaceChild(newParagraphElement,
    currentParagraphElement);
```

- removeChild:

```
parentNode.removeChild(currentParagraphElement);
```

- cloneNode:

```
var clonedNode =
    currentParagraphElement.cloneNode(true);
```

Where Can I Learn More?

Resource	Website
HTML5 W3C Specification	http://www.w3.org/TR/html5/
W3C Document Object Model	http://www.w3.org/DOM/
Live DOM Viewer	http://software.hixie.ch/utilities/js/live-dom-viewer/

Summary

In this lesson, you should have learned how to:

- Create HTML5 documents
- Create HTML5 forms to request information and process it
- Validate HTML5 forms input
- Write JavaScript functions for HTML5 events
- Manipulate HTML5 elements through DOM