Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar

Chapter 9

JavaScript 2:

Using JavaScript

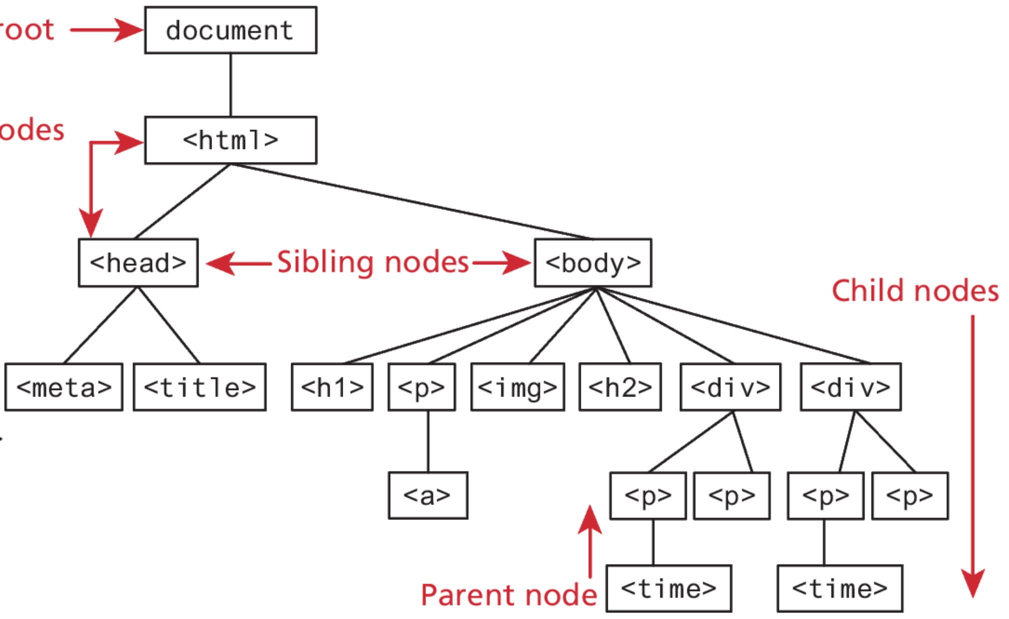
The DOM

The **Document Object Model** (*DOM*) is the data representation of the objects that comprise the structure and content of a document on the web.

Recall HTML defines the structure and content of a web page. The Document Object Model (DOM) is a programming interface for these web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

The Document Object Model (DOM)

```
<html>
<head>
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="#">Central Park</a>
   </p>
   <img src="park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>Ricardo on <time>2021-05-23</time</p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>Susan on <time>2021-11-18</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

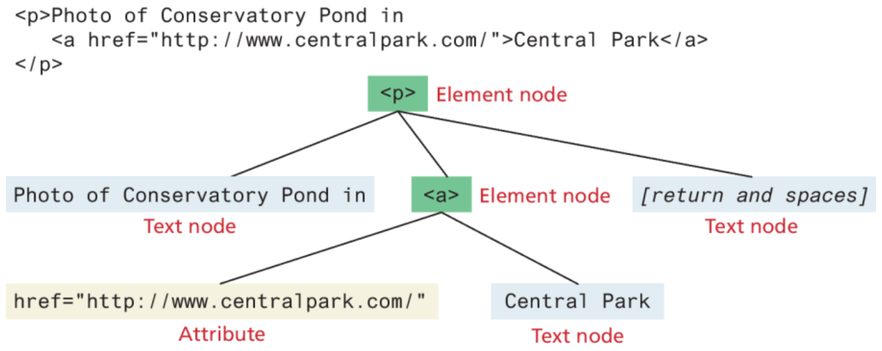


DOM Nodes and NodeLists

In the DOM, each element within the HTML document is called a **node**.

The DOM also defines a specialized object called a **NodeList** that represents a collection of nodes. It operates very similarly to an array.

Many programming tasks that we typically perform in JavaScript involve finding one or more nodes and then modifying them.



```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```

Some Essential Node Object Properties

- **childNodes** A NodeList of child nodes for this node
- **firstChild** First child node of this node
- **lastChild** Last child of this node
- **nextSibling** Next sibling node for this node
- **nodeName** Name of the node
- **nodeType** Type of the node
- **nodeValue** Value of the node
- **parentNode** Parent node for this node
- **previousSibling** Previous sibling node for this node
- **textContent** Represents the text content (stripped of any tags) of the node

Document Object

The **DOM document object** is the root JavaScript object representing the entire HTML document. It is globally accessible via the **document** object reference.

The properties of a document cover information about the page. Some are read-only, but others are modifiable. Like any JavaScript object, you can access its properties using either dot notation or square bracket notation

```
// retrieve the URL of the current page
let a = document.URL;
// retrieve the page encoding, for example ISO-8859-1
let b = document["inputEncoding"];
```

Document Methods

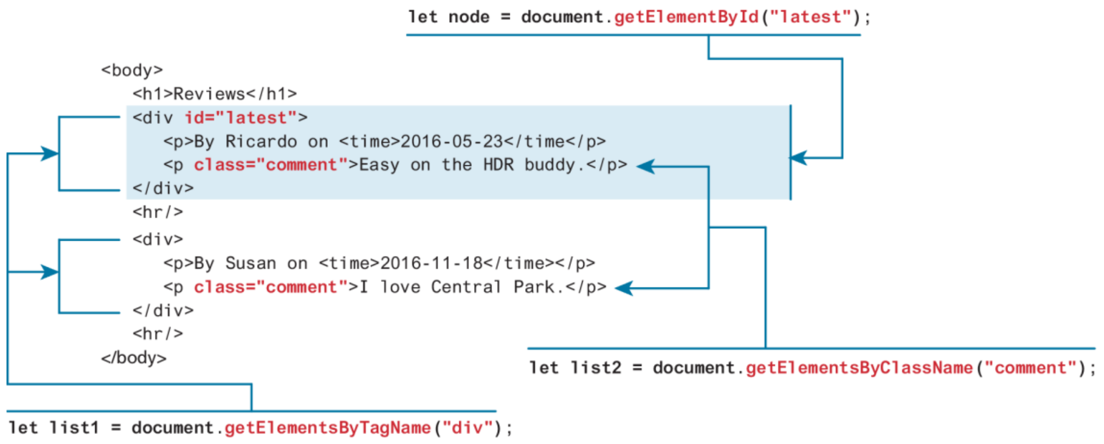
In addition to these properties, there are several essential methods you will use all the time (We used **document.write()**). These methods fall into three categories

- Selection methods
- Family manipulation methods
- Event methods

Selection Methods

The most important DOM methods

They allow you to select one or more document elements. The oldest 3 are: **getElementById("*id*"), getElementsByClassName("*name*")** and **getElementsByTagName("*name*")**



```
let node = document.getElementById("latest");
```

```
<body>
   <h1>Reviews</h1>
   <div id="latest">
      <p>By Ricardo on <time>2016-05-23</time></p>
      <p class="comment">Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p class="comment">I love Central Park.</p>
   </div>
   <hr/>
</body>
```

```
let list2 = document.getElementsByClassName("comment");
```

```
let list1 = document.getElementsByTagName("div");
```

Accessing elements and their properties

```
<p id="here">hello <span>there</span></p>
<ul>
    <li>France</li>
    <li>Spain</li>
    <li>Thailand</li>
</ul>
<div id="main">
    <a href="somewhere.html">
        <img src="whatever.gif" class="thumb">
    </a>
</div>

<script>

const node = document.getElementById("here");
console.log(node.innerHTML); // hello <span>there</span>
console.log(node.textContent); //"hello there"

const items = document.getElementsByTagName("li");
for (let i=0; i<items.length; i++) {
    // outputs: France, then Spain, then Thailand
    console.log(items[i].textContent);
}


</script>
```

LISTING 9.1 Accessing elements and their properties

Modifying the DOM

Now that you can access some of the node and element properties you might be wondering how one can make use of some of these properties. Since most of the properties listed in the previous tables are all read and write, this means that they can be programmatically changed.

- Changing an Element’s Style
- Changing the content of any given element
- DOM Manipulation Methods

InnerHTML vs textContent vs DOM Manipulation

The previous slide illustrated how you can programmatically access the content of an element node through its innerHTML or textContent property. These properties can also be used to modify the content of any given element.

For instance, you could change the content of the <div> with id="main" using the following:

```
const div = document.getElementById("main");
div.innerHTML = '<a href="hello.html">Hello There!</a>';
```

To only change the text of the element use:

```
div.textContent =' Goodbye!';
```

InnerHTML vs textContent vs DOM Manipulation (ii)

Using **innerHTML** is generally discouraged (even though you will likely see many examples online that use these approaches) because they are potentially vulnerable to Cross-Site Scripting (XSS) attacks

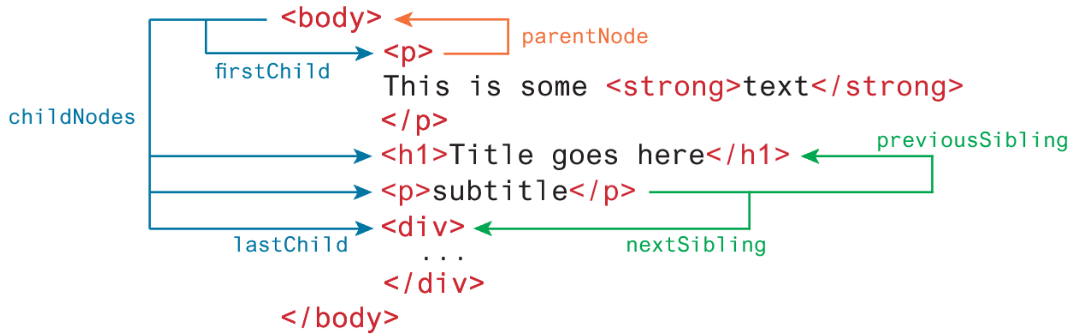
In practice, when you need to change the inner text of an element, it is preferable to use the **textContent** property instead of **innerHTML** since any markup is stripped from it.

In addition, when you need to generate HTML elements, it is better to use the appropriate DOM manipulation methods covered in the next section

DOM family relations

Each node in the DOM has a variety of “family relations” properties and methods for navigating between elements and for adding or removing elements from the document hierarchy.

Child and sibling properties can be an unreliable mechanism for selecting nodes and thus, in general, you will instead use selector methods



```
<body>
    <p>
    This is some <strong>text</strong>
    </p>
    <h1>Title goes here</h1>
    <p>subtitle</p>
    <div>
      ...
    </div>
</body>
```

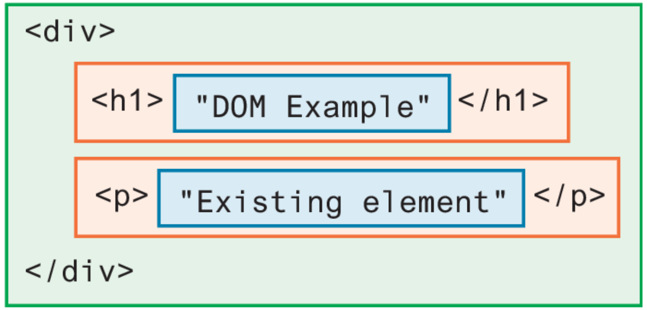
DOM Manipulation Methods

- **appendChild** Adds a new child node to the end of the current node.
- **createAttribute** Creates a new attribute node.
- **createElement** Creates an HTML element node.
- **createTextNode** Creates a text node.
- **insertAdjacentElement** Inserts a new child node at one of four positions relative to the current node.
- **insertAdjacentText** Inserts a new text node at one of four positions relative to the current node.
- **insertBefore** Inserts a new child node before a reference node in the current node.
- **removeChild** Removes a child from the current node.
- **replaceChild** Replaces a child node with a different child.

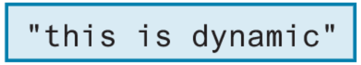
Visualizing the DOM modification

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
</div>
```

Visualizing the DOM elements

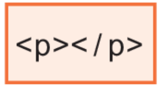


1 Create a new text node



```
const text = document.createTextNode("this is dynamic");
```

2 Create a new empty <p> element

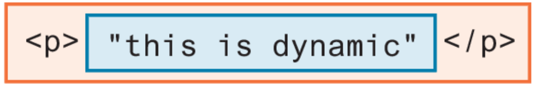


```
const p = document.createElement("p");
```

Visualizing the DOM modification (ii)

3 Add the text node to new <p> element

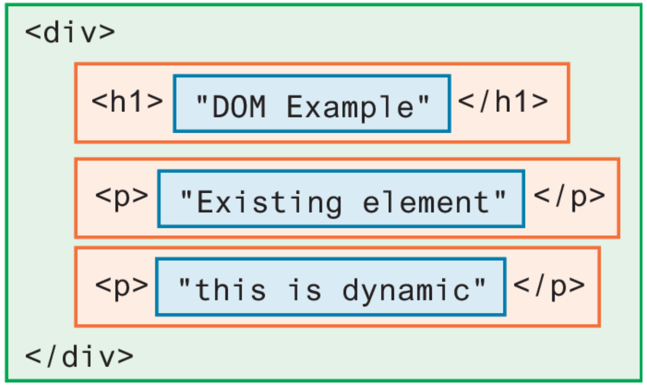
```
p.appendChild(text);
```



4 Add the <p> element to the <div>

```
const first = document.getElementById("first");
first.appendChild(p);
```

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
    <p>this is dynamic</p>
</div>
```



DOM Timing

Before finishing this section on using the DOM, it should be emphasized that the timing of any DOM code is very important.

You cannot access or modify the DOM until it has been loaded.

Modify the DOM within a function.

Copyright