CNU 국립 충남대학교 CHUNGNAM NATIONAL UNIVERSITY

7. JS HTML DOM

웹프로그래밍

2020년 2학기

충남대학교 컴퓨터공학과

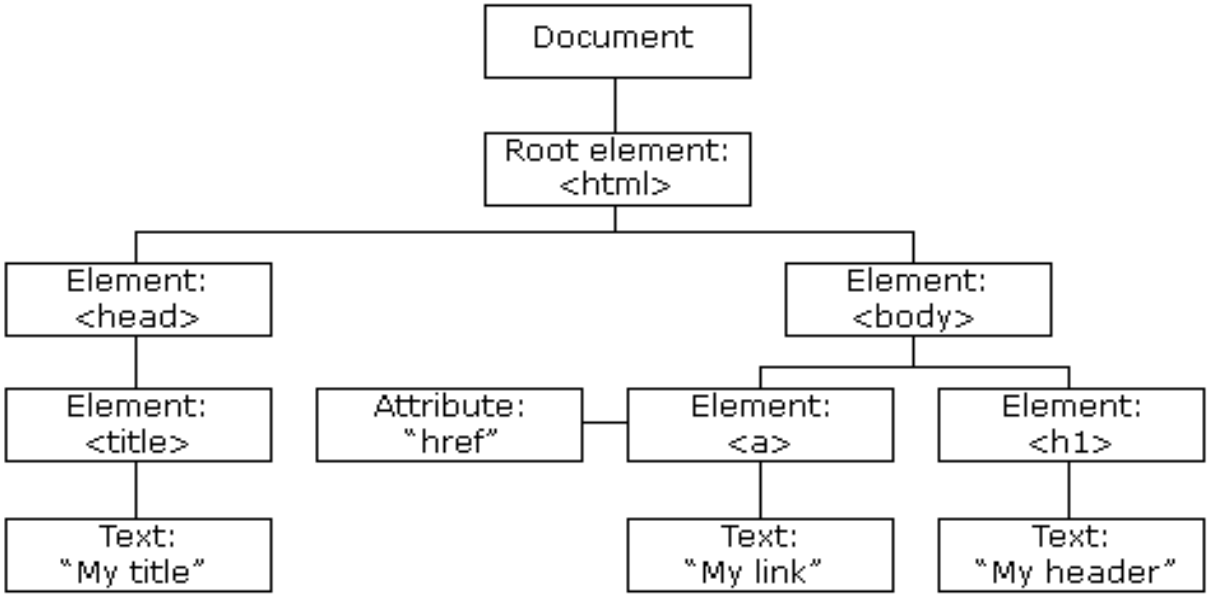
목차

- DOM Intro
- DOM Methods
- DOM Document
- DOM Elements
- DOM HTML
- DOM CSS
- DOM Events
- DOM Event Listener
- DOM Navigation
- DOM Nodes
- DOM Collections
- DOM Node Lists

JavaScript HTML DOM

- **The HTML DOM (Document Object Model)**
  - When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
  - The **HTML DOM** model is constructed as a tree of **Objects**

```
<html>
  <head>
    <title> My title</title>
  </head>
  <body>
     <a  href="" > My Link</a>
     <h1> My header</h1>
  </body>
</html>
```



JavaScript HTML DOM (cont'd)

- **With the object model, JavaScript gets all the power it needs to create dynamic HTML:**
  - JavaScript can change all the HTML elements in the page.
  - JavaScript can change all the HTML attributes in the page.
  - JavaScript can change all the CSS styles in the page.
  - JavaScript can remove existing HTML elements and attributes.
  - JavaScript can add new HTML elements and attributes
  - JavaScript can react to all existing HTML events in the page.
  - JavaScript can create new HTML events in the page.

JavaScript HTML DOM (cont'd)

- **What is the DOM?**
  - A W3C (World Wide Web Consortium) standard
  - Defines a standard for accessing documents
    - *"The W3C Document Object Model(DOM) is platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
  - Separated into 3 different parts
    - Core DOM – standard model for all document types
    - XML DOM – standard model for XML documents
    - HTML DOM – standard model for HTML documents

JavaScript HTML DOM (cont'd)

- **What is the HTML DOM?**
  - A standard **object** model and **programming interface** for HTML.
  - It defines:
    - The HTML elements as **objects**
    - The **properties** of all HTML elements
    - The **methods** to access all HTML elements
    - The **events** for all HTML elements
  - In other words : **The HTML DOM is a standard for how to get, change, add, or delete HTML elements**

JavaScript – HTML DOM Methods

- **The DOM Programming Interface** *Try it!*
  - The HTML DOM can be accessed with JavaScript (and with other programming languages).
  - In the DOM, all HTML elements are defined as **objects**.
  - The programming interface is the properties and methods of each object.
    - A **property** is a value that you can get or set (like changing the content of an HTML element).
    - A **method** is an action you can do (like add or deleting an HTML element).

JavaScript HTML DOM Document

- **The HTML DOM Document Object**
  - The document object represents your web page.
  - If you want to access objects in an HTML page, you always start with accessing the document object.

- **Finding HTML Elements**

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

JavaScript HTML DOM Document (cont'd)

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute(*attribute, value*)	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

JavaScript HTML DOM Document (cont'd)

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick=function() {*code*}	Adding event handler code to an onclick event

Finding HTML Objects *Refer to it!*

- The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.
- Later, in HTML DOM Level 3, more objects, collections, and properties were added.

JavaScript HTML DOM Elements

- **Finding HTML Elements**
  - There are several ways to do this
    - Finding HTML elements by id
    - Finding HTML elements by tag name
    - Finding HTML elements by class name
    - Finding HTML elements by CSS selectors
    - Finding HTML elements by HTML object collections

- **Finding HTML Element by Id** *Try it!*
  - The easiest way to find an HTML element in the DOM, is by using the element id
    - If the element is found, the method will return the element as an object (in myElement)
    - If the element is not found, myElement will contain **null**

JavaScript HTML DOM Elements (cont'd)

- **Finding HTML Elements by Tag Name** *Try it!* *Try it!*

- **Finding HTML Elements by Class Name** *Try it!*
  - If you want to find all HTML elements with the same class name, use **getElementsByClassName( )**
    - Finding elements by class name does not work in Internet Explorer 8 and earlier versions

- **Finding HTML Elements by CSS Selectors** *Try it!*
  - If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the **querySelectorAll( )** method
    - The **querySelectorAll( )** method does not work in Internet Explorer 8 and earlier versions

JavaScript HTML DOM Elements (cont'd)

- **Finding HTML Elements by HTML Object Collections** *Try it!*
  - The following HTML object (and object collections) are also accessible
    - document.anchors
    - document.body
    - document.documentElement
    - document.embeds
    - document.forms
    - document.head
    - document.images
    - document.links
    - document.scripts
    - document.title

- *Start the Exercise*

JavaScript HTML DOM – Changing HTML

- **Changing the HTML Output Stream**
  - In JavaScript, **document.write( )** can be used to write directly to the HTML output stream. *Try it!*
  - Never use **document.write(**) after the document is loaded. It will overwrite the document.

- **Changing HTML Content** *Try it!* *Try it!*
  - The easiest way to modify the content of an HTML element is by using the **innerHTML** property

```
document.getElementById(id).innerHTML = new HTML
```

JavaScript HTML DOM – Changing HTML (cont'd)

- **Changing the Value of an Attribute** *Try it!*

```
document.getElementById(id).attribute = new value
```

- *Start the Exercise*

JavaScript HTML DOM - Changing CSS

- **Changing HTML Style** *Try it!*

```
document.getElementById(id).style.property = new style
```

- **Using Events** *Try it!*
  - The HTML DOM allows you to execute code when an event occurs
  - Events are generated by the browser when "things happen" to HTML elements
    - An element is clicked on
    - The page has loaded
    - Input fields are changed

JavaScript HTML DOM - Changing CSS (cont'd)

- **More Examples**
  - Visibility *Try it!*
- *HTML DOM Style Object Reference*
- *Start the Exercise*

JavaScript HTML DOM Events

Reacting to Events

- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute.

onclick = *JavaScript*

- Examples of HTML events:
  - When a user clicks the mouse *Try it!* *Try it!*
  - When a web page has loaded
  - When an image has been loaded
  - When the mouse moves over an element
  - When an input field is changed
  - When an HTML form is submitted
  - When a user strokes a key

JavaScript HTML DOM Events (cont'd)

- **HTML Event Attributes** *Try it!*
  - To assign events to HTML elements you can use event attributes

- **Assign Events Using the HTML DOM** *Try it!*
  - The HTML DOM allows you to assign events to HTML elements using JavaScript

- **The onload and onunload Events** *Try it!*
  - The **onload** and **onunload** events are triggered when the user enters or leaves the page
  - The **onload** event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information
  - The **onload** and **onunload** events can be used to deal with cookies

JavaScript HTML DOM Events (cont'd)

- **The onchange Event** *Try it!*
  - The **onchange** event is often used in combination with validation of input fields
- **The onmouseover and onmouseout Events** *Try it!*
  - The **onmouseover** and **onmouseout** events can be used to trigger a function when the user mouses over, or out of, an HTML element
- **The onmousedown, onmouseup and onclick Events** *Try it!*
  - The **onmousedown**, **onmouseup**, and **onclick** events are all parts of mouse-click.
    - First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

JavaScript HTML DOM Events (cont'd)

- **More Examples**
  - onmousedown and onmouseup
    - Change an image when a user holds down the mouse button. *Try it!*
  - onload
    - Display an alert box when the page has finished loading. *Try it!*
  - onfocus
    - Change the background-color of an input field when it gets focus. *Try it!*
  - Mouse Events
    - Change the color of an element when the cursor moves over it. *Try it!*

- **HTML DOM Event Object Refernce**

The JavaScript this Keyword *Try it!*

- **What is this?**
  - The JavaScript **this** keyword refers to the object it belongs to
  - It has different values depending on where it is used
    - In a method, **this** refers to the owner object
    - Alone, **this** refers to the global object
    - In a function, **this** refers to the global object
    - In a function, in strict mode, **this** is **undefined**.
    - In an event, **this** refers to the element that received the event
    - Methods like **call( )**, and **apply( )** can refer **this** to any object
- **this in Event Handlers** *Try it!*
  - In HTML event handlers, **this** refers to the HTML element that received the event

```
<button onclick="this.style.display='none'  ">Click to Remove Me!</button>
```

JavaScript HTML DOM EventListener

- **The addEventListener( ) method** *Try it!*

```
<p id="demo"></p>

<script>
document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
```

JavaScript HTML DOM EventListener (cont'd)

- Attaches an event handler to the specified element
- Attaches an event handler to an element without overwriting existing event handlers
- You can add many event handlers to one element
- You can add many event handlers of the same type to one element, i.e two "click" events
- You can add event listeners to any DOM object not only HTML elements. i.e the window object
- Makes it easier to control how the event reacts to bubbling
- When using the **addEventListener( )** method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup
- You can easily remove an event listener by using the **removeEventListener( )** method

JavaScript HTML DOM EventListener (cont'd)

- **Syntax**

  element.addEventListener (*event*, *function*, *useCapture*);

  - First parameter is the type of the event (like "**click**", or "**mousedown**").
  - Second parameter is the function we want to call when the event occurs.
  - Third parameter is a boolean value specifying whether to use event bubbling or event capturing. (optional)
  - Note that you don't use the "on" prefix for the event; use "**click**" instead of "**onclick**"

- **Add an Event Handler to an Element** *Try it!* *Try it!*

JavaScript HTML DOM EventListener (cont'd)

- **Add Many Event Handlers to the Same Element** *Try it!* *Try it!*
  - The **addEventListener( )** method allows you to add many events to the same element, without overwriting existing events

- **Add an Event Handlers to the Window Object** *Try it!*
  - The **addEventListener( )** method allow you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the **xmlHttpRequest.object**

JavaScript HTML DOM EventListener (cont'd)

- **Passing Parameters** *Try it!*
  - When passing parameter values, use an "anonymous function" that calls the specified function with the parameters

```
<p id="demo"></p>
<script>
var p1 = 5;
var p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
  myFunction(p1, p2);
});

function myFunction(a, b) {
  var result = a * b;
  document.getElementById("demo").innerHTML = result;
}
</script>
```

JavaScript HTML DOM EventListener (cont'd)

- **Event Bubbling or Event Capturing?** ***Try it!***
  - Two ways of event propagation in the HTML DOM, bubbling and capturing
  - Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a element, and the user clicks on the <p> element, which element's "click" event should be handled first?
  - In bubbling
    - The inner most element's event is handled first and then the outer
    - The <p> element's click event is handled first, then the element's click event.
  - In capturing
    - The outer most element's event is handled first and then the inner
    - The element's click event will be handled first, then the <p> element's click event

JavaScript HTML DOM EventListener (cont'd)

- With the addEventListener( ) method you can specify the propagation type by using the "useCapture" parameter

```
addEventListener (event, function, useCapture);
```

  - The default value is false, which will use the bubbling propagation
  - When the value is set to true, the event uses the capturing propagation

The removeEventListener( ) method *Try it!*

- The **removeEventListener( )** method removes event handlers that have been attached with the addEventListener( ) method

JavaScript HTML DOM EventListener (cont'd)

- *HTML DOM Event Object Reference*
- *Start the Exercise*

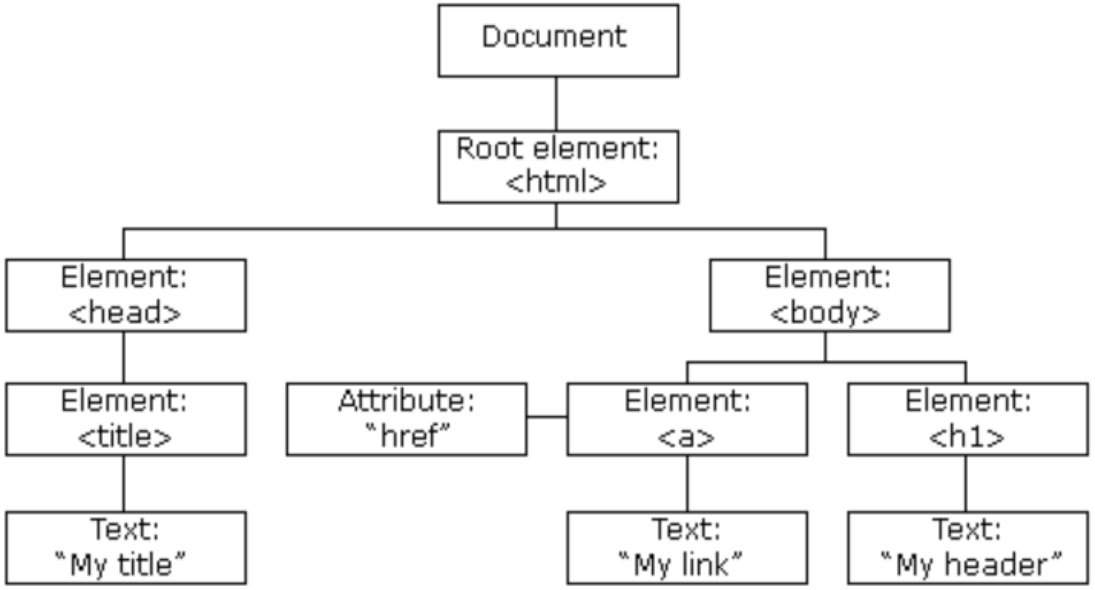
JavaScript HTML DOM Navigation

- You can navigate the node tree using node relationships.

DOM Nodes

- According to the W3C HTML DOM standard, everything in an HTML document is a node:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text inside HTML elements are text nodes
  - Every HTML attribute is an attribute node (deprecated)
  - All comments are comment nodes

JavaScript HTML DOM Navigation (cont'd)



- With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.
- New nodes can be created, and all nodes can be modified or deleted.

JavaScript HTML DOM Navigation (cont'd)

- **Node Relationships**
  - The nodes in the node tree have a hierarchical relationship to each other.
  - The terms parent, child, and sibling are used to describe the relationships.
    - In a node tree, the top node is called the root (or root node).
    - Every node has exactly one parent, except the root (which has no parent).
    - A node can have a number of children.
    - Siblings (brothers or sisters) are nodes with the same parent.

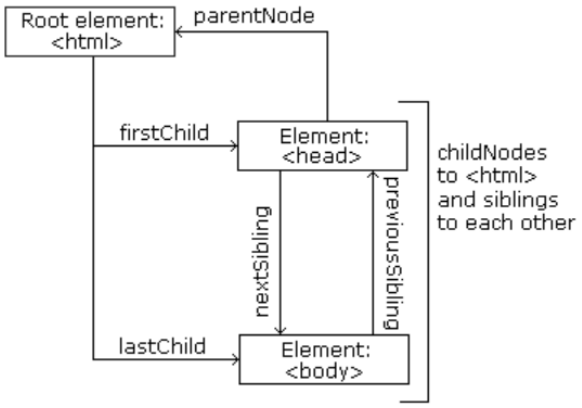
JavaScript HTML DOM Navigation (cont'd)

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



- From the HTML above you can read
  - **<html>** is the root node
  - **<html>** has no parents
  - **<html>** is the parent of **<head>** and **<body>**
  - **<head>** is the first child of **<html>**
  - **<body>** is the last child of **<html>**

JavaScript HTML DOM Navigation (cont'd)

- and
  - **<head>** has one child **: <title>**
  - **<title>** has one child (a text node) : "DOM Tutorial"
  - **<body>** has two children **: <h1>** and **<p>**
  - **<h1>** has one child : "DOM Lesson one"
  - **<p>** has one child "Hello world!"
  - **<h1>** and **<p>** are siblings

Navigating Between Nodes

- Node properties to navigate between nodes with JavaScript:
  - **parentNode , childNodes[*nodenumber*] , firstChild, lastChild, nextSibling, previousSibling**

JavaScript HTML DOM Navigation (cont'd)

- **Child Nodes and Node Values** *Try it! Try it! Try it!*
  - A common error in DOM processing is to expect an element node to contain text

Example:

```
<title id="demo">DOM Tutorial</title>
```

    - The element node **<title>** (in the example above) does **not** contain text
    - It contains a **text node** with the value "DOM Tutorial".
  - The value of the text node can be accessed by the node's **innerHTML** property

```
var myTitle = document.getElementById("demo").innerHTML;
```

  - Accessing the innerHTML property is the same as accessing the **nodeValue** of the first child

```
var myTitle = document.getElementById("demo").firstChild.nodeValue;
```

JavaScript HTML DOM Navigation (cont'd)

- Accessing the first child can also be done like this

```
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

DOM Root Nodes

- Two special properties that allow access to the full document.
  - **document.body** - the body of the document *Try it!*
  - **document.documentElement** - the full document *Try it!*

JavaScript HTML DOM Navigation (cont'd)

- **The nodeName property** *Try it!*
  - Specifies the name of a node.
    - nodeName is read-only.
    - nodeName of an element node is the same as the tag name.
    - nodeName of an attribute node is the attribute name.
    - nodeName of a text node is always #text.
    - nodeName of the document node is always #document.
  - **Note: nodeName** always contains the uppercase tag name of an HTML element.

JavaScript HTML DOM Navigation (cont'd)

- **The nodeValue property**
  - Specifies the value of a node.
    - nodeValue for element nodes is **null**
    - nodeValue for text nodes is the text itself
    - nodeValue for attribute nodes is the attribute value

JavaScript HTML DOM Navigation (cont'd)

- **The nodeType property** *Try it!*
  - Returns the type of node. nodeType is read-only.
  - The most important node types are:

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

  - Type 2 is deprecated in the HTML DOM (but works). It is not deprecated in the XML DOM

JavaScript HTML DOM Elements (Nodes)

- Adding and Removing Nodes (HTML Elements)

Creating New HTML Elements (Nodes) *Try it!*

1. Frist, create the element (element node).
2. Then append it to an existing element.

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

결과

This is a paragraph.

This is another paragraph.

This is new.

JavaScript HTML DOM Elements (Nodes) (cont'd)

Creating New HTML Elements – insertBefore ( )

- The **appendChild( )** method appended the new element as the last child of the parent.
- If you don't want you can use the **insertBefore( )** method *Try it!*

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

결과

This is new.

This is a paragraph.

This is another paragraph.

JavaScript HTML DOM Elements (Nodes) (cont'd)

Removing Existing HTML Elements

- To remove an HTML element, use the **remove( )** method *Try it!*

The `remove()` method does not work in older browsers, see the example below on how to use `removeChild()` instead.

```
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<button onclick="myFunction()">Remove Element</button>

<script>
function myFunction() {
  var elmnt = document.getElementById("p1");
  elmnt.remove() ;
}
</script>
```

JavaScript HTML DOM Elements (Nodes) (cont'd)

Removing a Child Node

- For browsers that does not support the **remove( )** method, you have to fine the parent node to remove an element *Try it!*

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

결과

This is another paragraph.

JavaScript HTML DOM Elements (Nodes) (cont'd)

Replacing HTML Elements *Try it!*

- To replace an element to the HTML DOM, use the **replaceChild( )** method

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>
```

결과

This is new.

This is another paragraph.

JavaScript HTML DOM Collections

The HTMLCollection Object

- The **getElementsByTagName( )** method returns an **HTMLCollection** object
- An **HTMLCollection** object is an array-like list(collection) of HTML elements

Example

```
var x = document.getElementsByTagName("p");
```

The elements in the collection can be accessed by an index number.

To access the second <p> element you can write:

```
y = x[1];
```

- Note : The index starts at o

JavaScript HTML DOM Collections (cont'd)

- **HTML HTMLCollection Length**
  - The **length** property defines the number of elements in an **HTMLCollection** *Try it!*
  - The **length** property is useful when you want to loop through the elements in a collection *Try it!*
  - **An HTMLCollection is NOT an array!**
    - An HTMLCollection may look like an array, but it is not.
    - You can loop through the list and refer to the elements with a number (just like an array).
    - However, you cannot use array methods like valueOf(), pop(), push(), or join() on an HTMLCollection

JavaScript HTML DOM Node Lists

- **The HTML DOM NodeList Object** *Try it!*
  - A **NodeList** object is a list (collection) of nodes extracted from a document.
  - A **NodeList** objet is almost the same as an **HTMLCollection** object
  - All browsers return a NodeList object for the property **childNodes**
  - Most browsers return a NodeList object for the method **querySelectorAll( )**

Example

```
var myNodeList = document.querySelectorAll("p");
```

The elements in the NodeList can be accessed by an index number.

To access the second <p> node you can write:

```
y = myNodeList[1];
```

  - **Note :** The index starts at o

JavaScript HTML DOM Node Lists (cont'd)

- **HTML DOM Node List Length**
  - The **length** property defines the number of nodes in a node list *Try it!*
  - The **length** property is useful when you want to loop through the nodes in a node list ***Try it!***

- **The Difference Between an HTMLCollection and NodeList**
  - An **HTMLCollection** is a collection of HTML elements
  - A **NodeList** is a collection of document nodes
  - Both an HTMLCollection object and a NodeList object
    - An array-like list (collection) of objects
    - Have a length property defining the number of items in the list (collection)
    - Provide an index (0, 1, 2, 3, 4, ..) to access each item like an array

JavaScript HTML DOM Node Lists (cont'd)

- **HTMLCollection**
  - Can be accessed by their name, id, or index number
- **NodeList item**
  - Can only be accessed by their index number
  - Can contain attribute nodes and text nodes
- **A node list is not an array!**
  - A node list may look like an array, but it is not.
  - You can loop through the node list and refer to its nodes like an array.
  - However, you cannot use Array Methods, like valueOf(), push(), pop(), or join() on a node list.

참고 – 그 외 주제들

- JavaScript HTML DOM Animation