Web Application Development for Industrial Engineers

การพัฒนาเวปแอปพลิเคชันสำหรับวิศวกรอุตสาหการ

Document Object Model (DOM)

What is the DOM?

- The Document Object Model (DOM) is a programming interface for 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.

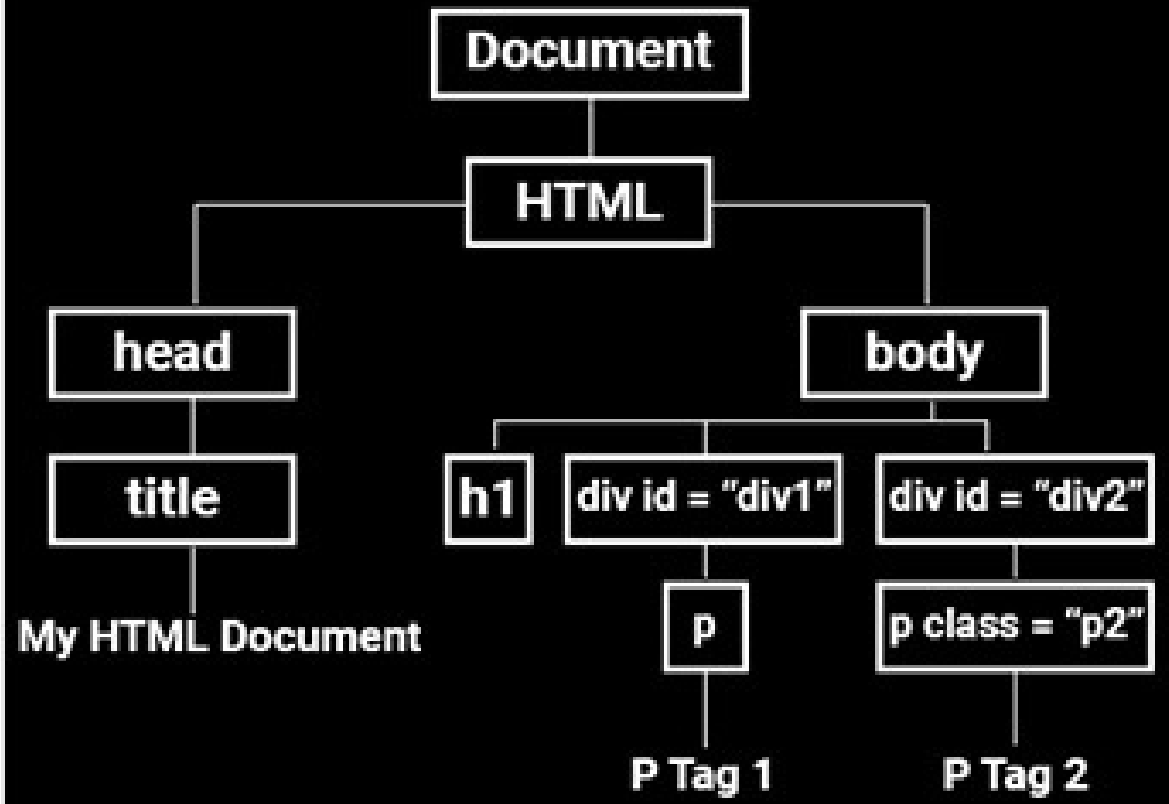
What is Document Object Model ?

HTML Document

index.html

```
1   <html>
2       <head>
3           <title>My HTML Document</title>
4       </head>
5
6       <body>
7           <h1>Heading</h1>
8           <div id="div1">
9               <p>P Tag 1</p>
10          </div>
11          <div id="div2">
12              <p class="p2">P Tag 2</p>
13          </div>
14      </body>
15  </html>
```

Document Object Model (DOM)



Data type

- `Document` : represents any web page loaded in the browser
- `Node` : represents an object located within a document.
  - `Element` : represents an element in HTML.
  - `TextNode` : specifies text in an element.
  - `Attr` : speicifies attributes of an element.
- `NodeList` : A nodeList is an array of elements.

DOM Navigation

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
  </head>
  <body>
    <h1>Headings</h1>
    <div id="div1">
      div1 text
      <p>p text</p>
    </div>
    <div id="div2">div2 text</div>
  </body>
</html>
```

```
// document object
console.log(document.childNodes);

// html
const html = document.childNodes[1];
console.log(html);

// head, body, text nodex
console.log(html.childNodes);
const head = html.childNodes[0];
const body = html.childNodes[2];
console.log(head);
console.log(body);
```

```
// #div1
console.log(body.childNodes);
const div1 = body.childNodes[3];
console.log(div1);

// We can change the text of #div1
div1.childNodes[0].textContent = 'Changed';

// Note that this is different from
// div1.textContent = 'Changed';
```

Other DOM Navigation APIs

```
parentNode;
childNodes;
firstChild;
lastChild;
nextSibling;
previousSibling;
```

DOM Manupulation

```
document.querySelector(selector);
document.querySelectorAll(name);
document.createElement(name);
parentNode.appendChild(node);
element.remove();
element.innerHTML;
element.innerText;
element.textContent;
element.style;
element.setAttribute();
element.getAttribute();
element.addEventListener();
```

`querySelector` and `querySelectorAll`

```
const div = document.querySelector('div');
const divs = document.querySelectorAll('div');
console.log(div);
console.log(divs);
```

`createElement` and `appendChild`

```
const div1 = document.querySelector('#div1');
const para = document.createElement('p');
para.innerText = 'New Text!';
div1.appendChild(para);
```

remove

```
const div1 = document.querySelector('#div1');
div1.remove();
```

`innerText`, `textContent`, `innerHTML`

```
const div1 = document.querySelector('#div1');
div1.innerText = 'New Text!';
div1.textContent = 'New Text!';
div1.innerHTML = '<p>New Text!</p>';
```

Difference between innerText and textContent

style

```
const div1 = document.querySelector('#div1');
console.dir(div1.style);
div1.style.backgroundColor = 'red';
```

`setAttribute`, `removeAttribute`, `getAttribute`

```
const div1 = document.querySelector('#div1');
btn = document.createElement('button');
btn.textContent = 'Click Me';
div1.appendChild(btn);

btn.setAttribute('disabled', null);
btn.removeAttribute('disabled');

btn.setAttribute('id', 'btnId');
const btnId = btn.getAttribute('id');
console.log(btnId);
```

Introduction to events

Event

- Events are *actions* that happen in the system you are programming, which the system tells you about.
  - So your code can react to them.
- For example, if the user clicks a button on a webpage, you might want to react to that action by displaying an information box.

Event type

- The user selects a certain element or hovers the cursor over a certain element.
- The user chooses a key on the keyboard.
- The user resizes or closes the browser window.
- A web page finishes loading.
- A form is submitted.
- A video is played, paused, or finishes.
- An error occurs.

Event type

- Event reference
- `Element` -> `click event`

Event handler

- To react to an event, you attach an event handler to it.
- This is a block of code that runs when the event fires.
- Event handlers are sometimes called *event listeners*.

Example

```
<button>Click Me</button>
```

Example

```
const btn = document.querySelector('button');

function random(number) {
  return Math.floor(Math.random() * (number + 1));
}

function clickHandler() {
  const rndCol = `rgb(${random(255)}, ${random(255)}, ${random(255)})`;
  document.body.style.backgroundColor = rndCol;
}

btn.addEventListener('click', clickHandler);
```

Adding multiple listeners

(Add to the above code)

```
function doubleClickHandler() {
  alert('Reset to white');
  document.body.style.backgroundColor = 'white';
}

btn.addEventListener('dblclick', doubleClickHandler);
```

Remove listeners

```
btn.removeEventListener('dblclick', doubleClickHandler);
```

Event object

- Sometimes, inside an event handler function, you'll see a parameter specified with a name such as `event`, `evt`, or `e`.
- This is called the **event object**, and it is automatically passed to event handlers to provide extra features and information.

Example

HTML

```
<div style="width: 16rem; height: 16rem; border: 1px solid gray"></div>
```

Example

JavaScript

```
const div = document.querySelector('div');

div.addEventListener('mousemove', onMouseMove);

function onMouseMove(e) {
  div.innerText = `${e.offsetX}, ${e.offsetY}`;
  div.style.backgroundColor = `rgb(${e.offsetX}, ${e.offsetY}, ${
    (e.offsetX + e.offsetY) / 2
  })`;
}
```

Event target

`event.target`

- The element that caused the event.
- Useful when you want to reuse the event handler.

```
<head>
  <style>
    body {
      display: flex;
      flex-wrap: wrap;
      gap: 1rem;
    }
    div {
      width: 8rem;
      height: 8rem;
      border: 1px solid gray;
      display: flex;
      justify-content: center;
      align-items: center;
    }
  </style>
</head>
```

```
<body>
  <div id="div1">Box1</div>
  <div id="div2">Box2</div>
  <div id="div3">Box3</div>
  <div id="div4">Box4</div>
  <div id="div5">Box5</div>
  <div id="div6">Box6</div>
</body>
```

```
const divs = document.querySelectorAll('div');

divs.forEach((div) => {
  div.addEventListener('mousemove', handler);
});

function handler(e) {
  e.target.innerText = `${e.offsetX}, ${e.offsetY}`;
  e.target.style.color = 'white';
  e.target.style.backgroundColor = `rgb(${e.offsetX}, ${e.offsetY}, ${
    (e.offsetX + e.offsetY) / 2
  })`;
}
```

Event bubbling

- When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.

```
<head>
  <style>
    body * {
      margin: 10px;
      border: 1px solid blue;
    }
  </style>
</head>
```

```
<body>
  <form id="form">
    FORM
    <div id="div">
      DIV
      <p id="p">P</p>
    </div>
  </form>
</body>
```

```
const formElement = document.querySelector('form');
const divElement = document.querySelector('div');
const pElement = document.querySelector('p');

formElement.addEventListener('click', () => handleClick('form'));
divElement.addEventListener('click', () => handleClick('div'));
pElement.addEventListener('click', () => handleClick('p'));

function handleClick(msg) {
  alert(`Click: "${msg}"`);
}
```

- Notice how we can pass the data to the handler.

Event target (during bubbling)

- The most deeply nested element that caused the event is called a target element, accessible as `event.target`.

```
const formElement = document.querySelector('form');
const divElement = document.querySelector('div');
const pElement = document.querySelector('p');

formElement.addEventListener('click', (e) => handleClick(e, 'form'));
divElement.addEventListener('click', (e) => handleClick(e, 'div'));
pElement.addEventListener('click', (e) => handleClick(e, 'p'));

function handleClick(e, msg) {
  const id = e.target.getAttribute('id');
  alert(`Click: "${msg}",  Event.target: "${id}"`);
}
```

- Notice how we can pass **both** the event object and data to the handler.

Stopping bubbling

```
function handleClick(e, msg) {
  const id = e.target.getAttribute('id');

  // Stop propagation
  e.stopPropagation();

  alert(`Click: "${msg}",  Event.target: "${id}"`);
}
```