JavaScript

Buổi 6: JavaScript DOM (Document Object Model)

Nội dung buổi 6

- Giới thiệu về DOM.
- Làm việc với phần tử HTML.
- Thay đổi thuộc tính phần tử HTML.
- DOM Events. Event Handler.
- Đối tượng Event.
- Xử lý lan truyền Event.

Giới thiệu về DOM

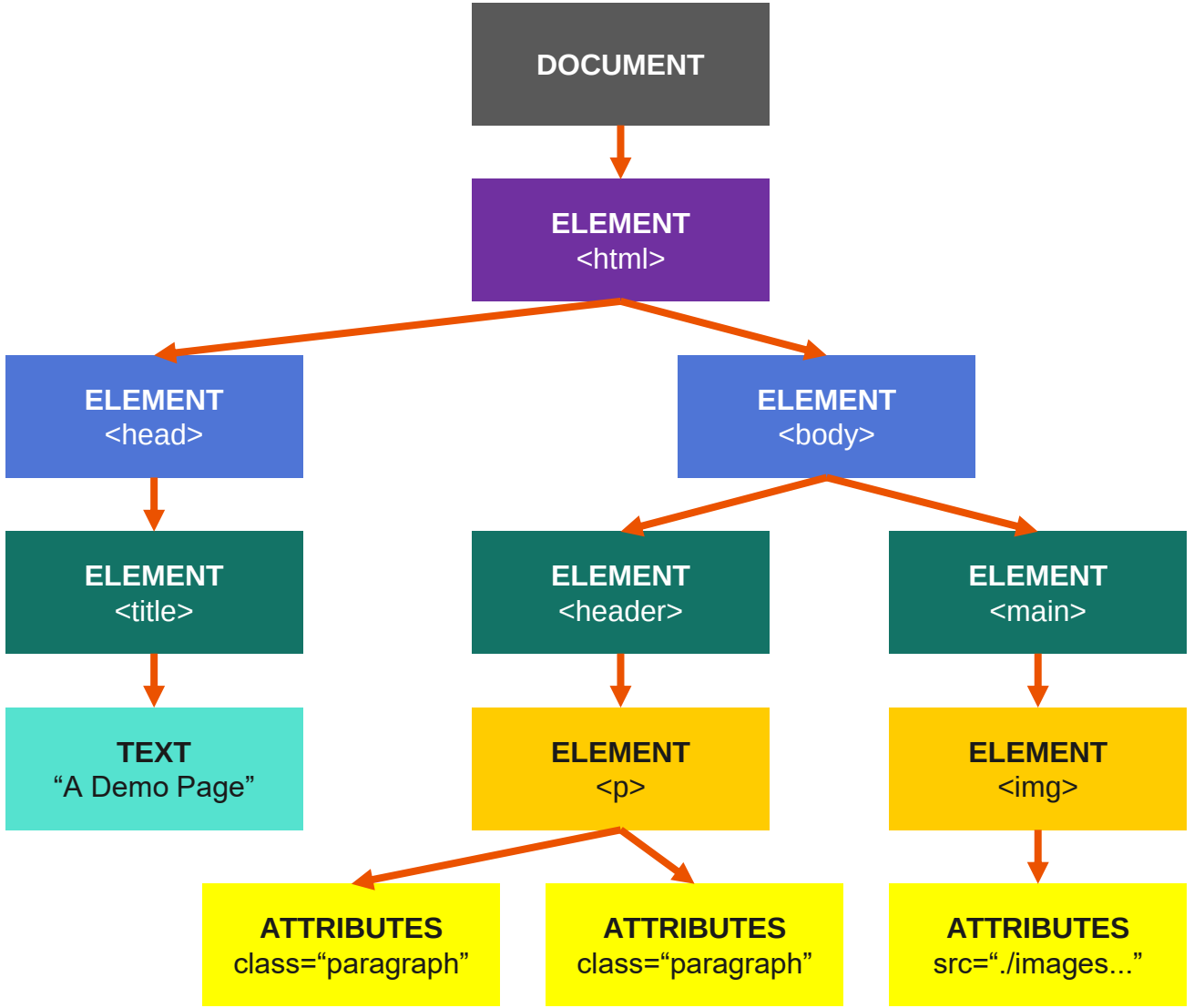
Giới thiệu về DOM

- DOM (Document Object Model) là một cây cấu trúc dữ liệu được dùng để truy xuất các dữ liệu từ một văn bản định dạng HTML. DOM cho phép JavaScript truy cập vào các phần tử HTML để thay đổi, tùy biến các thành phần của HTML và cả CSS.
- DOM không phải là JavaScript, các trình duyệt cung cấp các API (Application Programming Interface) để JavaScript có thể làm việc với DOM.

Giới thiệu về DOM

```html
<html>
  <head>
    <title>A Demo Page</title>
  </head>

  <body>
    <header>
      <p class="paragraph">Hello world!</p>
    </header>
    <main>
      <img src="./images/bay-max.jpg" alt="Image" />
    </main>
  </body>
</html>
```



Làm việc với phần tử HTML

Làm việc với phần tử HTML

DOM cho phép chúng ta truy cập và làm việc với các phần tử HTML thông qua các phương thức của Web API đặc biệt có tên là document.

Để lấy ra các phần tử HTML trong DOM có thể sử dụng các cách sau:

```
document.querySelector(".element");
document.querySelectorAll(".some-list li");
// => Trả về các phần tử trong một NodeList.

document.getElementById("the-id");
document.getElementsByTagName("p");
document.getElementsByClassName("class-name");
// => Trả về các phần tử trong một HTMLCollection.
```

Bài thực hành số 1

Bài thực hành số 1

Trong file HTML tạo các nội dung sau:

- Tạo một thẻ div có class là “text-content”, trong thẻ div này có 3 thẻ p chứa các nội dung khác nhau, mỗi thẻ p có tên class riêng.
- Tạo một list với thẻ ul, li và tạo một số content văn bản cho các thẻ li.

Sử dụng các phương thức của DOM để làm các bài sau.

Bài 1: In ra cửa sổ console thẻ p đầu tiên trong thẻ div có class là “text-content”.

Bài 2: In ra cửa sổ console thẻ p thứ hai trong thẻ div có class là “text-content”.

Bài 4: In ra cửa số console các thẻ p trong thẻ div có class là “text-content”. Đổi kết quả về dạng mảng và in ra cửa sổ console (Gợi ý, sử dụng spread operator).

Bài 3: In ra cửa sổ console các thẻ li có trong list. Đổi kết quả về dạng mảng và in ra cửa sổ console .

Thay đổi nội dung và thuộc tính của phần tử HTML

Thay đổi nội dung

Để lấy ra giá trị hoặc thay đổi nội dung của một phần tử HTML có thể sử dụng 2 thuộc tính là **textContent** và **innerHTML**

textContent: Sử dụng để lấy giá trị và gán giá trị mới cho nội dung văn bản (text) của một phần tử HTML.

```
<body>
  <p class="text">Content</p>

  <script src="./script.js"></script>
</body>
```

Content

```
const paragraphEl = document.querySelector(".text");

paragraphEl.textContent; // "Content"
paragraphEl.textContent = "New Content";
```

New Content

Thay đổi nội dung

innerHTML: Sử dụng để lấy toàn bộ nội dung content của phần tử HTML, kể cả cú pháp thẻ HTML. Nếu sử dụng thuộc tính này để gán giá trị mới cho phần tử HTML, có thể sử dụng cả cú pháp HTML.

```html
<body>
  <p class="text">
    <i>Text in Italic</i>
  </p>

  <script src="./script.js"></script>
</body>
```

Text in Italic

```js
const paragraphEl = document.querySelector(".text");

paragraphEl.innerHTML.trim();
// => "<i>Text in Italic</i>"

paragraphEl.innerHTML = "<b>Text in Bold</b>";
```

Text in Bold

Thay đổi class

Thay đổi tên class với các phương thức đặc biệt của thuộc tính “classList”:

Phương thức **add**: Thêm vào tên class giá trị tham số truyền vào.

JavaScript:

```
const paragraphEl = document.querySelector(".text");

paragraphEl.classList.add("active");
```

HTML:

```
<!-- Trạng thái ban đầu -->
<p class="text">Content</p>

<!-- Sau khi sử dụng phương thức add -->
<p class="text active">Content</p>
```

Thay đổi class

Thay đổi tên class với các phương thức đặc biệt của thuộc tính “classList”:

Phương thức **remove**: Xóa đi tên class theo giá trị tham số truyền vào.

JavaScript:

```
const paragraphEl = document.querySelector(".text");

paragraphEl.classList.remove("active");
```

HTML:

```
<!-- Trạng thái ban đầu -->
<p class="text active">Content</p>

<!-- Sau khi sử dụng phương thức remove -->
<p class="text">Content</p>
```

Thay đổi class

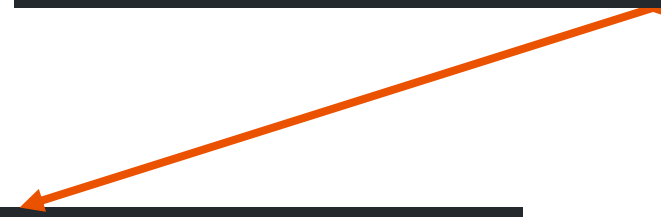
Thay đổi tên class với các phương thức đặc biệt của thuộc tính “classList”:

Phương thức **toggle**: Nếu phần tử HTML chưa có tên class giống tham số truyền vào thì thêm vào, có thì bỏ đi.

JavaScript:

```
const paragraphEl = document.querySelector(".text");

paragraphEl.classList.toggle("active");
```



HTML:

```
<!-- Trạng thái ban đầu -->
<p class="text">Content</p>

<!-- Sau khi sử dụng phương thức toggle -->
<p class="text active">Content</p>
```

HTML:

```
<!-- Trạng thái ban đầu -->
<p class="text active">Content</p>

<!-- Sau khi sử dụng phương thức toggle -->
<p class="text">Content</p>
```

Bài thực hành số 2

Bài thực hành số 2

Bài 1: Viết một hàm nhận 2 tham số là các giá trị chuỗi, tham số thứ nhất đại diện cho CSS Selector, tham số thứ 2 là nội dung text. Hàm tìm ra phần tử HTML có CSS Selector như tham số 1 và thay nội dung text của nó bằng giá trị của tham số thứ 2. Chú ý, trong trường hợp không tìm thấy phần tử HTML nào phương thức DOM sẽ trả về null, null.textContent sẽ báo lỗi. Trong trường hợp này, hàm trả về return (không cần trả về giá trị) để ngăn code không thực thi tiếp.

Bài 2: Viết một hàm nhận 2 tham số, tham số 1 là giá trị chuỗi đại diện cho CSS Selector, tham số 2 là một object gồm các key sau: element (tên thẻ HTML), content (text hoặc nội dung HTML), className (tên CSS class). Hàm tìm ra phần tử HTML có CSS Selector như tham số 1 và thay đổi nội dung HTML của nó theo dữ liệu của tham số thứ 2. (Style cho tên class trong object)

Bài 3: Viết một hàm nhận 2 tham số là các giá trị chuỗi, tham số thứ nhất đại diện cho CSS Selector, tham số thứ 2 nhận một trong 2 giá trị là “add” hoặc “remove”. Hàm tìm ra phần tử HTML có CSS Selector như tham số 1 và thêm hoặc xóa tên class “active” cho phần tử đó dựa theo giá trị của tham số thứ 2. (Style cho phần tử đó trong trường hợp có class “active”).

Thay đổi thuộc tính style

Để lấy ra giá trị hoặc thay đổi nội dung của một phần tử HTML có thể sử dụng thuộc tính **style** và truy cập vào các thuộc tính CSS qua thuộc tính này. Chú ý, các thuộc tính CSS lúc này sẽ được khai báo bằng cú pháp Camel Case. Ví dụ: font-size => fontSize.

```
<body>
  <span class="text" style="color: blue">Content</span>

  <script src="./script.js"></script>
</body>
```


```
const paragraphEl = document.querySelector(".text");

paragraphEl.style.color; // "blue"
paragraphEl.style.color = "green";
```

Thay đổi thuộc tính style

Tuy nhiên, thuộc tính *style* của một phần tử HTML ở trong DOM chỉ cho phép truy cập và thay đổi các thuộc tính và giá trị CSS khai báo theo kiểu *Inline CSS*. Để làm việc với tất cả các thuộc tính của một phần tử HTML cần sử dụng đến hàm đặc biệt *getComputedStyle()*.

```html
<body>
  <span class="text" style="color: blue">Content</span>

  <script src="./script.js"></script>
</body>
```

```js
const paragraphEl = document.querySelector(".text");

paragraphEl.style.color; // "blue"
paragraphEl.style.fontSize; // ""
getComputedStyle(paragraphEl).fontSize; // "16px"

// ❌ Computed Style chỉ cho đọc giá trị, không cho thay đổi
getComputedStyle(paragraphEl).fontSize = "24px";

// ✅ Sử dụng thuộc tính "style" để thay đổi giá trị.
paragraphEl.style.fontSize = "24px";
// => Thuộc tính và giá trị CSS mới được áp dụng sẽ được thêm vào
// ở giá trị của thuộc tính "style" trong thẻ HTML.
```

Thay đổi thuộc tính khác

DOM cho phép truy cập và thay đổi giá trị của mọi thuộc tính HTML.

```html
<body>
  <img class="image" src="./images/dog-port.jpg" alt="A good boy" />

  <script src="./script.js"></script>
</body>
```

```js
const imageEl = document.querySelector(".image");

imageEl.className; // "image"
imageEl.alt; // "A good boy"

// Absolute Link
imageEl.src; // "http://1xx.0.0.1:5500/images/dog-port.jpg"

// Relative Link
imageEl.getAttribute("src"); // "./images/dog-port.jpg"
```

Chú ý: *Thuộc tính "className" cũng có thể sử dụng để thay đổi tên class của phần tử HTML. Tuy nhiên, nó sẽ bỏ hết đi các tên class cũ và thêm tên class mới bằng giá trị mà nó được gán.*

Thay đổi thuộc tính khác

HTML cho phép khai báo các thuộc tính với tên gọi và giá trị bất kỳ. Kỹ thuật này thường được sử dụng để đính kèm thêm thông tin từ dữ liệu vào code HTML kết hợp với với JavaScript hoặc Template Engine.

Để truy cập và thay đổi giá trị của các thuộc tính này, cần sử dụng các phương thức **getAttribute** và **setAttribute**.

```html
<body>
  <div class="product" product-id="p-123"></div>

  <script src="./script.js"></script>
</body>
```

```js
const productEl = document.querySelector(".product");

productEl.productId; // undefined
productEl.getAttribute("product-id"); // "p-123"
productEl.setAttribute("product-id", "p-321");
```

Chú ý: *Chỉ được sử dụng giá trị chuỗi để gán giá trị cho thuộc tính HTML.*

Thay đổi thuộc tính khác

HTML cung cấp kiểu khai báo thuộc tính có tiền tố (prefix) đặc biệt là “data-”. Tất cả các thuộc tính khai báo bằng cú pháp này sẽ được tổng hợp thành một object có key là tên của thuộc tính (sau tiền tố và viết theo cú pháp Camel Case) và giá trị của thuộc tính.

```
<body>
  <div class="product" data-product-id="p-123" data-product-type="book"></div>

  <script src="./script.js"></script>
</body>
```

```
const productEl = document.querySelector(".product");

productEl.dataset.productId; // "p-123"
productEl.dataset.productType; // "book"
productEl.dataset;
// => { productId: "p-123", productType: "book" }

// ES6
const { productId, productType } = productEl.dataset;
```

Các sự kiện trong DOM

Các sự kiện trong DOM

DOM cung cấp cho JavaScript các thuộc tính, phương thước để “lắng nghe” khi nào người dùng thực hiện các sự kiện tác động đến thành phần của một trang web trên trình duyệt như click, hover, click đúp, gõ bàn phím... Khi các sự kiện này kích hoạt, các thuộc tính và phương thức trên sẽ nhận một hàm để thực thi (Event Handler).

Cách 1 →

```html
<body>
  <button class="btn-1">Click me</button>

  <button class="btn-2" onclick="showMessage()">Click me too</button>

  <script src="./script.js"></script>
</body>
```

```js
const btn1 = document.querySelector(".btn-1");

function showMessage() {
  console.log("Hello");
}

btn1.onclick = showMessage;
// Hoặc
btn1.addEventListener("click", showMessage);
```

Cách 2 →

Cách 3 →

Các sự kiện trong DOM

Các sự kiện phổ biến trong DOM:

- click
- mouseenter: Hover vào.
- mouseout: Hover ra.
- focus: Click chuột hoặc tab vào ô input.
- blur: Click chuột hoặc tab ra khỏi ô input.
- dbclick: Click đúp.
- keydown: Phím ấn xuống.
- keypress: Nhấn giữ phím.
- keyup: Thả phím ấn.
- submit: Gửi dữ liệu từ form.
- scroll: Cuộn trang.
- change: Giá trị của input thay đổi.

Tài liệu đầy đủ: *https://www.w3schools.com/jsref/dom_obj_event.asp*

Bài thực hành số 3

Bài thực hành số 3

Bài 1: Trong HTML, viết một thẻ có content là một số (mặc định là 0) và hai nút button có nội dung là “+” và “-”. Khi click vào nút cộng hoặc nút trừ sẽ cập nhật giá trị của số trên.

Bài 2: Giả sử cửa sổ trình duyệt của bạn là một căn phòng, nền trắng là đang bật đèn. Hãy tạo một nút button để tắt đèn của căn phòng này. Hay quy định content hoặc style cho nút này để người dùng biết ấn vào sẽ thế nào (Ví dụ đèn đang sáng thì content ghi là “On”, ấn tắt đi thì chuyển sang “Off”). Có thể thêm icon hình bóng đèn và đổi màu nền của button sang màu tối hơn khi đèn tắt.

Bài 3: Tạo cửa sổ modal. Tạo một button, khi click button đó thì mở ra một cửa sổ ở vị trí chính giữa cửa sổ tình duyệt và có một nút button để đóng. Click vào nút đó thì sẽ đóng cửa sổ vừa mở ra. (Gợi ý, sử dụng position fixed cho cửa sổ modal).

Bài 4 (Nâng cao): Cải tiến bài tập 3. Khi cửa sổ modal mở ra, nền của cả cửa sổ trình duyệt phía sau modal đó có một màu tối có độ trong suốt. Khi click vào nút đóng hay bên ngoài của cửa sổ modal đều đóng được cửa sổ này.

Đối tượng Event

Đối tượng Event

- Khi sự kiện trong DOM xảy ra, hàm Event Handler sẽ được gọi với một tham số truyền vào là một đối tượng Event.
- Đối tượng Event là một object chứa các thuộc tính và phương thức cung cấp thêm dữ liệu, cách thức để làm việc với DOM Event. Ví dụ như xem phần tử HTML cụ thể nào vừa được click, ngăn chặn hành vi reload trang khi submit form, ví trí của con trỏ chuột khi sự kiện xảy ra...

```
btn1.addEventListener("click", (event) => {
  console.log(event);
});
```

Đối tượng Event

Thuộc tính **target** trả về phần tử HTML cụ thể tạo ra sự kiện.

```html
<div class="buttons-group">
  <button class="btn">Button 1</button>
  <button class="btn">Button 2</button>
  <button class="btn">Button 3</button>
</div>
```


```js
const btnGroupEl = document.querySelector(".buttons-group");

btnGroupEl.addEventListener("click", (event) => {
  const clickTarget = event.target;

  // Nếu không click vào button thì hàm dừng lại.
  if (clickTarget.classList.contains("buttons-group")) {
    return;
  }

  console.log(clickTarget.textContent);
});
```

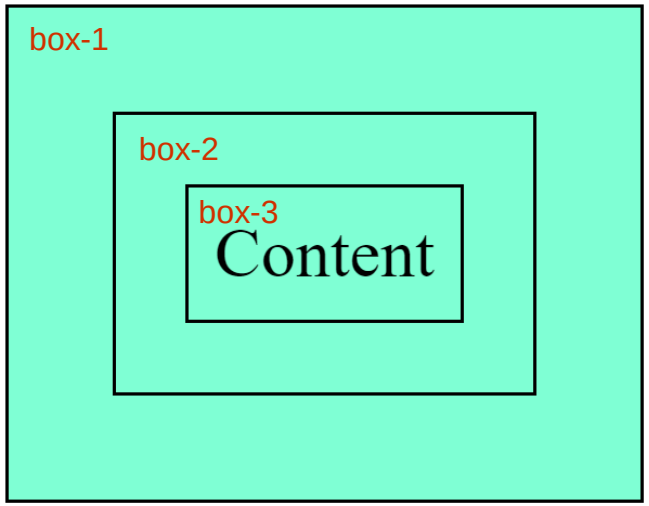
Khi người dùng click vào nút nào thì cửa sổ console sẽ in ra nội dung của nút button đó. Ví dụ nếu click vào nút 1, cửa sổ console in giá trị “Button 1”, tương tự với 2 nút còn lại.

Event Propagation - Lan truyền sự kiện

Lan truyền sự kiện xảy ra khi phần tử HTML cha và phần tử con của nó cùng lắng nghe chung 1 kiểu sự kiện.

Lúc này khi phần tử con kích hoạt sự kiện thì Event Handler của phần tử cha cũng được gọi.

```html
<div class="box-1">
  <div class="box-2">
    <div class="box-3">Content</div>
  </div>
</div>
```




```js
const box1El = document.querySelector(".box-1");
const box2El = document.querySelector(".box-2");
const box3El = document.querySelector(".box-3");

box1El.addEventListener("click", () => console.log("Box 1 click"));
box2El.addEventListener("click", () => console.log("Box 2 click"));
box3El.addEventListener("click", () => console.log("Box 3 click"));
```

Click “box-3”, cửa sổ console in:

“Box 3 click” “Box 2 click” “Box 1 click”

Click “box-2”, cửa sổ console in:

“Box 2 click” “Box 1 click”

Hành vi mong muốn là click box nào thì chỉ in ra nội dung liên quan đến box đó.

Event Propagation - Lan truyền sự kiện

Giải pháp: Sử dụng phương thức **stopPropagation** của đối tượng Event để ngăn không cho sự kiện ở một phần tử con lan truyền lên các phần tử cha của nó.

```
box2El.addEventListener("click", (event) => {
  event.stopPropagation();
  console.log("Box 2 click");
});
box3El.addEventListener("click", (event) => {
  event.stopPropagation();
  console.log("Box 3 click");
});
```

Chú ý:

- Khi khai báo đăng ký một sự kiện mới cần kiểm tra xem các phần tử HTML cha của phần tử đó có kiểu sự kiện trùng không mà gây ảnh hưởng không. Nếu có sử dụng **stopPropagation.**
- Với mỗi Event Handler cần kiểm soát các đối tượng của event qua thuộc tính **target** để đảm bảo đối tượng tạo ra sự kiện là đối tượng cần xử lý trong hàm.

Hoàn thành JavaScript – Buổi 6

Good job!