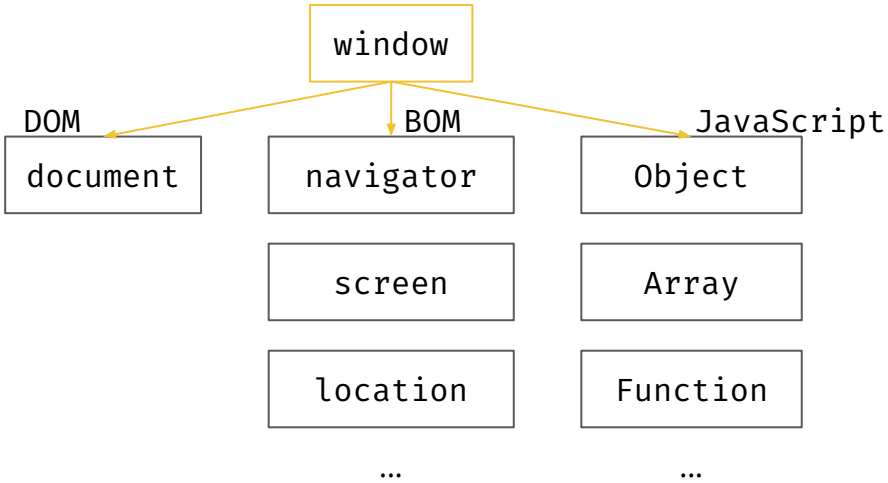
DOM

Ba Nguyễn

Introduction

JavaScript có thể chạy trên nhiều nền tảng / môi trường khác nhau (trình duyệt, máy chủ, ...). Mỗi môi trường cung cấp các chức năng riêng cho nó và ngôn ngữ JavaScript.

Trong môi trường trình duyệt bao gồm:



Concepts

window là đối tượng toàn cục - ***globalThis*** hay đối tượng *gốc*, đại diện cho cửa sổ trình duyệt

DOM (Document Object Model) là đối tượng đại diện cho toàn bộ nội dung trên trang, các thao tác thay đổi, thêm xóa các nội dung trên trang với JavaScript được thực hiện thông qua **DOM**

```
document.body.style.backgroundColor = "red";
```

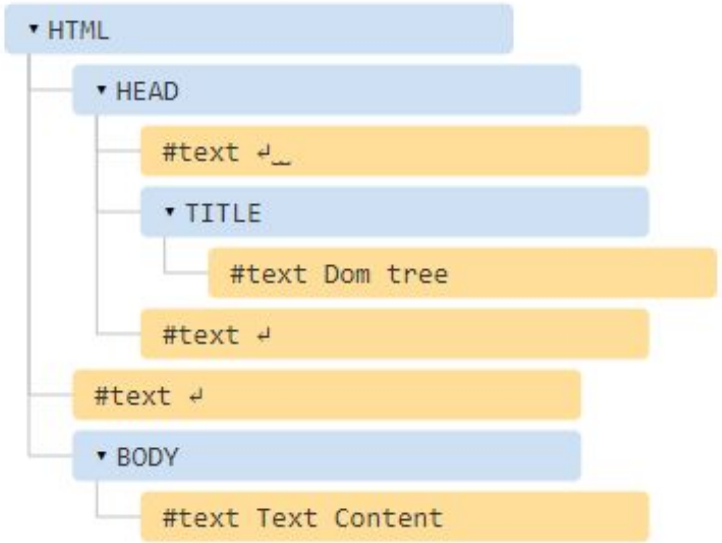
BOM (Browser Object Model) đại diện cho các đối tượng khác được cung cấp bởi trình duyệt, các đối tượng **BOM** cung cấp các phương thức quản lý/điều khiển trình duyệt

```
location.href = "https://google.com";
```

DOM

Trong **DOM**, *mọi thứ trong HTML đều là một object (hay một node)*, kể cả các thẻ rỗng, thẻ lồng nhau, comments, hay nội dung text bên trong các thẻ cũng là một object riêng biệt và đều có thể truy cập và chỉnh sửa thông qua ***DOM***

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dom tree</title>
  </head>
  <body>
    Text Content
  </body>
</html>
```

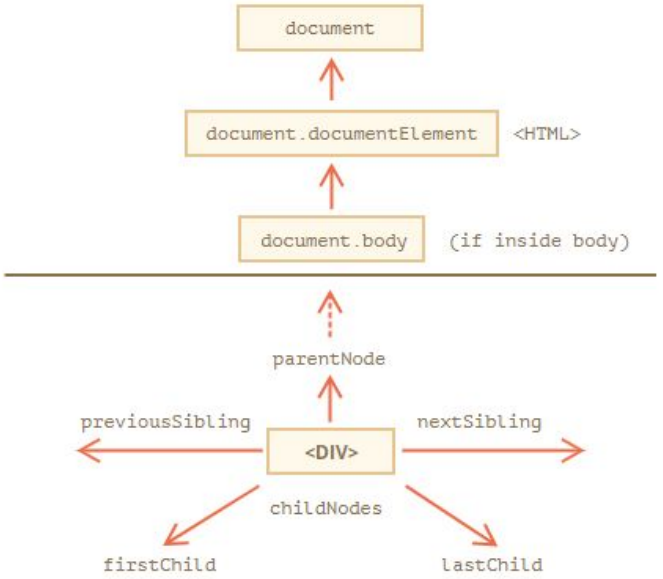


DOM

Tất cả thao tác với **DOM** được thực hiện thông qua object `document`

Một số node mặc định trong `document`:

```
// <!Doctype html>
document.doctype;
// <html>
document.documentElement;
document.head; // <head>
document.body; // <body>
```



💡 Lưu ý khi đặt thẻ `<script>` ở `<head>`

DOM Concepts

- ***Node***: Mọi thứ trong HTML đều trở thành một ***node*** trong ***DOM***, bất kể là phần tử HTML, hay text bên trong, dấu xuống dòng (dấu cách) giữa các phần tử, comment, ... anything
- ***Child nodes***: Chỉ bao gồm các ***node là con trực tiếp***
- ***Element*** chỉ bao gồm các ***phần tử HTML***
- ***DOM*** cung cấp các thuộc tính và phương thức để truy cập tới các node, ***nó tự động cập nhật khi một node được thêm, xóa khỏi trang***
- ***DOM Collections*** không phải là mảng, không có các phương thức mảng, để sử dụng được các phương thức của mảng, sử dụng:

```
Array.from(collection);
```

Nodes

```
// trỏ tới node cha
el.parrentNode;
// trỏ tới node phía trước
el.previousSibling;
// trỏ tới node phía sau
el.nextSibling;
// trỏ tới node con đầu tiên
el.firstChild;
// trỏ tới node con cuối cùng
el.lastChild;
// trả về Nodelist chứa toàn bộ node con
el.childNodes;
```

Elements

```
// trỏ tới phần tử HTML cha
el.parrentElement;
// trỏ tới phần tử HTML phía trước
el.previousElementSibling;
// trỏ tới phần tử HTML phía sau
el.nextElementSibling;
// trỏ tới phần tử HTML con đầu tiên
el.firstElementChild;
// trỏ tới phần tử HTML con cuối cùng
el.lastElementChild;
// trả về HTMLCollection chứa toàn bộ phần tử HTML con
el.children;
```

Loops

```
// DOM Collections không phải là mảng
// để sử dụng được các phương thức của mảng:
// Array.from(collection);

// Để duyệt qua một collection, sử dụng for of
for (node of document.body.children) {
  // code
}

// Hoặc .forEach() (với Nodelist)
document.body.childNodes.forEach((node) => {
  // code
});
```

Properties

Một số loại phần tử **DOM** cung cấp thêm một số thuộc tính bổ sung riêng của nó

```
// Ví dụ với <table>
table.caption; // <caption>
table.tHead; // <thead>
table.tFoot; // <tfoot>
table.tBodies; // collection of <tbody>
table.rows; // collection of all <tr> inside table
table.tBodies.rows; // collection of <tr> inside <tbody>
tr.rowIndex; // index of <tr> in <table>
tr.sectionRowIndex; // index of <tr> in <thead>,<tbody>,<tfoot>
tr.cells; // collection of <td>
td.cellIndex; // index of <td> or <th> in <tr>
```

Exercise

Tạo một bảng, sử dụng JS để thay đổi màu cho các ô theo đường chéo trong bảng (sử dụng `td.style.backgroundColor = “red”` để đổi màu nền)

1:1	1:2	1:3	1:4	1:5
2:1	2:2	2:3	2:4	2:5
3:1	3:2	3:3	3:4	3:5
4:1	4:2	4:3	4:4	4:5
5:1	6:2	5:3	5:4	5:5

Searching

Duyệt **DOM** bằng các thuộc tính chỉ nên dùng khi các phần tử nằm gần nhau, để tìm kiếm / lựa chọn một phần tử bất kỳ trên trang, **DOM** cung cấp các phương thức:

```
// search and return element by id
document.getElementById("id");
// search and return FIRST match element
el.querySelector("CSS Selector");
// search and return ALL match element
el.querySelectorAll("CSS Selector");
```

💡 `getElementById("id")` chỉ dùng được trên `document`, còn `querySelector*` thì có thể dùng trên bất kỳ phần tử nào (nhưng chỉ tìm kiếm các phần tử con)

Searching

Một số phương thức hữu ích khác

```
// check if element matches the CSS Selector
// return true or false
el.matches("CSS Selector");
// return collection by tag
document.getElementsByTagName("tag");
//return collection by class
document.getElementsByClassName("class");
```

💡 `getElementsBy*` tự động cập nhật trạng thái theo **DOM**, trong khi `querySelector*` trả về một collection tĩnh

Node Contents

Các thuộc tính để lấy/cập nhật nội dung các ***Node DOM***:

```
// get HTML inside element as "string"
el.innerHTML;
// replace HTML inside element
el.innerHTML = "<h2>LoL</h2>";
// 'append' HTML into element
el.innerHTML += "<p>Ooops</p>";
// get HTML include element
el.outerHTML;
// replace el by another HTML
el.outerHTML = "<div>Content</div>";
// get all text inside element, excludes tags
el.textContent;
el.textContent = "<h2>LoL</h2>";
```

Exercise

Sử dụng JS tạo đồng hồ đếm giờ hiển thị trên trang web

HTML Attributes vs DOM properties

Khi trang được tải, trình duyệt phân tích mã HTML và tạo ra cấu trúc **DOM** tương ứng. Hầu hết các thuộc tính HTML tiêu chuẩn sẽ được chuyển đổi thành thuộc tính **DOM** tương ứng.

```
<p id="p" title="LoL" type="text">Content</p>

<script>
    let p = document.getElementById("p");
    p.id; // p
    p.title; // LoL
    p.type; // undefined
</script>
```

HTML Attributes vs DOM properties

Các phương thức ***thao tác với thuộc tính*** của phần tử **DOM**:

```
// checks for existence
el.hasAttribute(attr);
// gets the value
el.getAttribute(attr);
//sets the value
el.setAttribute(attr, value);
// removes the attribute
el.removeAttribute(attr);
```

HTML Attributes vs DOM properties

Attributes và Propeties được *đồng bộ* với nhau, khi một attribute trong HTML thay đổi, property tương ứng trong **DOM** cũng thay đổi theo, và ngược lại (chỉ trừ một số ít trường hợp)

```
<input id="inp" type="text" value="LoL">

<script>
    let inp = document.getElementById("inp");
    inp.type = "checkbox"; // change type
    inp.value = "Ba Nguyễn"; // ???
</script>
```

💡 Giá trị của các thuộc tính cũng có thể có kiểu dữ liệu khác nhau, VD:

```
inp.checked; // false
```

Datasets

Để sử dụng các thuộc tính tùy chỉnh, HTML cung cấp attribute `data-*`

Tất cả thuộc tính bắt đầu với `data-` được chuyển đổi thành một thuộc tính tương ứng trong đối tượng `dataset` của phần tử ***DOM*** tương ứng.

```
<p id="p" data-content="LoL">Content</p>

<script>
    let p = document.getElementById("p");

    p.dataset.content; // LoL
</script>
```

Modifying the document

```
// Tạo mới một element:
// document.createElement("tagName");

let img = document.createElement("img");
img.src = "images/avatar.png";
img.alt = "Ba đẹp trai 🤣";

let p = document.createElement("p");
p.textContent = "Ba đẹp trai 🤣";
```

Modifying the document

```
<!-- Các phương thức thêm một Node vào DOM -->
<!-- ul.before(Node || String) -->
<ul>
    <!-- ul.prepend(Node || String) -->
    <li></li>
    <li></li>
    <!-- ul.append(Node || String) -->
</ul>
<!-- ul.after(Node || String) -->
```

Modifying the document

```
let div = document.createElement("div");
let p = document.createElement("p");

div.append(p, "<i>OMG</i>");
document.body.prepend(div);
```

💡 Chuỗi được thêm giống như `textContent`, chứ không tự động parse thành HTML như `innerHTML`

Modifying the document

```
// Các phương thức khác
el.insertAdjacentText(where, text);
el.insertAdjacentHTML(where, htmlString);
el.insertAdjacentElement(where, domNode);
```

```
<!— beforebegin —>
<ul>
    <!— afterbegin —>
    <li></li>
    <!— beforeend —>
</ul>
<!— afterend —>
```

Modifying the document

```
document.body.insertAdjacentText(
    "afterbegin",
    "<h1>Just text</h1>"
);

document.body.insertAdjacentHTML(
    "beforeend",
    "<h1>Now HTML</h1>"
);

let p = document.createElement("p");
document.body.insertAdjacentElement("beforeend", p);
```

Modifying the document

Nếu thêm cùng một ***node*** đã tồn tại, các phương thức chỉ đổi vị trí của ***node***, chứ không thêm nhiều ***node*** trùng nhau. Có thể sử dụng phương thức `el.cloneNode(true)` để sao chép một node và thêm vào ***DOM***

Xóa một ***node*** khỏi ***DOM***

```
node.remove();
el.removeChild(node);
```

DocumentFragment

`DocumentFragment` là một đối tượng đặc biệt, nó tạo ra một ***node ảo***, cho phép bọc nhiều ***node*** khác trong nó

```
let todos = ["ăn", "ngủ", "code"];
let ul = document.getElementById("ul");

function makeList(arr) {
    let fr = new DocumentFragment();
    arr.forEach(i => {
        let li = document.createElement("li");
        li.textContent = i;
        fr.append(li);
    });
    return fr;
}

ul.append(makeList(todos));
```

Styles & Classes

Có 2 cách tạo kiểu CSS cho phần tử bằng JS

- Tạo một class trong CSS và thêm class vào phần tử
- Thêm trực tiếp CSS vào thuộc tính style (inline CSS)

Thông thường, thêm hoặc xóa một class được ưu tiên hơn thay đổi trực tiếp thuộc tính style. Các thuộc tính/phương thức ***DOM*** thường dùng:

```
el.className; // class
el.classList; // collection
el.classList.add("class"); // add a class
el.classList.remove("class"); // remove a class
el.classList.toggle("class"); // "on - off" a class
el.classList.contains("class"); // boolean
```

Styles & Classes

Trong các trường hợp thay đổi trực tiếp giá trị thuộc tính CSS của phần tử, lưu ý:

- Đối với thuộc tính có dấu gạch nối, chuyển sang dạng *camelCase*
- Các thuộc tính có prefix chuyển sang dạng *capitalize*

```
el.style.width = "100px";
el.style.backgroundColor = "red";
el.style.WebkitBorderRadius = "10px";
```

- Để xóa (reset) một thuộc tính, gán cho nó một giá trị rỗng
- Phải đặt đúng đơn vị (với những giá trị như *pixel, %, rem*, ...)
- Đặt Full CSS: `style.cssText` hoặc `el.setAttribute("style", "...")`

Styles & Classes

Thuộc tính `style` chỉ hoạt động với *inline* CSS, để đọc các thuộc tính được áp dụng cho phần tử bởi các ***class***, sử dụng phương thức:

```
getComputedStyle(el , pseudo);
```

Phương thức trả về một object chứa đầy đủ các thuộc tính CSS được áp dụng bởi các class. Các giá trị được tính toán và trả về theo đơn vị cố định (thường là pixel). Lưu ý:

- Khi lấy giá trị từ computed style, nên sử dụng tên thuộc tính đầy đủ
- `getComputedStyle()` bị giới hạn với pseudo `:visited`

Forms

Form và các thành phần của form (như ***input***, ***button***, ...) có rất nhiều thuộc tính và sự kiện. Các phần tử `<form>` trên trang đều thuộc một đối tượng đặc biệt `document.forms`, các thành phần trong form như `<input>`, `<button>`, ... cũng thuộc một đối tượng đặc biệt `form.elements`

```
<form action="" name="login">
    <input type="text" name="username" />
    <button type="submit" name="btn">Login</button>
</form>

<script>
    let login = document.forms.login; // document.forms[0]
    let username = login.username; // login.elements.username
</script>
```

Form Elements

Đối với `<input>` và `<textarea>`, sử dụng thuộc tính `value`, `checked` để lấy/đặt giá trị

```
<form action="" name="demo">
    <input type="text" name="msg" value="I Hate You" />
    <input type="checkbox" name="remember" />
</form>
<script>
    let form = document.forms.demo;
    let msg = form.msg.value; // I Hate You
    form.msg.value = "LoL";
    form.remember.checked = "false";
</script>
```