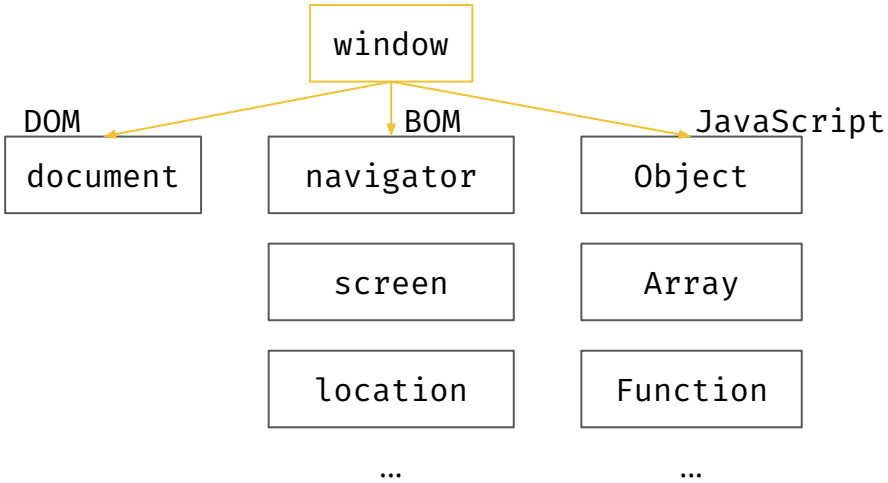
Introduction

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 được thực hiện thông qua **DOM**

Ví dụ:

```
setInterval(() => document.body.classList.toggle("bg"), 1000);
```

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

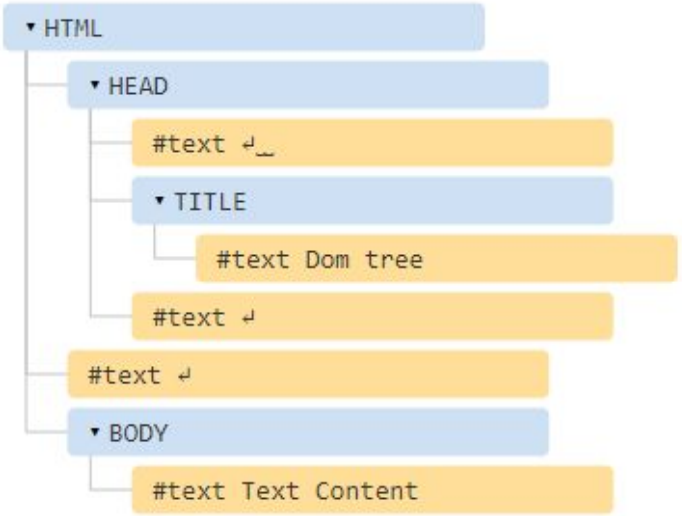
Ví dụ:

```
setTimeout(() => (location.href = "https://google.com"), 2000);
```

DOM

Trong **DOM**, mọi thứ trong HTML đều là một object, kể cả các thẻ lồng nhau, hay nội dung 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>
```

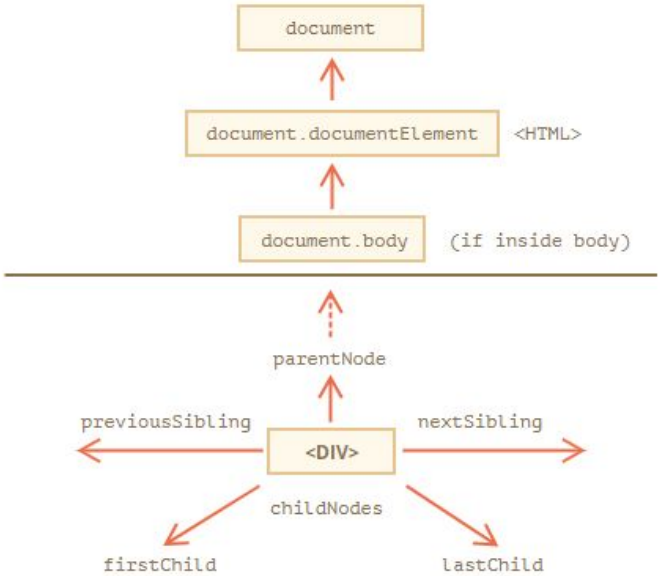


DOM

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

Một số node mặc định trong `document`:

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



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

DOM

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 node: Chỉ bao gồm các node là con trực tiếp

Element chỉ bao gồm các phần tử HTML

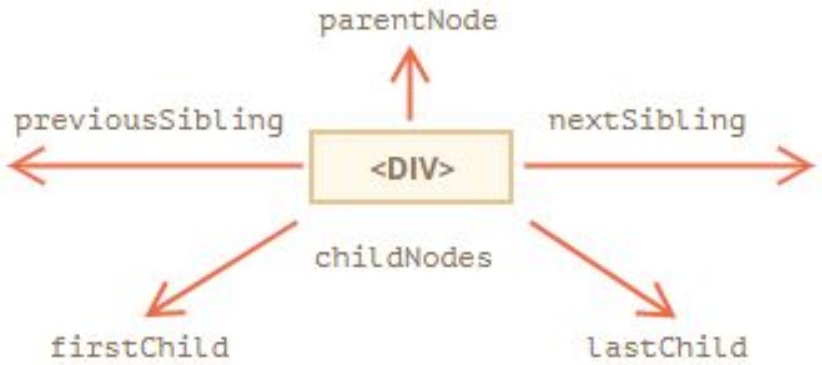
DOM cung cấp các thuộc tính và phương thức để truy cập tới các node, nó tự động cập nhật khi một node được thêm, xóa khỏi trang

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);
```

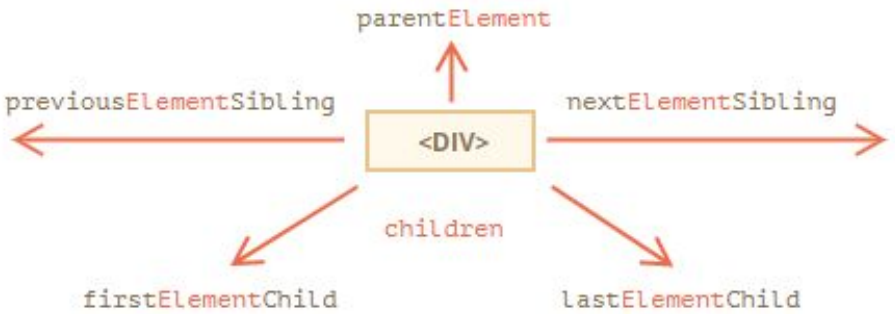
Nodes

```
// 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, sử dụng:

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

Để duyệt qua một collection, sử dụng cú pháp **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> or <tfoot>
tr.cells; // collection of <td>
td.cellIndex; // index of <td> or <th> in <tr>
```

Exercise

1. 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)

```
<!DOCTYPE html>
<html lang="en">
  <head> ···
  </head>
  <body>
    <table>
      <tbody>
        <tr>
          <td> ... </td>
        </tr>
      </tbody>
    </table>
    <script src="script.js"></script>
  </body>
</html>
```

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` mà không dùng được trên phần tử khác, còn `querySelector*` thì có thể dùng trên bất kỳ phần tử nào (chỉ tìm kiếm 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 = "😡😡😡";
```

Node Contents

```
// 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 = "  ";
// get all text inside element, excludes tags
el.textContent;
el.textContent = "<h2>LoL</h2>";
```

Exercise

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

```
<!DOCTYPE html>
<html lang="en">
  <head>…
  </head>
  <body>
    <p id="clock"></p>
    <script src="script.js"></script>
  </body>
</html>
```

11:17:38, 30/12/2020

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" type="text"></p>
<input id="input" type="text" />
<script>
  document.querySelector("p").type; // undefined
  document.querySelector("p").id; // p
  document.querySelector("input").type; // text
  document.querySelector("input").id; // input
</script>
```

Để lấy/thêm/sửa/xóa thuộc tính của phần tử, sử dụng các phương thức:

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

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="input" type="text" value="lol" />
<script>
  input.id = "lol"; // change id
  input.type = "email"; // change type
  input.value = "ba@techmaster.vn"; // ???
</script>
```

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

```
input.checked; // true
```

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** . VD:

```
<p
  id="p"
  data-name="Ba"
  data-age="28"
  data-multi-words="wtf">
</p>
<script>
  p.dataset.name; // Ba
  p.dataset.age; // 28
  p.dataset.multiWords; // wtf
</script>
```

Exercise

1. Sử dụng JS, điền giá trị từ object ra HTML thông qua **data-prop**

```
<div id="user">
  <p data-prop="name"></p>
  <p data-prop="age"></p>
  ...
</div>

<script>
  let user = {
    name: "Ba",
    age: 28,
    // ...
  };
```

Ba

28

...