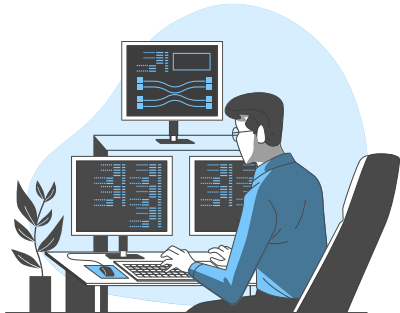
Advanced

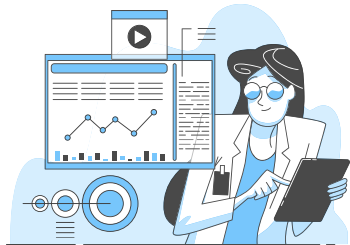
JS

Khóa học Backend

Bài 13:
Javascript nâng cao (Tiết 3)



Nội dung



01

Storage API

1.1. localStorage

- **localStorage** là **kho lưu trữ** ở **phía người dùng**.
- **Lưu trữ dữ liệu vô thời hạn**, dữ liệu sẽ được lưu trữ cho tới khi người dùng xóa lịch sử duyệt web.

- localStorage có **5 phương thức**:
  - **setItem()** : Thêm dữ liệu vào localStorage.
  - **getItem()** : Lấy dữ liệu từ localStorage.
  - **removeItem()** : Xóa dữ liệu ra khỏi localStorage.
  - **clear()** : Xóa toàn bộ dữ liệu ra khỏi localStorage.
  - **key()** : Lấy tên key của dữ liệu đang lưu trữ trong localStorage.

1.1. localStorage

- localStorage.**setItem()**
  - Dùng để **thêm dữ liệu** vào localStorage.
  - Dữ liệu sẽ được lưu trữ ở **dạng key và value**.
  - Cú pháp:

```
localStorage.setItem(key, value);
```

1.1. localStorage

- localStorage.**getItem()**
  - Dùng để **lấy dữ liệu**.
  - Truyền vào tên key muốn lấy.
  - Nếu không tồn tại thì trả về null.
  - Cú pháp:

```
localStorage.getItem(key);
```

1.1. localStorage

- localStorage.**removeItem()**
  - Dùng để **xóa dữ liệu**.
  - Truyền vào tên key muốn xóa.
  - Cú pháp:

```
localStorage.removeItem(key);
```

1.1. localStorage

- localStorage.**clear()**
  - **Xóa toàn bộ dữ liệu** của localStorage.
  - Truyền vào tên key muốn xóa.
  - Cú pháp:

```
localStorage.clear();
```

1.1. localStorage

- localStorage.**key()**
  - Lấy ra key của localStorage.
  - Nếu không có trả ra null.
  - Cú pháp:

```
localStorage.key(index);
```

- Trong đó: **index** là vị trí của key đó, có thể là 0, 1, 2, ...

1.2. sessionStorage

- **sessionStorage** là **kho lưu trữ theo phiên**.
- Lưu trữ dữ liệu cho một phiên làm việc, có nghĩa **dữ liệu** sẽ bị **mất khi** bạn **tắt browser**.

- sessionStorage có **5 phương thức**:
  - **setItem()** : Thêm dữ liệu vào sessionStorage.
  - **getItem()** : Lấy dữ liệu từ sessionStorage.
  - **removeItem()** : Xóa dữ liệu ra khỏi sessionStorage.
  - **clear()** : Xóa toàn bộ dữ liệu ra khỏi sessionStorage.
  - **key()** : Lấy tên key của dữ liệu đang lưu trữ trong sessionStorage.

1.2. sessionStorage

- sessionStorage.**setItem()**
  - Dùng để **thêm dữ liệu** vào sessionStorage.
  - Dữ liệu sẽ được lưu trữ ở **dạng key và value**.
  - Cú pháp:

```
sessionStorage.setItem(key, value);
```

1.2. sessionStorage

- sessionStorage.**getItem()**
  - Dùng để **lấy dữ liệu**.
  - Truyền vào tên key muốn lấy.
  - Nếu không tồn tại thì trả về null.
  - Cú pháp:

```
sessionStorage.getItem(key);
```

1.2. sessionStorage

- sessionStorage.**removeItem()**
  - Dùng để **xóa dữ liệu**.
  - Truyền vào tên key muốn xóa.
  - Cú pháp:

```
sessionStorage.removeItem(key);
```

1.2. sessionStorage

- sessionStorage.**clear()**
  - **Xóa toàn bộ dữ liệu** của sessionStorage.
  - Truyền vào tên key muốn xóa.
  - Cú pháp:

```
sessionStorage.clear();
```

1.2. sessionStorage

- sessionStorage.**key()**
  - Lấy ra key của sessionStorage.
  - Nếu không có trả ra null.
  - Cú pháp:

```
sessionStorage.key(index);
```

- Trong đó: **index** là vị trí của key đó, có thể là 0, 1, 2, ...

02

So sánh localStorage, sessionStorage, cookies

02. So sánh localStorage, sessionStorage, cookies

	cookies	**localStorage**	**sessionStorage**
Hết hạn (Expires)	Sau khi tắt trình duyệt hoặc đặt thủ công	Vô thời hạn	Sau khi tắt trình duyệt
Có thể truy cập/Lưu trữ	Server side hoặc Client side	Client side	Client side
Kích thước lưu trữ	4 KB	5 MB	10 MB

03

Closure (bao đóng)

03. Closure (bao đóng)

- **Closure** là **một hàm** ở **bên trong hàm khác**.
- Nó có thể sử dụng các biến toàn cục, biến cục bộ của hàm cha và biến cục bộ của chính nó.
- Nếu có các biến trùng tên thì closure sẽ ưu tiên theo thứ tự: biến cục bộ của chính nó, biến cục bộ của hàm cha, biến toàn cục.
- Nếu muốn return nhiều hàm closure thì bạn phải sử dụng một object, trong đó mỗi phần tử sẽ là một hàm closure.
- Closure thay đổi được giá trị biến toàn cục lẫn cục bộ.

04

Default parameters (Tham số mặc định)

04. Default parameters (Tham số mặc định)

- Tham số lấy giá trị mặc định nếu tham số đó không có giá trị truyền vào.
- Có 2 cách khai báo tham số mặc định:
  - Gán mặc định tại vị trí khai báo
  - Gán bên trong function.
- Ví dụ 1: Gán mặc định tại vị trí khai báo.

```
const sum = (a = 0, b = 0) => {
  return (a + b);
}

console.log(sum(10, 20)); // Trả về: 30
console.log(sum(10)); // Trả về: 10
```

04. Default parameters (Tham số mặc định)

- Tham số lấy giá trị mặc định nếu tham số đó không có giá trị truyền vào.
- Có 2 cách khai báo tham số mặc định:
  - Gán mặc định tại vị trí khai báo
  - Gán bên trong function.
- Ví dụ 2: Gán bên trong function.

```
const sum = (a, b) => {
  a = a || 0;
  b = b || 0;
  return (a + b);
}

console.log(sum(10, 20)); // Trả về: 30
console.log(sum(10)); // Trả về: 10
```

05

Spread syntax

05. Spread syntax

- Spread syntax (Cú pháp trải ra) để lặp lại các phần tử của mảng (array) hoặc đối tượng (object).
- Cú pháp: **...name**
- Ví dụ 1: Spread syntax với Array

```
const array1 = [1, 2, 3];
const array2 = [...array1, 4, 5];

console.log(array2);
// Trả về: [1, 2, 3, 4, 5]
```

05. Spread syntax

- Spread syntax (Cú pháp trải ra) để lặp lại các phần tử của mảng (array) hoặc đối tượng (object).
- Cú pháp: **...name**
- Ví dụ 2: Spread syntax với Object

```
const object1 = {
  fullName: "Le Van A",
  phone: "0123456789"
}

const object2 = {
  ...object1,
  email: "levana@gmail.com"
}

console.log(object2);
```

06

Rest parameters

06. Rest parameters

- Rest parameters (Tham số "còn lại") là tham số đại diện cho những tham số không được khai báo của một function.
- Cú pháp: **...name**
- Ví dụ 1:

```
const number = (num1, num2, ...numOther) => {
  console.log("num1:", num1);
  console.log("num2:", num2);
  console.log("Other:", numOther);
}

number(1, 2, 3, 4, 5, 6);
/*
  Trả về:
    num1: 1
    num2: 2
    Other: (4) [3, 4, 5, 6]
*/
```

06. Rest parameters

- Rest parameters (Tham số "còn lại") là tham số đại diện cho những tham số không được khai báo của một function.
- Cú pháp: **...name**
- Ví dụ 2:

```
const number = (...allNumber) => {
  console.log("All:", allNumber);
}

number(1, 2, 3, 4, 5, 6);
/*
  Trả về:
    All: (6) [1, 2, 3, 4, 5, 6]
*/
```

07

Destructuring

07. Destructuring

- Destructuring (phá vỡ cấu trúc) để dễ dàng lấy các phần tử của Array hoặc Object.
- Ví dụ 1: Destructuring với Array

```
const array = [1, 2, 3];

const [a, b] = array;

console.log(a); // Trả về: 1
console.log(b); // Trả về: 2

console.log(array[0]); // Trả về: 1
console.log(array[1]); // Trả về: 2
```

07. Destructuring

- Destructuring (phá vỡ cấu trúc) để dễ dàng lấy các phần tử của Array hoặc Object.
- Ví dụ 2: Destructuring với Object

```
const infoUser = {
  fullName: "Le Van A",
  phone: "0123456789"
}

const {fullName, phone} = infoUser;

console.log(fullName); // Trả về: "Le Van A"
console.log(phone); // Trả về: "0123456789"

console.log(infoUser.fullName); // Trả về: "Le Van A"
console.log(infoUser.phone); // Trả về: "0123456789"
```

Bài tập

- **Link bài tập**: https://dacavn.notion.site/B-i-t-p-b-i-13-Javascript-n-ng-cao-Ti-t-3-1da44b750a0e4d6b97e7d0584719ec38?pvs=4