Báo cáo môn CSDL Web Giữa kì

Nhóm số 4

Thành viên:

Phạm Ngọc Hải, Lương Đức Anh, Trần Minh Đức, Nguyễn Văn Thắng

Chủ đề:

JavaScript

Khoa: Khoa học dữ liệu

Môn: Cơ sở dữ liệu Web

Giảng viên: Vũ Tiến Dũng

Trường Đại học Khoa học Tự Nhiên

Đại học Quốc Gia Hà Nội

1. Giới thiệu về JavaScript

1.1. Định nghĩa

1.2. Nguồn gốc

1.3. Ưu - nhược của JavaScript

1.4. Tại sao nên học JavaScript

2. JavaScript: Kiến thức cơ bản

2.1. Cấu trúc code: Các khối lệnh

2.1.1 Câu lệnh

2.1.2 Comment

2.1.3. Biến

2.2. Các kiểu dữ liệu:

2.2.1. Number, BigInt

2.2.2. String

2.2.3. Boolean (kiểu dữ liệu logic)

2.2.4. Object (đối tượng)

2.3. Hàm

2.4. Các lệnh phổ biến

2.4.1. Lệnh if/else

2.4.2. Vòng lặp for / while

2.4.3. Vòng lặp với label

2.5. Tương tác của JavaScript với HTML

2.5.1. Truy cập JavaScript file

2.5.2. Tương tác với website

3. Các cơ chế đặc biệt của JavaScript

3.1. Cơ chế DOM (Document Object Model)

3.2. Cơ chế bất đồng bộ

3.2.1. Cách cơ chế hoạt động

5.2.2. Ứng dụng thực tiễn

3.3. Cơ chế lưu trữ data - Local Storage

3.3.1. Cách cơ chế hoạt động

3.3.2. Ứng dụng thực tiễn

4. Tổng kết

1. Giới thiệu về JavaScript

1.1. Định nghĩa

JavaScript là ngôn ngữ lập trình bậc cao và hướng đối tượng, được phát triển bởi Netscape Communication, được dùng cho các ứng dụng client / server.
JavaScript được các nhà phát triển sử dụng để lập trình các website tương tác (Interactive website) và các trang web động (Dynamic website):

+ Interactive website là trang web mà thiết lập những tương tác giữa người dùng và những nội dung hiện có của trang web. Một số tương tác có thể kể đến như chia sẻ nội dung, bình luận, trò chuyện trực tiếp, …

+ Dynamic website là 1 tập hợp các trang, với các nội dung có thể thay đổi tuỳ theo những thuộc tính từ người dùng như vị trí, hành động thực hiện trong quá khứ, múi giờ, …, sử dụng các ngôn ngữ như PHP và Python để kết nối với database, cho phép sự thay đổi trong các nội dung và các tương tác.

1.2. Nguồn gốc

JavaScript được biết đến là LiveScript, nhưng vào khoảng thời gian phát triển thì Java đang được coi là 1 hiện tượng nên đã Netscape đã ký thoả thuận marketing với Sun và đổi tên từ LiveScript thành JavaScript.

1.3. Ưu - nhược của JavaScript

Ưu điểm	Nhược điểm
- Nhanh và đơn giản: JavaScript rất dễ để hiểu và học, với cấu trúc đơn giản. Vì là ngôn ngữ thông dịch nên JavaScript giảm thời gian cần để chạy so với các ngôn ngữ khác như Java,	- Dễ bị khai thác, chèn mã độc - Hạn chế trên một số trình duyệt - Thiếu tính năng đa kế thừa

- Khả năng phản hồi nhanh chóng	

1.4. Tại sao nên học JavaScript

JavaScript là ngôn ngữ lập trình phổ biến nhất trên thế giới, được sử dụng ở gần như mọi website phổ biến: Google, Facebook, X, Amazon, Netflix, …

Từ những ngày đầu hình thành dưới cái tên LiveScript cho đến hiện tại, đã có rất nhiều thư viện và framework được phát triển để hỗ trợ và tăng độ hiệu quả khi lập trình bằng JavaScript để tạo ra các website hoặc ứng dụng di động, có thể kể đến những framework phổ biến như React, Angular, Vue.js, …

JavaScript không chỉ giới hạn ở xây dựng trang web mà còn có thể xây dựng các ứng dụng từ client-side tới backend, các ứng dụng / game mobile, các ứng dụng trên PC, hay kể cả trên cloud, …

2. JavaScript: Kiến thức cơ bản

2.1. Cấu trúc code: Các khối lệnh

2.1.1 Câu lệnh

Câu lệnh là các cấu trúc cú pháp và lệnh thực hiện các hành động cụ thể.
Một trong các hành động quen thuộc khi học code là thực thi câu lệnh in ra “Hello World” thì sẽ được thực thi như sau:

```
1  alert('Hello');
2  alert('World');
```

Ở đây, cụm “Hello World” được tách ra thành 2 lần thực thi câu lệnh alert, và thường được phân cách bởi dấu “ ; “ để giúp code dễ đọc hơn. Tất nhiên ta có thể lược bỏ dấu “ ; “ vì

JavaScript sẽ mặc định coi là có dấu " ; " mỗi khi kết thúc dòng, tất nhiên có những trường hợp ngoại lệ như sau:

```
alert(3 +
1
+ 2);
```

2.1.2 Comment

Người dùng có thể để lại comment ở code để chú thích, 1 dòng chú thích có thể bắt đầu bằng dấu " // "

Chú thích nhiều dòng có thể gói gọn trong 1 khối bắt đầu bằng " /* " và kết thúc bằng " */ " như sau:

```
/* An example with two messages.
This is a multiline comment.
*/
```

Chú ý: Ta không thể lồng các comment lại với nhau như sau vì JavaScript mặc định coi dấu " */ " đầu tiên là hết comment.

```
/*
  /* nested comment ?!? */
*/
```

2.1.3. Biến

Biến là 1 dữ liệu lưu trữ được đặt tên, có thể dùng cho các dữ liệu khác nhau và cho nhiều kiểu dữ liệu khác nhau.

Để khai báo biến, ta sử dụng lệnh let như sau:

```
let message = 'Hello!'; // define the variable and assign the value

alert(message); // Hello!
```

Ở đây ta đã vừa khai báo và gán giá trị vào biến message cùng lúc.

Ta có thể đồng thời khai báo nhiều biến cùng lúc ở các dòng khác nhau:

```
let user = 'John',
  age = 25,
  message = 'Hello';
```

1 biến có thể thay đổi giá trị tuỳ ý không giới hạn như sau:

```
let message;

message = 'Hello!';

message = 'World!'; // value changed

alert(message);
```

Khi giá trị mới được thay vào, giá trị cũ bị thay thế và xoá khỏi biến.

Chú ý: Không được khai báo cùng 1 biến 2 lần trở lên vì sẽ xảy ra lỗi Syntax Error.

```
let message = "This";

// repeated 'let' leads to an error
let message = "That"; // SyntaxError:
```

2.2. Các kiểu dữ liệu:

1 giá trị trong JavaScript sẽ luôn thuộc 1 kiểu dữ liệu nào đó, và có 8 kiểu dữ liệu cơ bản. Một biến có thể được gán 1 giá trị thuộc kiểu dữ liệu bất kì, VD:

```
// no error
let message = "hello";
message = 123456;
```

Có nghĩa là 1 biến ở JavaScript không bị cố định 1 kiểu dữ liệu nhất định và có thể thay đổi tuỳ ý. Ở đây ta sẽ đi qua 1 số kiểu dữ liệu.

2.2.1. Number, BigInt

Number là kiểu dữ liệu của cả kiểu dữ liệu int và float thường thấy ở các ngôn ngữ lập trình khác. Các toán tử cơ bản như cộng " + ", trừ " - ", nhân " * ", chia " / " cũng được áp

dụng cho kiểu dữ liệu Number. Cũng tồn tại những giá trị đặc biệt và thuộc kiểu dữ liệu Number: Infinity, - Infinity, NaN. Kiểu dữ liệu Number bị giới hạn trong khoảng từ -(2^53-1) đến (2^53-1). Quá khoảng này sẽ là kiểu dữ liệu BigInt

BigInt là kiểu dữ liệu tồn tại cho những giá trị vượt quá khoảng của dữ liệu Number để có thể xác định 1 cách chính xác. 1 giá trị BigInt có thể được tạo ra bằng cách thêm "n" vào cuối 1 số nguyên như sau:

```
// the "n" at the end means it's a BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

2.2.2. String

Kiểu dữ liệu String được dùng cho các giá trị xâu, và có 2 kiểu gói các giá trị xâu vào như sau:

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

1. Sử dụng " " và ' '. 2 kiểu này không có sự khác nhau đáng kể trong JavaScript.
2. Sử dụng ` `. Thường được sử dụng gán các biến và phép toán vào giá trị xâu bằng cách gán vào trong cụm "${...}":

```
let name = "John";

// embed a variable
alert( `Hello, ${name}!` ); // Hello, John!

// embed an expression
alert( `the result is ${1 + 2}` ); // the result is
```

Không tồn tại kiểu dữ liệu Char cho các giá trị là chữ cái, và chỉ có 1 kiểu dữ liệu là String.

2.2.3. Boolean (kiểu dữ liệu logic)

Kiểu dữ liệu Boolean chỉ có 2 giá trị: True và False. Như các ngôn ngữ lập trình khác, kiểu dữ liệu Boolean cũng có các toán tử logic như OR " || ", AND " && ", và NOT " ! ". Nhưng đồng thời cũng có 1 toán tử logic khác là " ?? ".

Toán tử OR thường được dùng trong các câu lệnh If nếu chỉ cần 1 trong các mệnh đề điều kiện đưa ra được thoả mãn. Điểm đặc biệt là với toán tử OR, mỗi khi đi qua các mệnh đề sẽ chuyển về kiểu dữ liệu Boolean, nếu 1 mệnh đề đúng, thì sẽ tự khắc dừng lại và trả lại giá trị Boolean đó (là true).

Toán tử AND được sử dụng khi cần các điều kiện phải được thỏa mãn cùng lúc. Tương tự như toán tử OR, tuy nhiên toán tử AND sẽ dừng lại khi 1 mệnh đề điều kiện chuyển thành kiểu dữ liệu Boolean trả về false. Bậc ưu tiên của toán tử AND cao hơn toán tử OR, có nghĩa là trong 1 loạt mệnh đề được đưa ra, toán tử AND được ưu tiên thực hiện trước.

Toán tử NOT được dùng để chuyển 1 mệnh đề có giá trị boolean từ True sang False và ngược lại.

Toán tử ?? có phần khác biệt, là sẽ đánh giá dựa trên 1 giá trị nào đó không phải null hay undefined, nếu thoả mãn thì trả về giá trị đầu tiên, ngược lại trả về giá trị thứ 2.

Ta có 2 mệnh đề A và B được đưa ra, với mệnh đề A là biến user, mệnh đề B là 1 giá trị String “Anonymous”

```
let user;

alert(user ?? "Anonymous"); // Anonymous (user is undefined)
```

Do user là 1 biến chưa được định nghĩa, không có giá trị, kiểu dữ liệu (undefined) nên sẽ trả lại giá trị String.

Toán tử ?? là 1 phiên bản khác của việc thực thi toán tử OR với 1 mệnh đề yêu cầu xác định 1 biến hay 1 giá trị không phải null hay undefined; nhưng điểm khác biệt là tại đó: toán tử OR không phân biệt được giữa false, 0, 1, giá trị String rỗng, và null/ undefined, mà coi là như 1 là true, 0 thì là false. VD:

```
let firstName = null;
let lastName = null;
let nickName = "Supercoder";

// shows the first truthy value:
alert(firstName || lastName || nickName || "Anonymous"); // Supercoder
```

Với cùng 1 tập các điều kiện ở toán tử OR, lẽ ra sẽ phải trả lại null nếu dùng toán tử ??, thì ở đây sẽ trả lại giá trị thoả mãn toán tử OR và dừng lại lập tức.

Toán tử === như toán tử == nhưng kiểm tra chặt chẽ hơn, cả kiểu dữ liệu lẫn giá trị.

2.2.4. Object (đối tượng)

Đây là kiểu dữ liệu được sử dụng để lưu trữ các dữ liệu phức tạp hơn các kiểu dữ liệu nguyên thuỷ như Number, BigInt, String, Boolean chỉ chứa 1 giá trị duy nhất.

1 đối tượng có thể chứa 1 bộ gồm rất nhiều dữ liệu đa dạng và phức tạp, và có thể khai báo như sau:

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

Ta có thể gán các thuộc tính vào 1 đối tượng theo cặp key : value trong cụm {...} như sau:

```
let user = {     // an object
  name: "John",  // by key "name" store value "John"
  age: 30        // by key "age" store value 30
};
```

Key cũng có thể có nhiều từ, nhưng phải được gói như 1 kiểu dữ liệu String:

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // multiword property name must be quoted
};
```

Giữa các cặp Key - Value được phân cách bởi dấu “ , “, và value ở đây có thể thuộc kiểu dữ liệu bất kì.

Để thêm bớt, truy cập value của 1 key, ta có thể thực hiện như sau:

```
let user = {};

// set
user["likes birds"] = true;

// get
alert(user["likes birds"]); // true

// delete
delete user["likes birds"];
```

2.3. Hàm

Hàm là các khối lệnh trong 1 chương trình, có thể được tái sử dụng nhiều lần, tránh lặp code. 1 hàm có thể được tạo như sau:

```
function name(parameter1, parameter2, ... parameterN) {
 // body
}
```

Keyword function ở đầu tiên và theo sau là tên hàm, cùng với danh sách các đối số được phân cách bởi dấu phẩy “ , “, và khối lệnh được thực thi của hàm ở trong ngoặc {...}. Những biến được tạo trong hàm chỉ tồn tại trong hàm đó. VD:

```
function showMessage() {
  let message = "Hello, I'm JavaScript!"; // local variable

  alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Error! The variable is local to the function
```

Như trên thì biến “message” được gọi trong hàm, nhưng gọi ở ngoài hàm thì lỗi vì không tồn tại biến.

Hàm cũng có thể truy cập các biến được khai báo ngoài hàm và toàn quyền thay đổi biến đó.

```
let userName = 'John';

function showMessage() {
  userName = "Bob"; // (1) changed the outer variable

  let message = 'Hello, ' + userName;
  alert(message);
}

alert( userName ); // John before the function call

showMessage();

alert( userName ); // Bob, the value was modified by the function
```

1 điểm đặc biệt là nếu trong hàm, ta khai báo biến ở ngoài 1 lần nữa, thì hàm sẽ lấy biến đó thay vì biến đã được khai báo ngoài hàm trước đó.

Hàm sẽ dùng đối số để truyền vào dữ liệu tương ứng với đối số được truyền dữ liệu vào.

2.4. Các lệnh phổ biến

2.4.1. Lệnh if/else

If else trong JavaScript có cấu trúc như trong Java, như sau

JS script.js

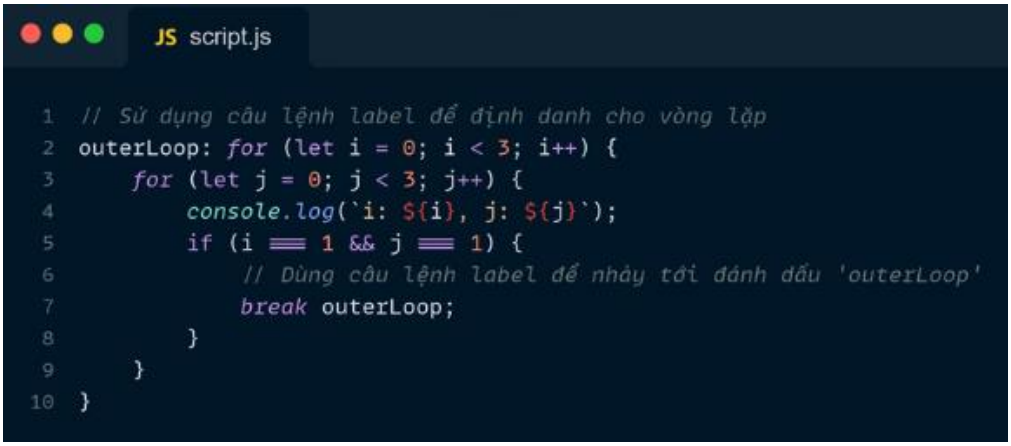
```
if (condition1) {
  /*  block of code to be executed
      if condition1 is true */
} else if (condition2) {
  /*  block of code to be executed
      if the condition1 is false
      and condition2 is true */
} else {
  /*  block of code to be executed
      if the condition1 is false
      and condition2 is false */
}
```

2.4.2. Vòng lặp for / while

Vòng lặp while cũng có cấu trúc như bên Java và hoạt động tương tự:

2.4.3. Vòng lặp với label

Trong ví dụ này, chúng ta sử dụng câu lệnh label để định danh cho vòng lặp outerLoop. Khi điều kiện i === 1 && j === 1 được đáp ứng, chúng ta sử dụng câu lệnh break kết hợp với câu lệnh label outerLoop để nhảy ra khỏi vòng lặp outerLoop. Điều này có nghĩa là vòng lặp for bên ngoài sẽ kết thúc ngay lập tức.



Lưu ý rằng khi sử dụng câu lệnh label trong JavaScript, nên cẩn thận và hạn chế việc sử dụng nó. Việc sử dụng câu lệnh label có thể làm cho mã của bạn trở nên khó đọc và khó hiểu, và nó thường không được khuyến nghị sử dụng trong các dự án lớn.

2.5. Tương tác của JavaScript với HTML

2.5.1. Truy cập JavaScript file

2.5.2. Tương tác với website

Khác với các ngôn ngữ như Java, PHP bắt đầu bằng hàm main, JavaScript chờ tương tác từ người dùng lên các hành động nhất định và phản hồi theo tương ứng. Các tương tác này được gọi là Event.

Một số event phổ biến từ HTML có thể kể đến như:

- Khi người dùng nhấp chuột
- Khi trang web đã tải
- Khi một hình ảnh đã tải
- Khi di chuyển chuột qua một phần tử
- Khi một trường nhập liệu được thay đổi
- Khi một biểu mẫu HTML được gửi
- Khi người dùng nhấn phím
- ...

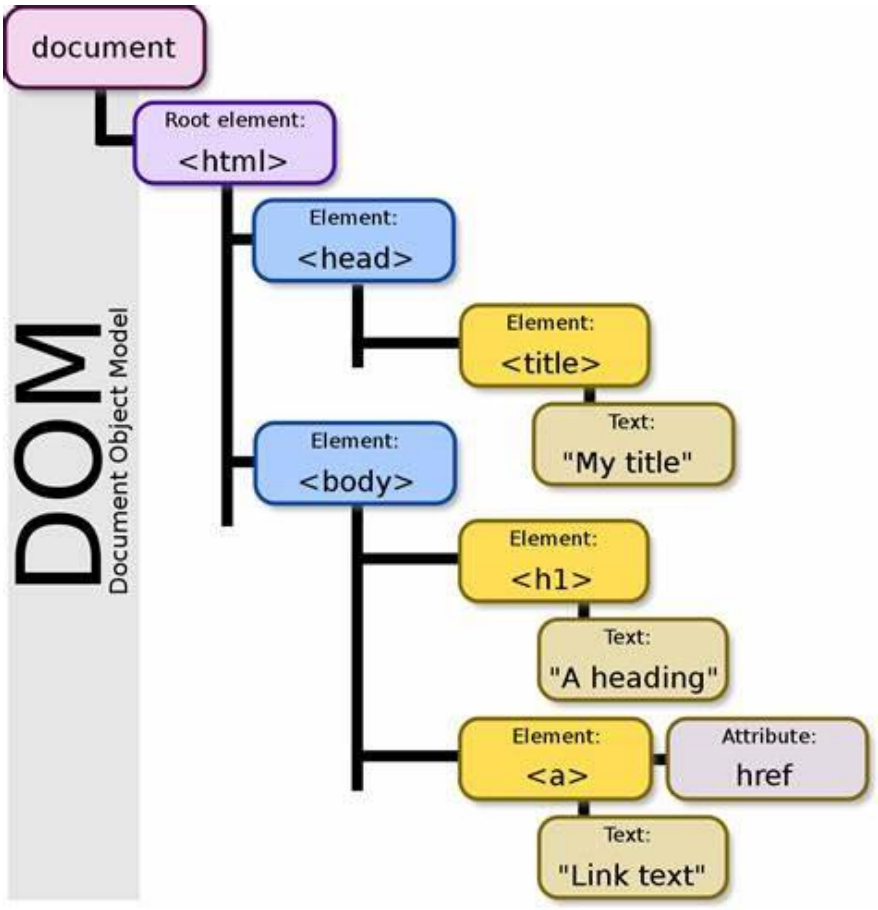
3. Các cơ chế đặc biệt của JavaScript

3.1. Cơ chế DOM (Document Object Model)

Document Object Model (DOM) là 1 chuẩn được định ra bởi W3C (World Wide Web Consortium), được dùng để truy xuất và thao tác các file dạng HTML, XML, JS, …

HTML DOM là 1 chuẩn mô hình đối tượng và interface cho HTML, định nghĩa các thuộc tính như đối tượng.

HTML DOM có cấu trúc như 1 cây, với các thành phần được biểu diễn dưới dạng node trên 1 cây được gọi là DOM tree.



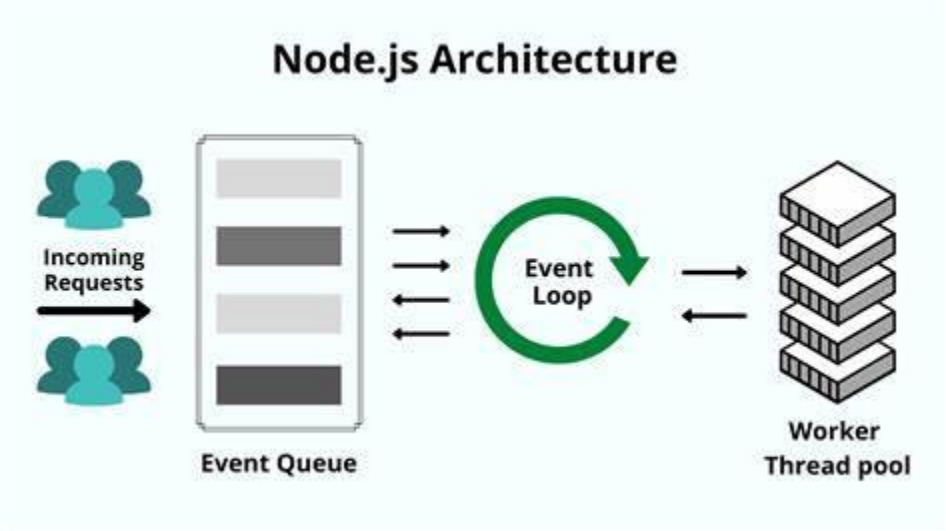
Hình ảnh 1 DOM tree

3.2. Cơ chế bất đồng bộ

3.2.1. Cách cơ chế hoạt động

JavaScript có khả năng chạy bất đồng bộ nhờ vào cơ chế "sự kiện lặp" (event loop) và "đồng bộ hóa không chặn" (non-blocking synchronization):

Sự kiện lặp (Event Loop): JavaScript sử dụng sự kiện lặp để quản lý việc lặp lại và xử lý các sự kiện trong một môi trường đơn luồng như trình duyệt hoặc Node.js. Event loop là một vòng lặp vô hạn kiểm tra và xử lý các sự kiện từ hàng đợi sự kiện. Khi một sự kiện xảy ra (ví dụ: người dùng nhấp chuột, kết thúc một tác vụ bất đồng bộ), sự kiện đó được đưa vào hàng đợi sự kiện và JavaScript sẽ xử lý các sự kiện này tuần tự. Điều này cho phép JavaScript tiếp tục thực thi các tác vụ khác trong khi đợi các sự kiện bất đồng bộ hoàn thành.



Quy trình của 1 event loop

Đồng bộ hóa không chăn (Non-blocking Synchronization): JavaScript sử dụng các cơ chế như Promise, async/await và callback để thực hiện đồng bộ hóa không chặn. Thay vì chờ đợi một tác vụ bất đồng bộ hoàn thành (như gọi API từ xa hoặc tải dữ liệu), JavaScript sẽ ghi nhớ tác vụ đó và tiếp tục thực thi các tác vụ khác. Khi tác vụ bất đồng

bộ hoàn thành, một sự kiện sẽ được tạo ra và sự kiện đó sẽ được đưa vào hàng đợi sự kiện để JavaScript xử lý. Điều này cho phép JavaScript không bị chặn và tiếp tục thực thi các tác vụ khác trong khi đợi các tác vụ bất đồng bộ hoàn thành.

Kết luận, sự kết hợp giữa sự kiện lặp và đồng bộ hóa không chặn cho phép JavaScript chạy bất đồng bộ một cách hiệu quả và không làm chặn luồng chính. Điều này làm cho JavaScript trở thành một ngôn ngữ lập trình hữu ích cho việc xử lý các tác vụ tốn thời gian như gọi API từ xa, tải dữ liệu và xử lý sự kiện người dùng mà không làm cho chương trình bị đơ hoặc không phản hồi.

5.2.2. Ứng dụng thực tiễn

Cơ chế bất đồng bộ (asynchronous) trong JavaScript cho phép thực hiện các tác vụ mà không làm chặn luồng chính của chương trình. Thay vì chờ đợi một tác vụ hoàn thành trước khi tiếp tục thực hiện các tác vụ khác, JavaScript sử dụng cơ chế bất đồng bộ để thực hiện các tác vụ song song và không chặn luồng chính.
JavaScript sử dụng các cơ chế bất đồng bộ như Promise, async/await và callback để xử lý các tác vụ bất đồng bộ. Điều này cho phép thực hiện các tác vụ như gọi API từ xa, tải dữ liệu từ cơ sở dữ liệu, xử lý tệp tin, v.v. mà không làm cho trình duyệt hoặc ứng dụng JavaScript bị đơ.
Ứng dụng của cơ chế bất đồng bộ trong JavaScript bao gồm:

+ Gọi API từ xa: Khi gọi API từ xa (ví dụ: gửi yêu cầu HTTP), cơ chế bất đồng bộ cho phép chương trình tiếp tục thực thi các tác vụ khác trong khi đợi kết quả trả về từ API.

+Xử lý sự kiện người dùng: JavaScript sử dụng cơ chế bất đồng bộ để xử lý sự kiện người dùng như nhấp chuột, gõ phím, v.v. Một khi sự kiện xảy ra, JavaScript có thể thực thi các tác vụ liên quan mà không làm chặn luồng chính.

+Tải dữ liệu từ cơ sở dữ liệu: Khi tải dữ liệu từ cơ sở dữ liệu, cơ chế bất đồng bộ cho phép chương trình tiếp tục thực thi các tác vụ khác trong khi đợi dữ liệu được trả về từ cơ sở dữ liệu.

+Xử lý tệp tin: Khi đọc hoặc ghi dữ liệu từ tệp tin, cơ chế bất đồng bộ cho phép chương trình tiếp tục thực thi các tác vụ khác trong khi đợi hoạt động đọc/ghi tệp tin hoàn thành.

Cơ chế bất đồng bộ giúp tăng hiệu suất và đáp ứng của ứng dụng JavaScript bằng cách cho phép thực thi các tác vụ song song và không chặn luồng chính. Điều này giúp tránh tình trạng chương trình đơ hoặc không phản hồi khi thực hiện các tác vụ tốn thời gian.

3.3. Cơ chế lưu trữ data - Local Storage

3.3.1. Cách cơ chế hoạt động

Local Storage là một API JavaScript cho phép bạn lưu trữ dữ liệu cục bộ trên trình duyệt của người dùng. Dữ liệu được lưu trữ trong Local Storage được lưu trữ dưới dạng một danh sách các cặp key-value. Key là một chuỗi duy nhất để xác định dữ liệu, và value là dữ liệu thực tế được lưu trữ.

Cơ chế hoạt động của Local Storage như sau:

Khi bạn sử dụng phương thức localStorage.setItem() để lưu trữ dữ liệu, trình duyệt sẽ tạo một cặp key-value mới trong Local Storage. Key là chuỗi bạn đã cung cấp cho phương thức, và value là dữ liệu bạn muốn lưu trữ.

Khi bạn sử dụng phương thức localStorage.getItem() để truy cập dữ liệu, trình duyệt sẽ trả về value của cặp key bạn đã cung cấp.

Khi bạn sử dụng phương thức localStorage.removeItem() để xóa dữ liệu, trình duyệt sẽ xóa cặp key-value tương ứng khỏi Local Storage.

3.3.2. Ứng dụng thực tiễn

Local Storage là một công cụ lưu trữ dữ liệu cục bộ trên trình duyệt của người dùng. Nó có một số **lợi ích** so với các phương pháp lưu trữ dữ liệu khác, bao gồm:

Dữ liệu được lưu trữ trên máy tính của người dùng, vì vậy nó sẽ vẫn còn tồn tại ngay cả khi tải lại trang web hoặc đóng trình duyệt.

Dữ liệu có thể được truy cập từ bất kỳ trang web nào trên cùng một trình duyệt.

Dữ liệu có thể được sử dụng để lưu trữ các thông tin như trạng thái của trang web, cài đặt của người dùng, hoặc dữ liệu cá nhân.

Tuy nhiên, Local Storage cũng có một số **hạn chế**, bao gồm:

Dữ liệu có thể bị xóa nếu người dùng xóa dữ liệu duyệt web của họ.

Dữ liệu có thể bị truy cập bởi các trang web độc hại. Để giảm thiểu rủi ro bảo mật, bạn nên sử dụng mã hóa cho dữ liệu lưu trữ trong Local Storage.

Nhìn chung, Local Storage là một công cụ lưu trữ dữ liệu hữu ích cho các ứng dụng web. Nó cung cấp một cách để lưu trữ dữ liệu trên máy tính của người dùng mà không cần sử dụng cookie.

4. Tổng kết

Qua những gì ta biết về JavaScript, có thể thấy JavaScript là 1 ngôn ngữ lập trình rất hiệu quả, đa năng và có thể sử dụng cho nhiều mục đích khác nhau: xây dựng trang web, app, … JavaScript là ngôn ngữ trình duyệt phổ biến nhất thế giới, được tích hợp hoàn toàn với HTML và CSS.

JavaScript phát triển qua nhiều năm phát triển cũng có rất nhiều thư viện và các framework hỗ trợ, như React, Angular, Vue, … và vẫn là 1 ngôn ngữ không ngừng phát triển và cải tiến, và sẽ là công cụ thiết yếu cho những nhà phát triển web.

* Câu trắc nghiệm:

1. Câu hỏi: Đối với hàm trong JavaScript, thuộc tính "arguments" được sử dụng để làm gì?
a. Lấy danh sách các tham số được truyền vào hàm.
b. Xác định kiểu dữ liệu của tham số đầu vào.
c. Trả về giá trị của biến arguments trong hàm.
d. Định nghĩa các tham số cho hàm.

2. Câu hỏi: Trong JavaScript, sự khác biệt giữa "asynchronous" (bất đồng bộ) và "synchronous" (đồng bộ) là gì?

a. "Asynchronous" là khi một tác vụ có thể chạy đồng thời với các tác vụ khác, trong khi "synchronous" là khi tất cả các tác vụ chạy theo thứ tự.
b. "Asynchronous" chỉ áp dụng cho khối mã lệnh, trong khi "synchronous" áp dụng cho hàm.
c. "Asynchronous" chỉ được sử dụng trong ngữ cảnh của Promise, trong khi "synchronous" áp dụng cho Callbacks.
d. "Asynchronous" và "synchronous" đều ám chỉ cùng một khái niệm, không có sự khác biệt nào.

3. Cho code sau:

```
function outer() {
    let x = 10;
    function inner() {
        console.log(x);
    }
    return inner;
}
let closureFunction = outer();
closureFunction();
```

Code trên trả về kết quả gì?
a) 10
b) undefined
c) Error
d) Ưu tiên thực thi outer trước, sau đó là inner

4. Cho code sau:

```
const obj = {
    name: "John",
    sayHello: function() {
        console.log(`Hello, ${this.name}!`);
    }
};
const sayHelloFunc = obj.sayHello;
sayHelloFunc();
```

Code trên trả về kết quả gì?

a) Hello, John!

b) Hello, undefined!

c) Error

d) Không có output nào được hiển thị

5. Câu hỏi: Trong JavaScript, hàm "apply()" và "call()" được sử dụng để làm gì?

a. Gọi một hàm ngay lập tức khi nó được định nghĩa.

b. Thay đổi phạm vi của "this" trong hàm.

c. Tạo một bản sao của hàm.

d. Gán một giá trị mới cho một biến.