CÁC LOẠI HÀM TRONG JAVASCRIPT

Mục lục

Mở đầu

Khi bắt đầu học một ngôn ngữ lập trình nào đó, một kiến thức mà chúng ta luôn phải quan tâm và ưu tiên để học đó là Funcion (hàm). Ngoài các hàm có sẵn (được ngôn ngữ lập trình đó cung cấp), chúng ta cũng có thể tự định nghĩa cho mình một (hoặc nhiều) hàm để giải quyết các vấn đề của mình trong quá trình phát triển dự án.
Vậy bạn có biết trong JavaScript có những loại hàm nào? hãy xem hết bài viết này để xem mình đã từng định nghĩa và sử dụng những loại hàm nào nhé.

Nội dung

1. Function (hàm) là gì?

Function (Hàm) là một tập hợp những đoạn mã dùng để thực hiện một nhiệm vụ nào đó. Nó có thể được gọi và tái sử dụng nhiều lần. Bạn có thể chuyển thông tin vào một hàm. Đồng thời, nó cũng có thể gửi trả thông tin trở lại. Nhiều ngôn ngữ lập trình có các hàm dựng sẵn mà người dùng chỉ cần lấy từ thư viện hàm. Ngoài ra, bạn cũng có thể tự tạo ra các hàm theo mục đích của riêng mình.

Cơ chế hoạt động của hàm có thể được giải thích một cách đơn giản như sau:

- Các hàm sẽ lần lượt được đọc theo thứ tự từ trên xuống dưới.
- Khi bạn gọi một hàm, chương trình sẽ tạm dừng ở đoạn đang chạy và tiến hành thực thi hàm.
- Khi hàm thực hiện xong, chương trình tiếp tục chạy ở nơi nó đang tạm dừng.
- Nếu hàm trả về một giá trị, giá trị đó chỉ dùng được khi bạn gọi xong hàm.

Có nhiều cách khác nhau để viết hàm. Cú pháp chính xác sẽ phụ thuộc vào ngôn ngữ lập trình mà bạn đang sử dụng. Chẳng hạn, cú pháp của Python sẽ hoàn toàn khác với cú pháp của JavaScript. Do đó, bạn cần ghi nhớ cú pháp để viết hàm được chính xác.

2. Các loại hàm trong JavaScript

a. Function Declaration

Đây là loại khai báo function cơ bản trong javascript và với loại này sẽ có áp dụng Hoisting trong javascript. Có nghĩa là ta có thể thực hiện lời gọi hàm trước khối lệnh định nghĩa hàm.

```
//Có thể gọi hàm trước/sau vị trí hàm được định nghĩa
doSomething();

//Hàm được định nghĩa theo dạng này gọi là Function Declaration
function doSomething() {
    console.log('doSomething');
}
```

b. Function Expressions

Function kiểu này sẽ được gán vào một biến và nó chỉ sẽ chạy khi runtime nếu biến đó được gọi và không áp dụng Hoisting. Như ví dụ bên dưới, chúng ta không thể thực hiện lời gọi hàm `doSomething()` ở dòng lệnh trước khi định nghĩa hàm.

```
//Hàm được định nghĩa theo dạng này gọi là Function Expressions
const doSomething = function() {
    console.log('doSomething');
}

//Chỉ có thể gọi hàm sau vị trí hàm được định nghĩa
doSomething();
```

c. Anonymous Function

Anonymous function là tên gọi của hàm ẩn danh hay còn gọi là hàm không có tên. Trong quá trình thực hiện đôi khi bạn muốn định nghĩa một hàm và sau sau đó chỉ thực thi hàm này 1 lần duy nhất và bạn không có nhu cầu sử dụng hàm này 1 lần (không có như cầu sử dụng lại reuse). Nếu vậy thay vì định nghĩa như 2 cách trên ta có thể chọn giải pháp sử dụng hàm ẩn danh.

```
let numbers = [1, 5, 6, 3, 3, 4, 5, 6];
//Mỗi khi duyệt qua một phần tử của mảng numbers thì sẽ thực thi hàm ẩn danh thực
hiện nhiệm vụ lấy giá trị hiện tại nhân với 2 và trả về kết quả.
let result = numbers.map(function(number){
    return number * 2;
})

// Hàm được định nghĩa theo dạng này gọi là hàm ẩn danh
function(number){
    return number * 2;
}
```

d. Arrow Function

Arrow function - còn được gọi là "fat arrow - Mũi tên béo", là cú pháp được mượn từ CoffeeScript (một ngôn ngữ chuyển tiếp), cú pháp này là cách ngắn gọn hơn dùng để viết function. Ở đây sử dụng kí tự `=>`, trông giống như một mũi tên "béo". Arrow function là một hàm vô danh và nó thay đổi cách `this` bind đến function. Arrow function làm code của ta trông ngắn gọn hơn, giúp đơn giản hóa function scoping cũng như từ khóa `this`. Arrow function hoạt động tương tự như `Lambdas` trong các ngôn ngữ khác như C # hay Python. Bằng cách sử dụng arrow function, chúng ta tránh được việc phải gõ từ khoá `function`, `return` và dấu ngoặc nhọn.

```
//Trường hợp không có return
const doSomething = () => console.log('doSomething');

//Trường hợp có return
const multiply = (number_1, numbers_2) => number_1 * number_2;

//Trường hợp có một tham số
// Cách 1
const power = (number) => number * 2;
// Cách 2
const power = number => number * 2;

//Trường hợp thân hàm có hơn 1 dòng lệnh
const isGreaterThan5 = (value) => {
    if(value > 5)
        return true;
    return false;
}
```

e. Callback Function

Để định nghĩa một Callback Function thì phải hội đủ những tính chất sau:

- Phải là một hàm
- Phải được truyền vào 1 hàm khác thông qua tham số (parameter)
- Phải được thực hiện lời gọi hàm bên trong hàm mà hàm này được truyền vào

```
//Định nghĩa hàm và đặt tên là callback
function callback(number_1, number_2){
    return number_1 + number_2
}

//Định nghĩa hàm useCallback
// trong trường hợp này tham số params sẽ trỏ cùng địa chỉ tham chiếu với callback
function useCallback(params){
    // khi thực hiện lời gọi params(10, 20) thì chương trình sẽ theo địa chỉ tham
chiếu và tìm đến định nghĩa hàm callback. Sau đó truyền 10 vào cho number_1 và 20
cho number_2
    console.log(params(10, 20))
```

```
}
//Ta truyền tham chiếu của hàm callback vào tham số của hàm useCallback
//Kết quả thực thi hàm useCallback sẽ hiển thị ra màn hình console giá trị là 30
useCallback(callback);
```

f. IIFE (Immediately Invoked Function Expression)

Với những loại hàm trên thì chúng ta phải cần phải thực hiện 2 bước, bước 1 là định nghĩa hàm và bước 2 là thực hiện lời gọi hàm. Trong JavaScript hỗ trợ chúng ta một giải pháp đó là IIFE (Immediately Invoked Function Expression) có nghĩa là khởi tạo một hàm và thực thi nó ngay lập tức.

IIFE hữu ích khi chúng ta muốn đóng gói code để nó không ảnh hưởng đến các biến toàn cục. Nó hữu ích khi chúng ta viết những thư viện, các bạn có thể kiểm tra các thư viện như jQuery hay bootstrap.bundle,.. Sẽ thấy các trường hợp áp dụng IIFE.

Cú pháp của IIFE

```
(function([parameters]){
 //Thân hàm
})([agruments]);
```

ví dụ:

```
//Kết hợp 2 thao tác định nghĩa và thực hiện lời gọi hàm
//Kết quả sau khi thực hiện sẽ hiển thị số 30 ở console
(function(number_1, number_2) {
    console.log(number_1 + number_2)
})(10, 20)
```

g. Generate Function

Generator Function là một tính năng mới của ngôn ngữ JavaScript giới thiệu từ phiên bản ES6.

- Generator đóng vai trò cơ bản để xây dựng các phương thức xử lý bất đồng bộ khác (side effect), ví dụ: async/await, sagas.
- Generator function là một function, có khả năng tạm ngưng thực thi trước khi hàm kết thúc, và có thể tiếp tục chạy ở 1 thời điểm khác.
- Generator function có ký tự * được đặt sau từ khoá function để phân biệt giữa function thông thường và generator function.

Cú pháp:

```
function* functionName([paramenters]) {
     //Thân hàm
}
```

Khi chúng ta gọi Generator function : " functionName() " , nó không trả về các kiểu dữ liệu cơ bản mà đẩy về một iterator object . Hàm next() của iterator object được sử dụng để truy xuất đến các node dữ liệu, sau mỗi bước resume lại generator function. Khi đó generator function sẽ thực thi hàm cho đến khi gặp từ khóa yield , hoặc return kế tiếp chưa được duyệt ở bước trước.

Nói cách khác hàm sẽ không được thực thi ngay sau khi gọi, mà thay vào đó generator function trả về iterator, giống như con trỏ trong vòng lặp. Sau khi hàm next() của iterator được gọi, generator function sẽ thực thi hàm cho đến khi gặp từ khóa yield đầu tiên. yield sẽ trả về giá trị cho iterator, generator function kết thúc cho đến khi hết giá trị để yield.

Ví dụ:

```
function* generateId() {
  yield 1;
  console.log('Tiếp tục chạy ...');
  yield 2;
  console.log('Tiếp tục chạy ...');
  return 3;
}

const newId  = generateId();

console.log(newId.next()); // {value:1, done:false}
                           // Tiếp tục chạy ...
console.log(newId.next()); // {value:2, done:false}
                           // Tiếp tục chạy ...
console.log(newId.next()); // {value:3, done:true}
```

3. Kết luận

Như vậy chúng ta đã tìm hiểu về định dạng/cú pháp của một số hàm trong JavaScript. Có thể các bạn đã từng sử dụng/làm việc với một trong nhưng loại hàm này, nhưng chưa xác định được tên gọi của từng loại hàm. Hy vọng qua bài viết này các bạn sẽ biết thêm về các loại hàm có trong javaScript và qua đó giúp bạn có thêm nhiều sự lựa chọn khi thiết kế các hàm trong quá trình phát triển phần mềm.