JavaScript ES6

Ba Nguyễn

Hoisting

Với ***cú pháp khai báo hàm (function declaration)***, hàm có thể được gọi trước khi nó được khai báo, tuy nhiên, với ***biểu thức hàm (function expression)***, hàm sẽ chỉ gọi được sau khi nó được khai báo

```
hi(); // 👌 oke
function hi() {
    // code
}
hi(); // 👌 oke
```

```
bye(); // ❌ error
let bye = function() {
    // code
}
bye(); // 👌 oke
```

💡 Khi thực thi, JavaScript *nhấc* - ***hosting*** - tất cả khai báo hàm (và cả biến nếu khai báo với `var`) lên đầu tập lệnh hoặc *block code*

Hoisting

Với biến được khai báo bởi `var`, nó cũng được *nhấc* lên đầu giống như hàm, tuy nhiên, giá trị chỉ được gán cho tới đúng vị trí câu lệnh trong mã, ***hoisting*** không hoạt động với `let` và `const`

```
alert(value); // undefined
var value = "Ba";
alert(value); // "Ba"
```

```
// var value; // undefined
// alert(value);
// value = "Ba";
// alert(value); // "Ba"
```

```
// Với let
alert(value); // ❌ error
let value = "Ba";
alert(value); // "Ba"
```

IIFE

IIFE (*Immediately Invoked Function Expression*) là hàm được khai báo và thực thi ngay lập tức. Một hàm ẩn danh được khai báo bên trong toán tử nhóm `()` và gọi ngay sau đó

```
// Syntax
(function () {
    // code
})(); // Call
```

```
let x = (function () {
    return 10;
})();

console.log(x); // 10
```

Closure

Trong JavaScript, hàm có thể ghi nhớ và truy xuất tới giá trị của các biến được khai báo bên ngoài nó, kết hợp với ***scope***, chúng ta có thể ***ẩn*** các giá trị khỏi mã bên ngoài, nhưng cho phép truy cập thông qua một hàm, đó được gọi là **closure**

```
function f() {
    var x = 1;

    return function () {
        return x;
    };
}

console.log(x); // error
getX = f(); // function() { return x; }
getX(); // x
```

JavaScript ES6

ECMA Script (ES) là bản đặc tả dành cho các ngôn ngữ Scripting như JavaScript, nó xác định các tính năng được triển khai trong ngôn ngữ. Các phiên bản mới hơn bổ sung rất nhiều tính năng và cú pháp cho JavaScript.

Các framework JavaScript phổ biến hiện tại như NodeJS, Angular của Google, ReactJS của Facebook sử dụng các tính năng JavaScript ES6 (và mới hơn).

Tìm hiểu thêm về các tiêu chuẩn JavaScript tại đây [wiki/ECMAScript]

JavaScript ES6 Features

Một số tính năng mới trong JavaScript

- Variable: let, const
- Scope
- Arrow function
- Default parameter
- Rest parameter, spread operator
- Template literal
- Destructuring assignment
- Module
- Class
- Promise
- Async/Await

Variables

Từ khóa `let` khai báo một biến với block scope, đồng thời biến không thể được khai báo lại

```
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

Variables

Từ khóa `const` khai báo hằng số (còn được gọi là "biến bất biến - immutable"), là các biến không thể gán lại giá trị.

Lưu ý: điều này chỉ làm cho bản thân biến trở nên bất biến chứ không phải giá trị của nó (ví dụ: trong trường hợp giá trị là một đối tượng, điều này có nghĩa là bản thân đối tượng vẫn có thể được thay đổi)

```
const PI = 3.141593;

PI = 3.14; // error

const user = {
    name: "Ba"
};

user.name = "Béo Ú"; // okela
```

Scope

Scope *chỉ phạm vi tồn tại của một biến*. Một biến chỉ tồn tại bên trong khối ***block code*** mà nó được khai báo. Một ***block code*** là đoạn code được đặt trong cặp dấu `{ }`. ***Các cấu trúc điều khiển, hàm, class là một block code.***

💡 Riêng biến khai báo với `var` chỉ bị giới hạn trong ***block code của function***

```
{
    let x = 1;
    const y = 2;
    var z = 3;
}
alert(x); // ❌ error
alert(y); // ❌ error
alert(z); // 👌 3
```

```
if (true) {
    let x = 1;
    const y = 2;
    var z = 3;
}
alert(x); // ❌ error
alert(y); // ❌ error
alert(z); // 👌 3
```

```
function a() {
    let x = 1;
    const y = 2;
    var z = 3;
}
alert(x); // ❌ error
alert(y); // ❌ error
alert(z); // ❌ error
```

Scope

```
for (let i = 0; i < 10; i++) {
  // code
}
console.log(i); // ✗ error i is not defined

for (var i = 0; i < 10; i++) {
  // code
}
console.log(i); // i = 10
```

Arrow functions

Arrow function là cú pháp mới, cho phép viết biểu thức hàm ngắn gọn hơn và bao gồm một số tính năng khác biệt so với hàm thông thường

Arrow function không tồn tại từ khóa `this` (hay `arguments`)

```
let secretMessage = () ⇒
    console.log("Hey, I love you 😘😘😘😘");
```

```
let arr = [1, 2, 3];
arr.map((i) ⇒ i * i);
```

Default Parameters

Chỉ định giá trị mặc định cho tham số

```
function hi(name = "Ba") {
    console.log("I love " + name);
}

hi(); // "I love Ba"
hi("Someone 😙"); // "I love Someone 😙"
```

Rest Parameters

Trong JavaScript, một hàm có thể được gọi với nhiều/ít đối số hơn số lượng tham số được khai báo trong hàm. Cú pháp **rest parameters** cho phép gọi hàm với số lượng đối số bất kỳ, các đối số truyền vào hàm được tổng hợp trong một mảng.

```
let sum = (a, b, ...others) => {
    let total = a + b;

    for (let other of others) {
        total += other;
    }

    return total;
};
```

Spread operator

Phân tách các phần tử của một tập hợp có thể lặp lại - *iterable* (như một mảng hoặc thậm chí một chuỗi) thành cả phần tử/tham số hàm riêng lẻ

```
var params = ["hello", true, 7];
var other = [1, 2, ...params]; // [ 1, 2, "hello", true, 7 ]

function f(x, y, ...a) {
    return (x + y) * a.length;
}
f(1, 2, ...params) === 9;

var str = "foo";
var chars = [...str]; // [ "f", "o", "o" ]
```

Template literals

Template literal (`` ``) là cú pháp mới, cho phép "nhúng" (nội suy) giá trị của một biến, biểu thức, hoặc thậm chí một phương thức vào chuỗi sử dụng cú pháp `${ }`

```
var customer = { name: "Foo" };
var card = { amount: 7, product: "Bar", unitprice: 42 };

var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`;
```

Destructuring assignment

Destructuring assignment là một cú pháp đặc biệt cho phép “giải nén” các mảng hoặc đối tượng thành một loạt các biến

```
let arr = ["John", "Smith"];

let [firstName, surname] = arr;
```

Destructuring assignment

Destructuring assignment là một cú pháp đặc biệt cho phép “giải nén” các mảng hoặc đối tượng thành một loạt các biến

```
let options = {
    title: "Menu",
    width: 100,
    height: 200,
};

let { title, width, height } = options;

alert(title); // Menu
alert(width); // 100
alert(height); // 200
```

Modules

Khi ứng dụng phát triển lớn hơn, việc tổ chức tất cả mã JavaScript trong một file khiến mã trở nên khó bảo trì, thay vào đó một cách làm phổ biến là chia tách mã ra thành các file với chức năng riêng biệt, được gọi là các **module**

Module là một file JavaScript, mỗi file JavaScript là một module 🙂

Các module có thể được tải và sử dụng các chức năng của nó ở một module khác. Để cho phép một module khác sử dụng/hoặc sử dụng một chức năng từ một module khác, JavaScipt cung cấp 2 từ khóa đặc biệt `export` và `import`

- `export` cho phép module khác quyền truy cập tới chức năng/biến trong module hiện tại
- `import` nhập các chức năng/biến từ module khác để sử dụng

Lưu ý: `export` và `import` chỉ hoạt động với giao thức **HTTP** (mở trang với **liveserver**)

Modules

Modules vs Regular JavaScript Files

Sự khác biệt giữa các file JavaScript "*thông thường*" và module:

- Module luôn *bật* mode `"use strict"`
- Module có scope của riêng nó, các biến/hàm/class/... không được `export` sẽ chỉ có thể truy cập được trong module hiện tại
- Các module sẽ được thực thi lần đầu tiên khi nhập, các câu lệnh `import` cùng một module sẽ chia sẽ chung dữ liệu đó
- Đối tượng `import.meta` chứa thông tin (phụ thuộc vào environment) về module hiện tại
- Trong module, `this` là `undefined`

Export

```
export let days = ["Mon", "Tue", " ... "];
export const MODULE_NAME = "module.js";
export function func() { }
export class User { }

function sayLove() { }
let name = "Ba";
export { sayLove, name };

export { sayLove as love, name as n };
```

Export Default

Lợi ích của module là chia nhỏ các tệp tin, mỗi chức năng đều nằm trong một module riêng. Ngoài các cách **`export`** thông thường, JavaScript hỗ trợ cú pháp **`export default`**, với cú pháp *gọn gàng* hơn 😃

Mỗi module chỉ có 1 **`export default`**, ngoài ra các giá trị **`export default`** không cần đặt tên

```
// 📂 module.js
export default () => console.log("Love You 😊");

// 📂 otherModule.js
function sayLove() { console.log("Love You 😊"); }
export { sayLove as default };

// 📂 index.js
// Đặt tên tùy ý khi import
import sayLove from "./module.js"
import sayLoveAgain from "./otherModule.js"
```

Import

```
import { func } from "./module.js";
import { func as f } from "./module.js";
// import default
import func from "./module.js";
import { default as func } from "./module.js";
// import everything as object
import * as m from "./module.js";
m.func(); // call function
// import and run
import "./module.js";
```

Exercises

Xây dựng form đăng nhập, với các yêu cầu:

- Module validate.js chứa các chức năng liên quan đến xác thực dữ liệu
- Module index.js thực hiện các thao tác xử lý form, và hiển thị kết quả trên giao diện
- Giao diện HTML sử dụng bootstrap (Form Component)

Class Basic

Lập trình hướng đối tượng (*Object-oriented Programing* - OOP) là một kỹ thuật lập trình cho phép mô tả (trừu tượng hóa) các **đối tượng trong thực tế** bằng ngôn ngữ lập trình.

Một đối tượng bao gồm:

- Thuộc tính (attribute): Là các thông tin, đặc điểm (trạng thái) của đối tượng
- Phương thức (method): Là các chức năng, hành vi của đối tượng

Trong lập trình hướng đối tượng, một class là một mẫu (blueprint) mà chương trình có thể sử dụng để tạo các đối tượng, cung cấp các giá trị ban đầu (thuộc tính - trạng thái) và triển khai hành vi (phương thức).

JavaScript ES5 sử dụng `function` và `new` để tạo ra các đối tượng. JavaScript ES6 sử dụng cú pháp `class` mới cung cấp nhiều tính năng mạnh mẽ hơn cho lập trình OOP.

Class Basic

Class vs Function

```
class User {
    constructor(name) {
        this.name = name;
    }

    love() {
        alert(`I Love ${this.name}`);
    }
}

let user = new User("Ba");
user.love();
```

```
function User(name) {
    this.name = name;
}

User.prototype.love = function () {
    alert(`I Love ${this.name}`);
};

let user = new User("Ba");
user.love();
```

Class Getter/Setter

```
constructor(name) {
    this.name = name;
}

get name() {
    return this._name;
}

set name(val) {
    if (val.length < 2) {
        alert("Tên quá ngắn");
        return;
    }
    this._name = val;
}
```

Exercises

Tạo `class Clock` bao gồm các chức năng hiển thị đồng hồ bấm giờ trên trình duyệt (HTML) và các phương thức `start()`, `record()`, `stop()`, `reset()` cho phép chạy/ghi/dừng/reset đồng hồ bấm giờ.