
Javascript
ThS. Dương Hữu Thành,
Khoa CNTT, Đại học Mở TP.HCM,
thanh.dh@ou.edu.vn.
1

1


Nội dung chính
Giới thiệu JS, ES6 và lập trình JS cơ bản
JS Function
JS Class
Call, Apply và Bind
Module Pattern
Callback, Promise, Async & Await
Axios
2

2

Giới thiệu Javascript

- Javascript (JS) truyền thống là ngôn ngữ thông dịch và phía client (client-side) hỗ trợ lập trình hướng đối tượng.
- JS cho phép
  - Thay đổi nội dung thành phần HTML.
  - Thay đổi thuộc tính các thành phần HTML.
  - Thay đổi CSS các thành phần HTML.
  - Kiểm tra dữ liệu vào người dùng phía client.

Giới thiệu Javascript

- Với sự ra đời của NodeJS, JS có thể trở thành ngôn ngữ phía server (server-side).
- Trong phát triển ứng dụng Web, những ứng dụng có thể vừa làm việc trên server và client gọi là các ứng dụng isomorphic.

Giới thiệu ES6

- ECMAScript6 (ES6) là một kỹ thuật lập trình nâng cao trong JS, mã nguồn viết bằng ES6 sẽ độc lập nền tảng (cross-browser).
- Chuẩn RS6 giúp cho chương trình rõ ràng, dễ bảo trì hơn.

Biến

- Biến trong JS có thể giữ giá trị bất kỳ kiểu dữ liệu nào (untyped language).
- JS sử dụng từ khoá var để khai báo biến và không cần chỉ định kiểu dữ liệu.
- Ví dụ:

```
var x = 5;
x = true;
x = "good";
```

Biến

- Nếu sử dụng var ở
  - ngoài hàm thì biến có phạm vi toàn cục.
  - trong hàm thì biến có phạm vi thuộc hàm đó.
  - trong block thì biến vẫn có hiệu lực ngoài phạm vi block đó.

```
function test (bool) {
  var amount = 0
  if(bool){
     var amount = 1
  }
  return amount
}
console.log(test(true));
```

Mong muốn là 0

Thực tế là 1

Biến let

- Từ ES6, thông thường khi cần khai báo biến phạm vi block thì ta thường dùng let thay cho var vì có nhập nhằng về phạm vi biến.
- Biến let chỉ có hiệu lực trong block {}.

```
function test (bool) {
    let amount = 0
    if(bool) {
        let amount = 1;
    }
    return amount
}
```

Biến const

- Biến const dùng khai biến có giá trị không thể thay đổi.

```
const a = 10;
```

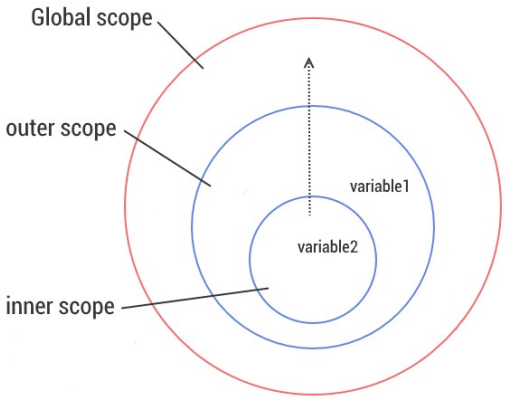
- Biến const có hiệu lực trong phạm vi block.

Phạm vi trong JS

- Phạm vi biến (scope) chỉ định nơi có thể truy cập vào các biến hoặc hàm.
  - Global scope: biến và hàm được khai báo toàn cục và có thể truy cập bất cứ nơi đâu.
  - Function scope (local scope): biến và hàm được khai báo trong phạm vi hàm và chỉ được truy cập trong hàm.
  - Block scope (let, const): biến được khai báo trong block bọc bởi {} và chỉ được truy cập trong block.

Phạm vi trong JS

- Quy tắc tìm biến dựa trên scope: JS tìm trong phạm vi hiện tại, đến các phạm vi ngoài nó, rồi đến phạm vi toàn cục.



Kiểu dữ liệu

- Kiểu số (Number) trong JS đều được biểu diễn theo chuẩn IEEE 754. Trong xử lý số có giá trị đặc biệt gọi là NaN (Not a Number) khi một phép tính nào đó trả về số không thành công.
- Kiểu chuỗi (String) là một dãy các ký tự 16-bit Unicode.
- Kiểu luận lý (Boolean) có 2 giá trị true và false.

Kiểu dữ liệu

- Kiểu Undefined chỉ có một giá trị duy nhất là **`undefined`**. Biến khai báo bằng let hoặc var và không khởi động giá trị sẽ có giá trị **`undefined`**.
- Kiểu Null chỉ có một giá trị duy nhất là **`null`** là một con trỏ trỏ tới đối tượng rỗng.
- Kiểu Object chứa tập các dữ liệu là các cặp key/value, sử dụng nhóm các dữ liệu và chức năng.

Chuỗi

- Trong JS, chuỗi đặt giữa cặp dấu ‘’ hoặc “”.
- Lấy chiều dài chuỗi s: s.legnth
- Các phương thức thông dụng đối tượng chuỗi
  - charAt(idx)
  - toLowerCase()/toUpperCase()
  - concat()
  - indexOf(s1 [, fromIndex])
  - lastIndexOf(s1 [, endIndex])
  - includes()

Chuỗi

- Các phương thức thông dụng đối tượng chuỗi
  - substring(fromIndex, endIndex)
  - substr(fromIndex, len)
  - slice(fromIndex, endIndex)
  - split(seperator)
  - replace(oldStr, newStr)
  - search(re)
  - match()
  - trim()

Chuỗi

- Một số ví dụ

```
let s = "Javascript is simple, Javascript is great!";

console.info(s.indexOf("Javascript")); // 0
console.info(s.lastIndexOf("Javascript")); // 22
console.info(s.search("\\s{2}")); // -1
console.info(s.includes("simple")); // true
```

Template literals

- Template literals giúp cho việc thực hiện nối chuỗi trong JS dễ dàng, trực quan hơn.
- Sử dụng cặp dấu `` và truyền các đối số cho các tham số với cú pháp ${}.

```
function show(firstName, lastName) {
    return `Hello, ${firstName} ${lastName}`;
}

console.log(show("Thanh", "Duong"));
// Hello, Thanh Duong
```


Số

- Chuyển số thành chuỗi
  - toString()
  - toFixed()
  - toPrecision()
- Chuyển chuỗi thành số
  - Number()
  - parseFloat()
  - parseInt()

Mảng

- Trong JS, mảng (array) chứa dãy các phần tử có thể có kiểu dữ liệu khác nhau.
  - Khai báo mảng: a = []
  - Truy cập phần tử thứ i: a[i]
  - Lấy số lượng phần tử của mảng: a.length

Mảng

- Các phương thức thông dụng

- push()/unshift()
- pop()/unshift()
- indexOf(x [, fromIndex])
- lastIndexOf(x, [, endIndex])
- reverse()
- sort()
- forEach(function(value, index) {})
- slice(beginIndex, endIndex)
- concat()
- join(seperator)

Mảng

- Một số cú pháp array destructuring trong ES6

```
// Gán các phần tử mảng cho các biến
const [a, b, c] = [15, 25, 10];
console.info(a, b, c); // 15 25 10
const [, d] = [true, "good", 1, 2, 3];
console.info(d); // good
const [e, f, ...r] = [9, 8, 1, 2, 3]
console.info(e, f, r); // 9 8 [1, 2, 3]

// Hoán vị hai phần tử
let [x, y] = [100, 200]
[x, y] = [y, x]
console.info(x, y); // 200 100
```


Date

- Đối tượng Date để tương tác với dữ liệu ngày, giờ (year, month, day, hour, minute, second, mili second).
- Các cách thức tạo đối tượng Date
  - new Date() → Trả về ngày hiện tại
  - new Date(miliseconds)
  - new Date(dateString)
  - new Date(year, month, day, hours, minutes, seconds, miliseconds)

Date

- Ví dụ

```
// YYYY-MM-DD
var d1 = new Date("2017-05-27");
// YYYY-MM
var d2 = new Date("2017-05");
// YYYY
var d3 = new Date("2017");
// YYYY-MM-DDTHH:MM:SSZ
var d4 = new Date("2017-05-27T10:30:05Z");
```


Date

- Một số phương thức thông dụng
  - getDay() → get day of week (0-6)
  - getDate() → get day of month (1-31)
  - getMonth() → get month
  - getFullYear() → get four-digits year (yyyy)
  - getHours() → get hours
  - getMinutes() → get minutes
  - getSeconds() → get seconds
  - getMilliseconds() → get mili seconds

Hàm

- JS sử dụng tử khoá function để định nghĩa hàm.

```
function funcName([Parameters]) {
    [statements]
    [return statement]
}
```

- Sử dụng hàm bằng cách gọi hàm thông qua tên hàm và truyền danh sách đối số tương ứng.

Hàm

- Nếu một tham số không được truyền đối số tương ứng, nó sẽ có giá trị là undefined.
- JS cho phép khai báo tham số có giá trị mặc định (default value).

```
function add(a, b = 5) {
    return a + b;
}

console.log(add(5)); // 10
console.log(add(5, 7); // 12
```

Literal Function

- Literal function được giới thiệu từ JS 1.2

```
var varName = function([Parameters]) {
   [statements]
   [return statement]
}
```

- Ví dụ

```
var sum = function(a, b) {
   return a + b;
}
console.info(sum(2, 3));
```


Javascript Closures

- JS hỗ trợ các hàm lồng nhau. Một closure là hàm trong (inner function) có thể truy cập vào các biện của các hàm ngoài (outer function).

```
function User(name){
   var displayName = function(greeting){
      console.log(greeting + '  + name);
   }
   return displayName;
}

var myFunc = User('Harvey');
myFunc('Welcome '); //Output: Welcome Harvey
```

Arrow Function

- Arrow function là cách mới để định nghĩa hàm trong JS. Nó giúp định nghĩa hàm ngắn gọn, rõ ràng bỏ qua những khai báo không cần thiết.
- Chú ý arrow function:
  - Không có kết buộc riêng tới this và super.
  - Không phù hợp sử dụng cho các phương thức call, apply, bind.
  - Không thể sử dụng cho các phương thức như constructor.

Arrow Function

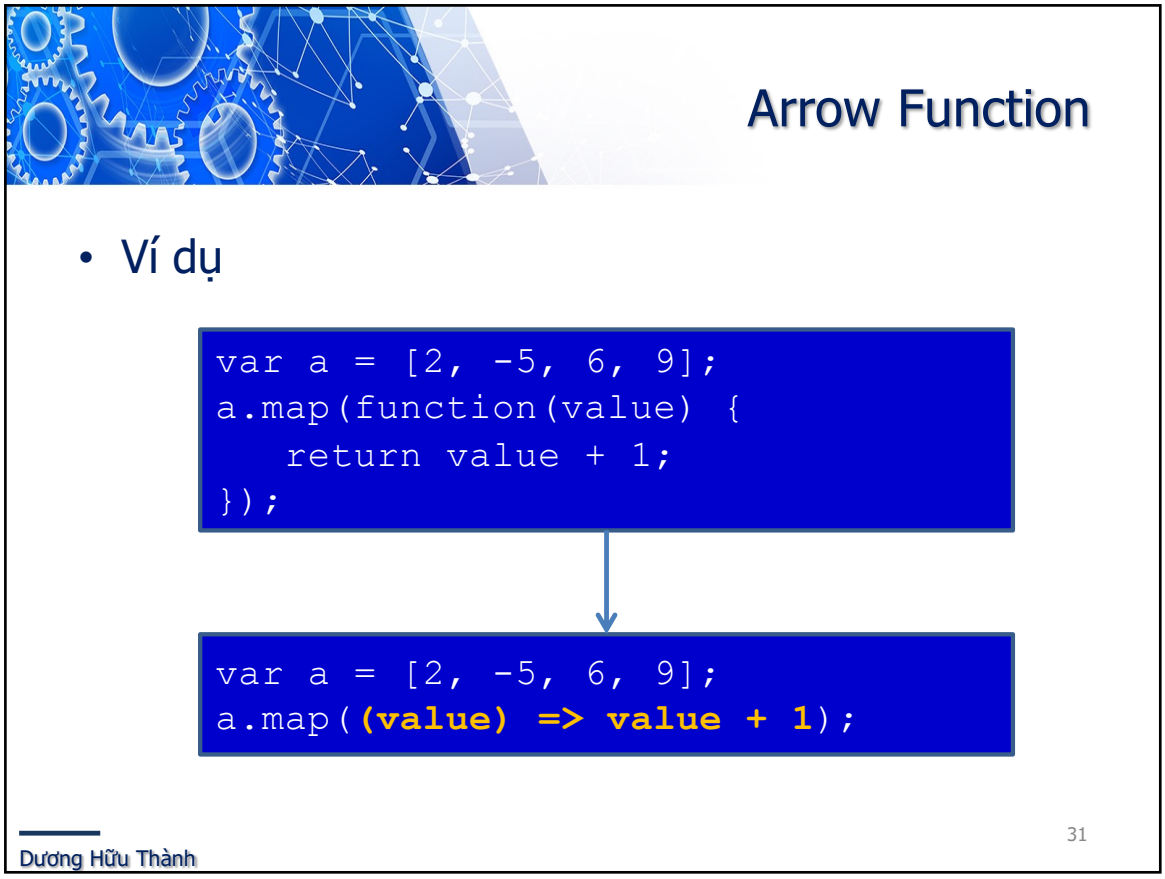
- Cú pháp

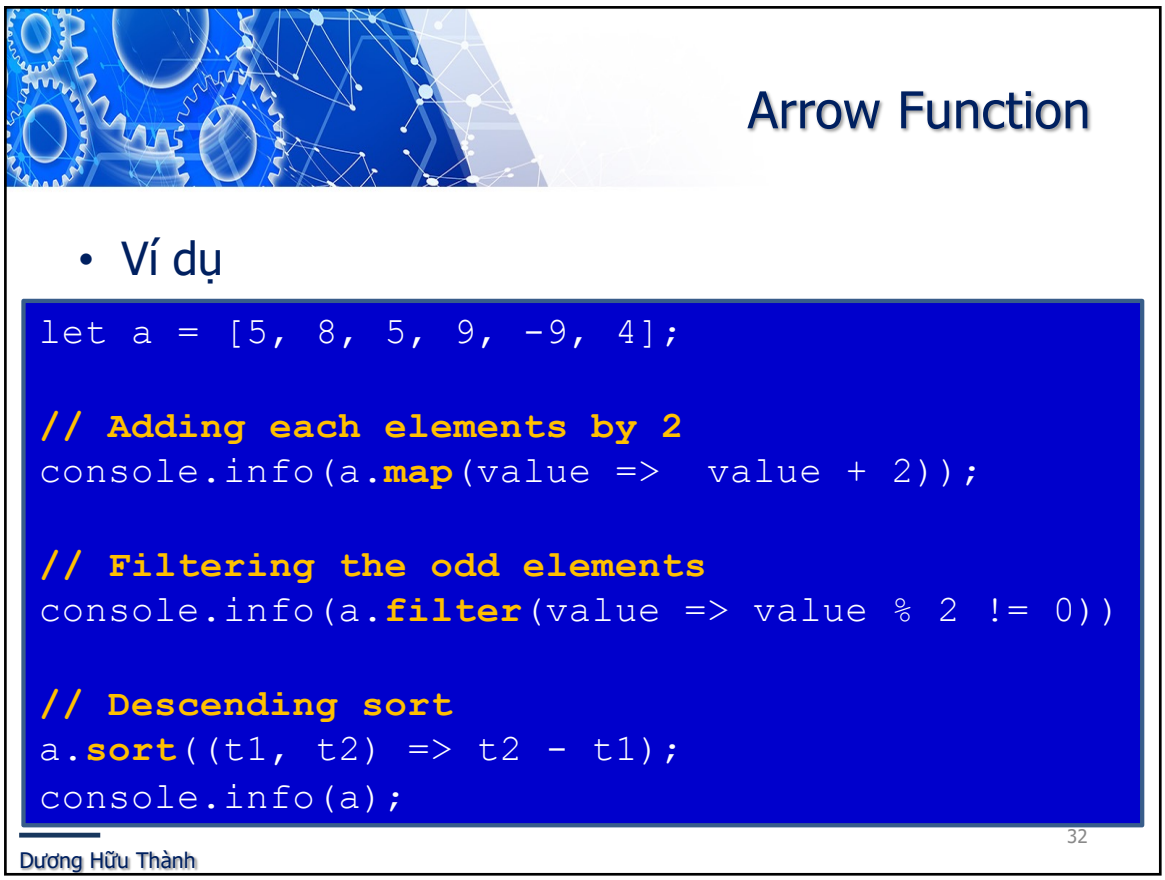
```
param -> expression
```

```
(param1, ..., paramN) -> expression
```

```
function add(a, b = 5) {
   return a + b;
}
```

```
var add = (a, b = 5) => a + b;
```


Arrow Function
• Ví dụ
var a = [2, -5, 6, 9];
a.map(function(value) {
return value + 1;
});
var a = [2, -5, 6, 9];
a.map((value) => value + 1);
31
Dương Hữu Thành


Arrow Function
• Ví dụ
let a = [5, 8, 5, 9, -9, 4];
// Adding each elements by 2
console.info(a.map(value => value + 2));
// Filtering the odd elements
console.info(a.filter(value => value % 2 != 0))
// Descending sort
a.sort((t1, t2) => t2 - t1);
console.info(a);
32
Dương Hữu Thành

Arrow Function

- Ví dụ: viết chương trình chỉ giữ lại các phần tử xuất hiện một lần trong mảng.

```
let a = [5, 6, 7, 6, 6, 5, 9]
let b = a.filter(x =>
          a.indexOf(x) == a.lastIndexOf(x))
console.info(b)
```


Hàm Callback

- Hàm callback là một hàm có thể truyền vào hàm khác như là đối số.

```
var add = (a, b) => a + b;
var sub = (a, b) => a - b;

function execute(callback) {
   var a = 10, b = 20;
   return callback(a, b);
}

console.log(execute(add)); // 30
console.log(execute(sub));  // -10
```

Hàm Callback

- Hàm callback được sử dụng để tiếp tục xử lý nào đó sau khi thực thi một tác vụ bất động bộ (asynchronous) hoàn tất.
- Ví dụ trong cú pháp jQuery

```
setTimeout(function() {
  console.info("calling the callback func");
}, 1000)
```


Multiple Expression

- JS cho phép thực thi nhiều biểu thức trên cùng dòng, thứ tự thực hiện từ trái sang phải và trả về giá trị từ biểu thức cuối cùng.
- Ví dụ

```
let x = 5;
function addFive(num) {
  return num + 5;
}
x = (x++ , x = addFive(x), x *= 2);

console.log(x) // 22
```

Đối tượng

- JS hỗ trợ lập trình hướng đối tượng, các đối tượng có sẵn hoặc do người dùng tạo đều kế thừa từ kiểu Object.

```
var student = {
    fn: "Le",
    ln: "Duong",
    fullName: function() {
        return `${this.fn} ${this.ln}`;
    }
}
```


Đối tượng

- Từ khoá this tham chiếu đối tượng hiện tại đang gọi thực thi hàm.
- Sử dụng toán tử new để tạo thể hiện (instance) của đối tượng.

```
var user = new Object();
user.firstName = "Thanh";
user.lastName = "Duong";
```

Đối tượng

- Ví dụ

```
function user(name) {
    this.name = name;
    this.hello = function() {
        return "Hello " + this.name;
    }
}
var u = new user("Thanh");
console.info(u.hello());
```


Đối tượng

- Sử dụng spread operator trong ES6

```
const info = {
  firstName: "Thanh",
  lastName: "Duong"
}

const student = {
  ...info,
  "country": "Vietnam",
  "city": "HCMC"
}

console.info(student)
```

→

```
{
  firstName: 'Thanh',
  lastName: 'Duong',
  country: 'Vietnam',
  city: 'HCMC'
}
```

Đối tượng

- Truy cập các thuộc tính và phương thức đối tượng.

```
<object>.<attr-name>
```

```
<object>["<attr-name>"]
```

```
<object>.<method-name>([<arguments>])
```


Đối tượng

- JSON (Javascript Object Notation) được sử dụng rộng rãi trong truyền và nhận dữ liệu web server.
- Các phương thức
  - Phương thức JSON.parse(jsonText) chuyển chuỗi json thành đối tượng JS.
  - Phương thức JSON.stringify(jsObject) chuyển đối tượng JS thành chuỗi json.

JS Prototype

- JS prototype cho phép thêm các thuộc tính và phương thức vào constructor đối tượng.
- Tất cả các đối tượng JS kế thừa các thuộc tính và phương thức từ prototype.
- Một đối tượng JS sẽ được thêm thuộc tính __proto__ bởi JS engine. __proto__ trỏ tới đối tượng prototype của hàm constructor.

JS Prototype

- Ví dụ

```
function Student(name, yearOfBirth) {
    this.name = name;
    this.year = yearOfBirth;
}
Student.prototype.getAge = function() {
    var d = new Date();
    return d.getFullYear() - this.year;
}
var s = new Student("Nguyen Van A", 1995);
console.log(s.getAge());
```

JS Prototype

- Ví dụ: thêm phương thức isDigitArray() vào các đối tượng Array.

```
Array.prototype.isDigitArray = function() {
    var isDigit = true;
    for (var i = 0; i < this.length && isDigit; i++)
        if (typeof this[i] != "number")
            isDigit = false;

    return isDigit;
};
var a = [5, 6, 3, 8, 5];
console.info(a.length); // 5
console.info(a.isDigitArray()); // true
```


Tương tác HTML DOM

- Các phương thức tìm thành phần HTML
  - document.getElementById()
  - document.getElementsByTagName()
  - document.getElementsByClassName()
  - document.getElementsByName()
  - document.querySelector()
  - document.querySelectorAll()

Tương tác HTML DOM

- Thay đổi các thuộc tính của thành phần HTML
  - Thay đổi nội dung
    - element.innerHTML = <html>
    - element.innerText = <raw-text>
  - Thay đổi giá trị thuộc tính
    - element.<property> = <value
    - element.setAttribute(<attr-name>, <value>)

Tương tác HTML DOM

- Thay đổi CSS của thành phần HTML
  - element.style.<property> = <value>
- Thêm sự kiện vào thành phần HTML
  - element.<event> = function() {}
- Lắng nghe sự kiện
  - element.addEventListener(event, function, capture)

setInterval() and setTimeout()

- setTimeout(func, duration)
  - Thực thi hàm func sau khoảng thời gian duration.
- setInterval(<function>, <duration>)
  - Thực thi hàm func sau mỗi khoảng thời gian duration.
- clearTimeout(timeoutVariable)
- clearInterval(intervalVariable)

JS Class

- JS class được giới thiệu trong ES6. Nó là một loại hàm sử dụng từ khoá class thay vì từ khoá function để tạo hàm.
- Sử dụng class có phương thức constructor() để khởi tạo đối tượng của lớp và có thể tạo các phương thức riêng trong lớp.

JS Class

```
class Point2D {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    calLen(p) {
        return Math.sqrt(Math.pow(this.x - p.x, 2)
                + Math.pow(this.y - p.y, 2));
    }
}
let p = new Point2D(2, 3);
let q = new Point2D(3, 4);
console.info(p.calLen(q)); // 1.4142135623730951
```


JS Class

- Class cũng có thể khai báo các phương thức getter và setter.
- Sử dụng từ khoá static để khai báo các phương thức tĩnh. Phương thức tĩnh được định nghĩa trong lớp không phải trên prototype. Do đó sử dụng phương thức tính dùng tên lớp.

```
class People {
    constructor(name) {
        this.name = name;
    }
    get name() {
        return this.name;
    }
    set name(n) {
        this.name = n;
    }
    static show() {

    }
}
```

JS Class

- Sử dụng từ khoá extends đề thực hiện kế thừa, lớp con sẽ kế thừa tất cả các phương thức từ lớp cha.

```
class People {
    constructor(name) {
        this.name = name;
    }
    show() {
        console.info(this.name);
    }
}
```


JS Class

```
class Lecturer extends People {
    constructor(name, degree) {
        super(name);
        this.degree = degree;
    }
    show() {
        super.show();
        console.info(this.degree);
    }
}
var lec = new Lecturer("Peter", "PhD");
lec.show();
```

Call, Apply và Bind

- Phương thức call() có thể sử dụng gọi các constructor của một đối tượng.

```
function People(name) {
   this.name = name;
}
function Lecturer(name, degree) {
   People.call(this, name);
   this.degree = degree;
}
var lec = new Lecturer("Nguyen Van A", "Dr");
console.info(`${lec.name} - ${lec.degree}`);
```


Call, Apply và Bind

- Phương thức call() dùng thực thi một hàm và chỉ định ngữ cảnh this, các đối số được truyền vào một cách riêng biệt.

```
const student = {
   name: "Thanh"
}
function hello(city, country) {
   console.info("Hello %s, %s, %s.",
       this.name, city, country);
}
hello.call(student, "HCM City", "Vietnam");
```

Call, Apply và Bind

- Phương thức apply() dùng thực thi một hàm và các đối số truyền vào dạng mảng.

```
const student = {
   name: "Thanh"
}
function hello(city, country) {
   console.info("Hello %s, %s, %s.",
            this.name, city, country);
}
hello.apply(student, ["HCM City", "Vietnam"]);
```


Call, Apply và Bind

- Ngoài ra, ta có thể dùng cú pháp spread (...).

```
function sum(x, y, z) {
  return x + y + z;
}

const numbers = [4, 5, 6]
console.info(sum.apply(null, numbers))
console.info(sum(...numbers)) # spread syntax
```

```
// destructuring assignment
[a, b, ...data] = [2, 3, 10, 20, 30];
console.info(a) // 2
console.info(b) // 3
console.info(data) // [10, 20, 30]
```

Call, Apply và Bind

- Phương thức bind() trả về thể hiện của phương thức và nó có thể thực thi phương thức đã kết bước với các đối số chỉ định.

```
const student = {
   name: "Thanh"
}
function hello(city, country) {
   console.info("Hello %s, %s, %s.",
        this.name, city, country);
}
var bound = hello.bind(student);
bound("HCM City", "Viet Nam");
```


Module Pattern

- Module là một tập tin JS, giữa các module có thể nạp và sử dụng các hàm của nhau.
- Module giúp chương trình tăng khả năng tái sử dụng, dễ bảo trì.
- Sử dụng từ khoá export đánh dấu một biến hay hàm có thể truy cập từ bên ngoài của module hiện hành.
- Sử dụng từ khoá import để nạp các biến hay hàm từ module khác để sử dụng.

Module Pattern

- Có hai loại export khác nhau: named và default. Trong module có thể có nhiều export dạng named, nhưng chỉ có một export dạng default.
- Export dạng named thường dùng cho export nhiều giá trị, khi import bắt buộc sử dụng cùng tên của đối tượng tương ứng.

Module Pattern

```
// hello.js
export function hello(name) {
   console.info("Hello %s!!!", name);
}
export default function add(a, b) {
   return a + b;
}
```

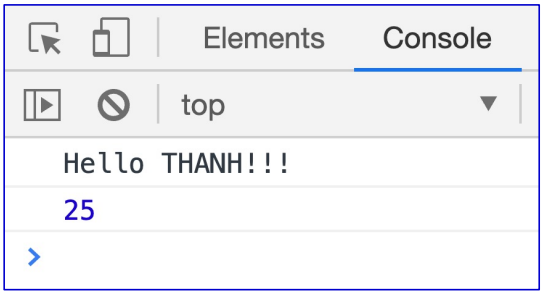
```
// demo.js
import add, {hello} from "./hello.js";

hello("Thanh");
console.info(add(5, 20));
```

Module Pattern

- Trong HTML:

```
<script type="module" src="demo.js"></script>
```

- Kết quả



Synchronous và Asynchronous

- Đồng bộ (Synchronous): nhiệm vụ tiếp theo chỉ được thực hiện chỉ khi nhiệm vụ hiện tại hoàn tất.
- Bất đồng bộ (Asynchronous): không chờ mỗi nhiệm vụ hoàn tất, tất cả các nhiệm vụ sẽ thực thi cùng lúc. Kết quả của mỗi nhiệm vụ sẽ được xử lý ngay khi nó hoàn tất.

Promise

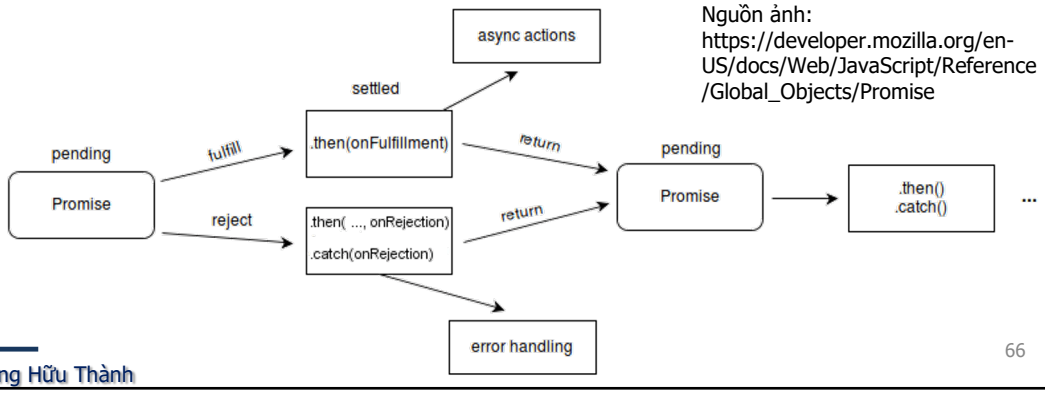
- Promise là một cơ chế cho phép thực thi các nhiệm vụ bất đồng bộ.

```
new Promise(function(resolve, reject) {
    ...
})
```

- Hàm resolve sẽ được gọi khi promise hoàn tất.
- Hàm reject sẽ được gọi khi promise thất bại.

Promise

- Các trạng thái của một promise
  - pending: trang thái khởi động.
  - fulfilled (resolved): nhiệm vụ hoàn tất thành công.
  - rejected: nhiệm vụ thất bại.



Nguồn ảnh: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Promise

- Ví dụ

```
new Promise(resolve => resolve(1)).then((k) => {
   console.info(k);
   return k += 10;
}).then((k) => {
   console.info(k);
}).then(() => console.info("asychronous"));
console.log('synchronous');
```

```
synchronous
1
11
asychronous
```


Promise

- Một số phương thức đối tượng promise cũng trả về một promise
  - Promise.prototype.then(onSuccess, onError): promise hoàn tất thành công (được gọi khi thực thi resolve trong Promise).
    - Gọi onSuccess nếu promise hoàn tất.
    - Gọi onError nếu promise xảy ra lỗi.
  - Promise.prototype.catch(error): promise bị từ chối (rejected) (được gọi khi thực thi reject trong Promise).

Promise

- Phương thức window.fetch() tạo một request giống XMLHttpRequest(xhr), nhưng phương thức này làm việc dựa trên Promise.
- Ví dụ

```
fetch('/api/categories').then((res) => {
  if (res.status == 200)
    res.json().then(data => console.info(data));
  else
    console.err("error status: " + res.status);
}).catch(err => {
  console.err(err);
});
```


Promise

```
fetch('/api/categories', {
    method: 'POST',
    body: JSON.stringify({
        name: "TEST"
    }),
    headers: {
        'Content-Type': 'application/json'
    }
}).then(res => res.json()).then((res) => {
    console.info(res);
}).catch(err => {
    console.error(err);
});
```

Async và Await

- Xử lý bất đồng bộ trong JS có nhiều phiên bản khác nhau:
  - ES5 sử dụng callback.
  - ES6 sử dụng promise.
  - ES7 sử dụng async và await.
- Nền tảng của async và await là promise. Mỗi hàm async sẽ trả về promise.

Async và Await

- Từ khoá async được sử dụng khai báo hàm bất đồng bộ.
- Từ khoá await được sử dụng tạm dừng để chờ hàm bất đồng bộ.

```
async function hello(name) {
   return "Hello " + name;
}
// true
console.info(hello("A") instanceof Promise);
// Hello A
hello("A").then((k) => console.info(k))
```

Async và Await

- Ví dụ

```
async function getCategories() {
  const cates = await fetch('/api/categories');
  const data = await cates.json();
  console.info(data);
}
```

- Khi sử dụng async/await để xử lý lỗi, ta kết hợp try/catch.

Axios

- Axios là một HTTP Client hoạt động dựa trên promise, nó có thể làm việc trên trình duyệt hoặc NodeJS phía server (isomorphic application).
- Axios cũng trả về Promise như fetch.
- Sử dụng CDN cho Axios

```
<script
src="https://cdn.jsdelivr.net/npm/axios/dist
/axios.min.js"></script>
```

Axios

- HTTP Get

```
const BASE_URL =
"https://6008e8380a54690017fc26af.mockapi.io/"
const getItems = async () => {
  try {
    const res = await axios.get(`${BASE_URL}/item`)
    console.info(res.data)
  } catch (errors) {
    console.error(errors)
  }
}
```


Axios

- HTTP Post

```
const addItem = async () => {
  try {
    const data = {
      "name": "My Item",
      "amount": 1000,
      "category_id": 1
    }
    const res = await axios.post(`${BASE_URL}/item`, data)
    console.log(res.data)
  } catch (errors) {
    console.error(errors)
  }
}
```


Q&A
ThS. Dương Hữu Thành,
Khoa CNTT, Đại học Mở TP.HCM,
thanh.dh@ou.edu.vn.
77