**TÌM HIỂU VỀ ES6**

**2.1 Biến Let, Var, Conts trong ES6**

**Biến Let**

tạo ra một biến chỉ có thể truy cập được trong block bao quanh nó

VD:

function foo() {

let x = 10;

if (true) {

let x = 20; // x này là x khác rồi đấy

console.log(x); // in ra 20

}

console.log(x); // in ra 10

}

Ngoài ra, khi ở global scope (tức là không nằm trong một function nào cả), từ khóa var tạo ra thuộc tính mới cho global object (this), còn let thì không

* + - VD:

var x = 'global';

let y = 'global';

console.log(this.x); // "global"

console.log(this.y); // undefined

* + let rất hiệu quả đó là sử dụng callback trong một vòng lặp
    - Khi sử dụng var

for (var i = 0; i < 5; i++) {

setTimeout(function(){

console.log('Yo! ', i);

}, 1000);

}

Kết quả :

Yo! 5

Yo! 5

Yo! 5

Yo! 5

Yo! 5

* + - Khi sử dụng let

for (let i = 0; i < 5; i++) {

setTimeout(function(){

console.log('Yo! ', i);

}, 1000);

}

Kết quả :

Yo! 0

Yo! 1

Yo! 2

Yo! 3

Yo! 4

**Biến Var trong ES6**

Biến var trong ES6 có thể tạo ra 1 biến có phạm vi truy cập xuyên suốt 1 function chứa nó.

Ví dụ:

function khoiTao()

{

var x = 5;

if(true)

{

var x = 10;// x khởi tạo ở đây giống với x đã khởi tạo trước đó

console.log(x);// in ra giá trị là 10

}

Console.log(x); // vẫn in ra giá trị 10.

}

**Biến Const trong ES6** : dùng để khai báo một hằng số, nếu một biến được khai báo là hằng số thì phải gán giá trị lúc khai báo luôn và sẽ không thể thay đổi giá trị cho biến đó được nửa.

Ví dụ

const A = 2;

A = 4;

Chạy lên sẽ nhận thông báo lỗi Uncaught TypeError: Assignment to constant variable

* 1. **Tìm hiểu về Arrow function**
* Arrow function
  + là kiểu cú pháp giúp đơn giản hoá việc việc định nghĩa hàm
  + Cách dùng arrow function
  + Trường hợp nhiều tham số:
    - Từ khoá function được bỏ đi
    - Thêm ký tự => (gần giống với mũi tên) đặt giữa dấu ngoặc kết thúc danh sách tham số ) và dấu ngoặc bắt đầu logic của hàm {
    - thay từ khoá var bằng const để quy định rằng giá trị của sumNumbers sẽ không thay đổi
    - VD:

//ES5

var phraseSplitterEs5 = function phraseSplitter(phrase) {

return phrase.split(' ');

};

//ES6

var phraseSplitterEs6 = phrase => phrase.split(" ");

console.log(phraseSplitterEs6("ES6 Awesomeness"));

// ["ES6", "Awesomeness"]

* + - Trường hợp 1 tham số:

//ES5

var docLogEs5 = function docLog() {

console.log(document);

};

//ES6

var docLogEs6 = () => { console.log(document); }

docLogEs6();

* + - Trường hợp không tham số:

//ES5

var phraseSplitterEs5 = function phraseSplitter(phrase) {

return phrase.split(' ');

};

//ES6

var phraseSplitterEs6 = phrase => phrase.split(" ");

console.log(phraseSplitterEs6("ES6 Awesomeness"));

// ["ES6", "Awesomeness"]

**2.3 Tìm hiểu về biến Object**

Trong Javascript, ta có 5 kiểu dữ liệu cơ bản và 1 kiểu dữ liệu phức hợp. 5 kiểu dữ liệu cơ bản bao gồm: Number, String, Boolean, Undefined và Null. Kiểu dữ liệu phức hợp là kiểu dữ liệu Object.

Kiểu dữ liệu Object được sử dụng rất thường xuyên trong Js vì nó rất biến hoá và mạnh mẽ, đây là khái niệm cơ sở cho hàng loạt các đặc điểm nổi bật khác của Javascript

Về mặt định nghĩa, một đối tượng (một object) là một **danh sách các item**, mỗi item là một cặp name-value, trong đó value có thể là: các kiểu dữ liệu cơ bản, function, hay cũng có thể là một object khác (kiểu dữ liệu phức hợp).

Ta gọi mỗi item là một property(thuộc tính) của object nếu value của item đó có kiểu dữ liệu là kiểu phức hợp hoặc các kiểu dữ liệu cơ bản, ngược lại nếu value của item nó là một hàm (một function) thì ta gọi nó là method của object (phương thức của object).

Ví dụ:

var myFirstObject = {

firstName: "Nam",

favoriteFood: "Fish"

};

Cách truy cập đến các thuộc tính của Object

Để lấy được các giá trị của thuộc tính trong object, ta có thể truy cập tới tên của thuộc tính bằng toán tử ngoặc vuông [ ] hoặc dấu chấm ., minh hoạ như sau:

myFirstObject.firstName; *//Nam*

myFirstObject['favoriteFood']; *//Fish*

Truy cập tới các thuộc tính không tồn tại của object sẽ cho ta giá trị là undefined.

Có một trường hợp nên chú ý, nếu tên của thuộc tính là số (tức là number), thì ta chỉ có thể truy cập bằng cách dùng dấu ngoặc vuông [ ] mà **không thể** dùng dấu chấm ., xem minh hoạ sau để hiểu rõ:

var myObject = {10: 'test1'}

myObject.10; *//Lỗi*

myObject['10']; *//test1*

Cách để tạo ra 1 Object

Thông thường, ta có 2 cách để tạo ra một object: dùng object literals và dùng Object constructor

**Dùng Object literals**

Mình không rõ tiếng Việt gọi literal là gì, nhưng đại khái là sẽ dùng cặp ngoặc nhọn { } để tạo một object. Literal có thể hiểu theo ngữ cảnh nào đó là việc “sử dụng chuỗi thuần tuý”. Đoạn code sau minh hoạ cho điều này:

var myBook = {10: 'test1'}; *//đối tượng có 1 thuộc tính*

*//đối tượng có 1 thuộc tính và 1 phương thức (method)*

var myCar = {

brand: 'Toyota',

run: function(){

console.log('running');

}

};

**Dùng Object constructor**

Cách này sẽ sử dụng phương thức khởi tạo (constructor) của kiểu dữ liệu Object để tạo ra các object. Phương thức khởi tạo này là một hàm để tạo ra các object mới, ta dùng kèm từ khoá new:

*//Tạo 1 đối tượng mới*

var myApple = new Object();

*//Thêm các thuộc tính cho đối tượng*

myApple.color = 'red';

myApple.shape = 'round';

myApple.howSweet = function(){

console.log('I am sweet');

};

Trong Js, thì các thuộc tính của object có thể là các **kiểu dữ liệu cơ bản**, các **phương thức**, hay cũng có thể là các **object** con. Kiểu dữ liệu Object cho ta một cách sử dụng rất linh hoạt và mạnh mẽ.

**2.4 Khai báo Class trong ES6**

**1, Class Definition**.

Với ES6 thì bạn không thể khai báo các thuộc tính như bình thường được mà bạn chỉ có thể gán nó vào các phương thức trong đối tượng được thôi.

Ví dụ 1: Khai báo một biến có kiểu dữ liệu là object

var employee = {

name: null;

age: null,

setName: function (name) {

this.name = name;

},

getName: function () {

return this.name;

},

setAge: function (age) {

this.age = age;

},

getAge: function () {

return this.age;

},

Ví dụ 2: Chuyển sang dạng ES6

class Employee {

setName (name) {

this.name = name;

}

getName () {

return this.name;

}

setAge (age) {

this.age = age;

}

getAge () {

return this.age;

}

};

Để khởi tạo đối tượng được khai báo theo chuẩn ES6 thì các bạn sử dụng từ khóa new

**Cú pháp :**

**new ClassName; hoặc new ClassName();**

Và với ES6, nó cũng hỗ trợ chúng ta một phương thức đặc biệt mà bất kỳ ngôn ngữ lập trình nào cũng có đối với class đó là constructor - phương thức khởi tạo. constructor trong ES6 cũng có tác dụng tương tự, nó sẽ tự động được gọi khi đối tượng được khởi tạo.

Ví dụ

class Employee {

constructor (name, age) {

this.name = name;

this.age = age;

}

setName (name) {

this.name = name;

}

getName () {

return this.name;

}

setAge (age) {

this.age = age;

}

getAge () {

return this.age;

}

};

**2, Class Inheritance.**

Với ES6, nó cũng đã cung cấp cho chúng ta sử dụng từ khóa extends để kế thừa từ đối tượng khác.

**Cú Pháp**:

class A extends B {

}

**Trong đó**: A là class đang được khai báo, và nó kế thừa lại các thuộc tính và phương thức từ class B.

Ví dụ

class MaleEmployee extends Employee {

constructor (name,age,wifeName) {

super(name,age);

this.wifeName = wifeName;

}

setWifeName (wifeName) {

this.setWifeName;

}

getWifeName () {

return this.wifeName;

}

}

**3, Base class access.**

Trong ES6, để thực hiện gọi các phương thức trong lớp cha khi đang ở lớp con, mà phương thức đó đã bị rewrite trong lớp con rồi thì các bạn sử dụng keyword super với cú pháp như sau:

super.methodName();

Trong đó, methodName là phương thức của lớp cha mà bạn muốn gọi.

Ví dụ

class Employee {

getClassName () {

return "Class Employee";

}

};

class MaleEmployee extends Employee {

getClassName () {

return "Class MaleEmployee";

}

classClassName () {

return super.getClassName();

}

}

var employee = new MaleEmployee();

console.log(employee.classClassName());

**4, Static Members.**

Trong ES6 cũng hỗ trợ chúng ta khai báo các thành phần tĩnh cho đối tượng, bằng cách sử dụng từ khóa static ở trước tên phương thức.

**VD**:

class Employee {

static defaultEmployee () {

return "Đây là phương thức defaultEmployee";

}

};

Và khi một phương thức được khai báo là static methods thì chúng ta sẽ không thể gọi phương thức đó một cách thông thường được nữa, mà chúng ta sẽ gọi theo cú pháp sau:

className.staticMethodName();

Trong đó:

* className là tên của class chứa static methods mà bạn muốn gọi..
* staticMethodName là tên của static methods mà bạn muốn gọi.

VD: Mình sẽ gọi phương thức defaultEmployee của class Employee ở trên.

console.log(Employee.defaultEmployee());

**5, Setter and Getter.**

Để khai báo các setter và getter trong ES6 chúng ta sử dụng keyword set và get trước tên các phương thức mà bạn muốn thiết lập nó là setter hoặc getter.

Ví dụ

class Employee {

constructor (name, age) {

this.name = name;

this.age = age;

}

set employeeName (name) {

this.name = name;

}

get employeeName () {

return this.name;

}

set employeeAge (age) {

this.age = age;

}

get employeeAge () {

return this.age;

}

};

Hoặc bạn cũng có thể thực hiện setter, getter bằng cách truy cập trực tiếp vào class mà không cần khởi tạo.