Objects

Ba Nguyễn

What is object?

Object (đối tượng) là kiểu dữ liệu đặc biệt, ***object mô phỏng một đối tượng thực tế trong ngôn ngữ lập trình***, mỗi object bao gồm 2 phần: ***properties*** *(thuộc tính)* và ***methods*** *(phương thức)*

Thông tin về object được lưu trữ dưới dạng các cặp ***key: value***. ***Key có kiểu string, value có thể là bất kỳ kiểu dữ liệu nào***.

Property là một thông tin mô tả về đối tượng, *một object có thể có nhiều properties.*

Method là một hành động (chức năng) mà đối tượng có thể thực hiện, *method* là một *key* có *value* là một ***function***

Mọi thông tin của object đều được truy xuất thông qua key

Objects

Mỗi ***person*** là một object, có các properties như ***name***, ***age***, ***job***, ... và các methods như ***speak***, ***laugh***, ***eat***, ...

Mỗi ***computer*** là một object, có các properties như ***brand***, ***series***, ***size***, ***cpu***, ***memory***, ..., và các methods như ***start***, ***shutdown***, ...,

Mỗi ***cat*** là một object, có các properties như ***name***, ***breed***, ***weight***, ..., và các methods như ***meow***, ***run***, ***bite***, ...,

Mỗi ***character*** là một object, có các properties như ***name***, ***level***, ***weapon***, ***damage***, ... và các methods như ***attack***, ***run***, ...

Objects

```
// Khai báo Object rỗng
let myComputer = {}; // literal syntax
// Hoặc khai báo kèm thông tin
// Mỗi cặp key: value được phân tách bằng dấu ,
let myComputer = {
    type: "Laptop",
    brand: "Dell",
    "operating system": "Window 10",
    start: function () {
        console.log("Startup");
    },
};
```

Objects

```
/* Để truy cập thông tin trong object, sử dụng cú pháp:
 *  - Dot Notation: object.key
 *  - Bracket Notation: object["key"] */
// Thêm thuộc tính
myComputer.cpu = "i7-4800HQ";
// Thay đổi thuộc tính
myComputer.brand = "Dell Workstation";
// Xóa thuộc tính
delete myComputer.ram;
// Truy cập giá trị
console.log(myComputer.brand); // Dell Workstation
console.log(myComputer["operating system"]); // Window 10
myComputer.start(); // Startup
```

Objects

💡 **Note**

- ***Key*** được lưu với kiểu dữ liệu ***string*** (mọi kiểu dữ liệu khác đều được chuyển về kiểu ***string***)
- ***Key*** không bị giới hạn về đặt tên giống như biến, ***nó có thể chứa ký tự đặc biệt, hoặc trùng với keyword*** trong Javascript
- Cú pháp truy cập thông tin trong object: ***object.key*** hoặc ***object[key]***
- Có thể *thêm, thay đổi giá trị, hoặc xóa* một property khỏi object
- Khi truy cập một property không tồn tại trong object, giá trị trả về sẽ là ***undefined***

```
// Cú pháp Bracket Notation: object[key] thường dùng với biến
let key = "operating system";
obj[key]; // obj["operating system"]
obj[key] = "Linux"; // obj["operating system"] = "Linux"
```

Objects

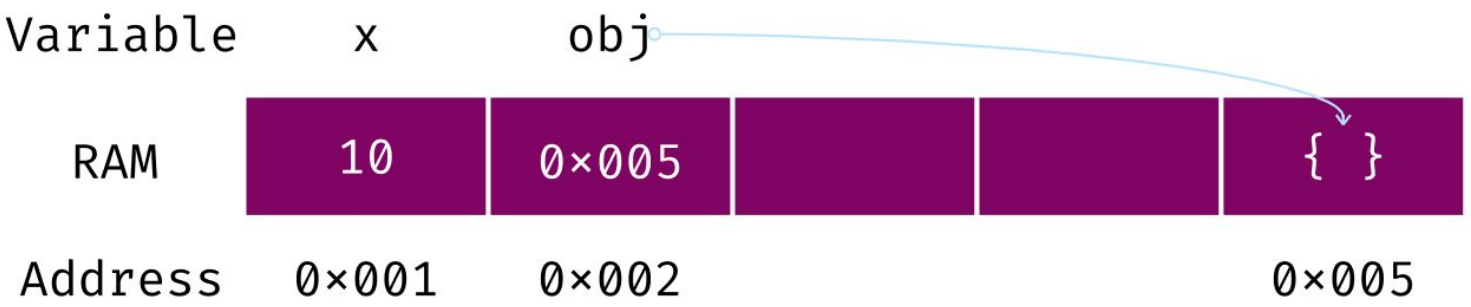
```
// Kiểm tra một key có tồn tại trong object hay không
// Sử dụng toán tử in
"type" in myComputer; // true
"cpu" in myComputer; // true
"ram" in myComputer; // false

// Lặp qua tất cả (*) thuộc tính trong object
// Sử dụng vòng lặp for in
for (let prop in myComputer) {
    console.log(prop + ": " + myComputer[prop]);
}
// type: Laptop
// brand: Dell Workstation
// ...
```

Object References

Một biến lưu trữ đối tượng *không thực sự lưu trữ thông tin về đối tượng đó*, mà lưu trữ *địa chỉ vùng nhớ của đối tượng* trong bộ nhớ máy tính (*tham chiếu - reference*), trong khi các kiểu dữ liệu nguyên thủy (*primitive*) biến lưu trữ trực tiếp giá trị

Khi truy xuất thông tin đối tượng, JavaScript di chuyển tới địa chỉ thực của đối tượng thông qua biến và thao tác trên đó



Object References

Khi so *sánh/sao chép* một object, nó so *sánh/sao chép địa chỉ vùng nhớ (tham chiếu)*

Khi 2 biến cùng tham chiếu tới một đối tượng, một biến thay đổi giá trị của đối tượng, biến còn lại cũng nhận được sự thay đổi đó

```
let copy = myComputer; // Copy reference
console.log(copy == myComputer); // true
copy.type = "Desktop";
console.log(myComputer.type); // Desktop
```

Object References

```
// Reference vs Primitive khi sử dụng với function
let x = 10;
let obj = { x: 10 };

function func(pri, ref) { // pri = x, ref = obj
    pri = 100; // x không đổi
    ref.x = 100; // obj.x thay đổi
}
func(x, obj);

console.log(x); // 10
console.log(obj.x); // 100
```

Copying Object

```
// Sao chép giá trị trong object sử dụng for in
let copy = {};

for (let prop in myComputer) {
    copy[prop] = myComputer[prop];
}

copy.type = "Desktop";
console.log(myComputer.type); // Laptop
```

Copying Object

```
/*
 * Sao chép giá trị trong object
 * sử dụng Object.assign(target, source)
 */
let copy = Object.assign({}, myComputer);
copy.type = "Desktop";
console.log(myComputer.type); // Laptop

// ??? Nếu một thuộc tính là object khác
```

Property Flags

Property được lưu bởi một ***key***, ngoài ***value*** lưu trữ giá trị, *property* còn bao gồm các ***flags*** khác

- ***writeable***: nếu ***true***, giá trị có thể thay đổi, nếu ***false*** thì thuộc tính chỉ có thể đọc
- ***enumerable***: nếu ***true***, thuộc tính xuất hiện trong vòng lặp ***for in*** hoặc ***Object.assign()***, nếu ***false*** thì không
- ***configurable***: nếu ***true***, thuộc tính *có thể xóa hoặc thay đổi*, nếu ***false*** thì không

Mặc định khi khai báo object và thuộc tính với ***literal syntax***, các ***flags*** được đặt thành ***true***

```
Object.getOwnPropertyDescriptor(myComputer, "type");

/**
 * value: 'Laptop'
 * writeable: true
 * enumerable: true
 * configurable: true
 */
```

Property Flags

```
// Để thay đổi giá trị các flags, sử dụng
// Object.defineProperty(obj, prop, descriptor)
// Object.defineProperties(obj, properties)
let obj = Object.defineProperties({}, {
    x: {
        value: 10,
        writable: true,
        // Các flags không xuất hiện mặc định là false
    },
    // ... other properties
});
```

Property Flags

```
/**
 * Khi sao chép object với for in hoặc Object.assign()
 * nó chỉ sao chép giá trị của thuộc tính
 * để sao chép cả giá trị và các flags khác
 * sử dụng kết hợp Object.defineProperties()
 * và Object.getOwnPropertyDescriptors()
 */

let copy = Object.defineProperties(
    {},
    Object.getOwnPropertyDescriptors(obj)
);
```

Object Methods

Method là một hành động (chức năng) mà đối tượng có thể thực hiện, *method* là một *key* có *value* là một ***function***

```
let obj = {
    a: function () { /* Code */ },
};

obj.b = function () { /* Code */ };

// Gọi phương thức của object
obj.a();
obj["b"]();
```

this

Để có thể truy cập được tới các thuộc tính trong object, các methods tồn tại một ***toán tử (biến - từ khóa)*** *đặc biệt* - ***this*** - *tham chiếu tới chính object gọi nó*, thông qua ***this***, methods có thể truy cập được tới các thuộc tính trong object

```
let obj = {
    x: 10,
    printX: function () {
        console.log(this.x); // obj.x
    },
};

obj.printX(); // 10
```

this

Các hàm được khai báo bên ngoài object cũng tồn tại biến ***this***

```
function printX() { console.log(this.x); }

let obj = {
    x: 10,
    printX: printX,
};

obj.printX(); // 10
```

💡 Giá trị của ***this*** không được xác định khi khai báo function, nó xác định khi method được gọi. Nếu một function không được gọi bởi một object, ***this*** sẽ là đối tượng đặc biệt ***globalThis***

Getter vs Setter

Ngoài các properties thông thường (*data properties*), object có thể có các properties khác (*accessor properties*) là các hàm được biệt được thực thi khi muốn lấy giá trị thuộc tính (***getter***) hoặc khi muốn cập nhật giá trị cho thuộc tính (***setter***)

```
let obj = {
    get prop() {
        return this._prop;
    },
    set prop(value) {
        if (!isValid(value)) { return; }
        else { this._prop = value; }
    }
};
obj.prop = "Olala"; // setter
console.log(obj.prop); // getter
```

Getter vs Setter

Note

- ***Getter*** và ***Setter*** là 2 function đặc biệt, đối với mã bên ngoài object, nó giống như thuộc tính thông thường
- ***Getter*** cho phép tùy chỉnh giá trị trả về khi mã bên ngoài muốn truy cập giá trị của một thuộc tính
- ***Setter*** cho phép thêm các logic xử lý khi mã bên ngoài muốn cập nhật giá trị của một thuộc tính (validate dữ liệu phải hợp lệ, format lại dữ liệu, ...)

Constructor Function

Hàm khởi tạo (***Constructor function***) cho phép tạo ra nhiều đối tượng dựa trên một *bản mẫu* với danh sách *properties và methods* được xác định trước

```
// Khai báo
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.hi = function () {
        console.log("Hi, I'm " + this.name);
    };
}
// Khởi tạo object với từ khóa new
let ba = new Person("Ba", 29);
```

Object to primitive

JavaScript hỗ trợ chuyển đổi kiểu dữ liệu tự động, đối với object cũng tương tự. Object được tự động chuyển đổi về kiểu ***primitive*** khi sử dụng các *built-in function* hoặc *toán tử* yêu cầu kiểu dữ liệu ***primitive*** sử dụng các phương thức đặc biệt ***toString()*** và ***valueOf()***

- Đối với các *built-in function và toán tử cần kiểu dữ liệu* ***string***: object ***ưu tiên*** gọi phương thức ***toString()***, nếu không có ***toString()*** thì gọi phương thức ***valueOf()***
- Đối với các *built-in function và toán tử cần kiểu dữ liệu* ***number***: object ***ưu tiên*** gọi phương thức ***valueOf()***, nếu không có ***valueOf()*** thì gọi phương thức ***toString()***

Object to primitive

```
let obj = {
    name: "Ba",
    age: 29,
    toString: function () { return this.name; },
    valueOf: function () { return this.age; }
}
console.log("Teacher " + obj);; // Teacher Ba
console.log(obj + 1); // 30
```

Object

```
// Một số phương thức đặc biệt với Objects
Object.keys(myComputer);
// ["type", "brand", ...]
Object.values(myComputer);
// ["Laptop", "Dell", ...]
Object.entries(myComputer);
// [ ["type", "Laptop"], ["brand", "Dell"], ...]
Object.fromEntries(entries);
// { ... }
```