Cú pháp class căn bản

Chúng ta đã được học về constructor function, giúp chúng ta tạo được nhiều object cùng một loại như user, car,... Nhưng hôm nay, chúng ta có một cú pháp mới hơn, nhiều tính năng hơn đó là class

Cú pháp class

```
class MyClass {
  // class methods
  constructor() { ... }
  method1() { ... }
  method2() { ... }
  method3() { ... }
  ...
}
```

Sau đó sử dụng `new MyClass()` để tạo một object mới với tất cả phương thức được liệt kê. Phương thức `constructor()` được gọi tự động bằng `new`, vì thế chúng ta có thể khởi tạo object ngay tại đây.

Ví dụ:

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

  sayHi() {
    alert(this.name)
  }
}

// Usage:
let user = new User('John')
user.sayHi()
```

Khi `new User("John")` được gọi:

1. Một object mới được tạo
2. `constructor` chạy với tham số được đưa vào và gán `this.name` vào nó.

Sau đó chúng ta có thể gọi tất cả các phương thức như là `user.sayHi()`

Lưu ý: Không có dấu phẩy giữa các phương thức (method)

Class là gì?

Vậy chính xác thì `class` là gì?

Trong Javascript, class là một loại function

Ví dụ:

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

// class is a function
alert(typeof User) // function

// ...or, more precisely, the constructor method
alert(User === User.prototype.constructor) // true

// The methods are in User.prototype, e.g:
alert(User.prototype.sayHi) // alert(this.name);

// there are exactly two methods in the prototype
alert(Object.getOwnPropertyNames(User.prototype)) // constructor, sayHi
```

Không chỉ là một syntactic sugar

Thỉnh thoảng một số người nói rằng `class` là một "syntactic sugar" (cú pháp mà được thiết kế để làm mọi thử dễ đọc hơn chứ không giới thiệu tính năng gì mới), bởi vì chúng ta có thể khai báo y hệt mà không dùng từ khóa `class`

```
// viết lại class User bằng function cơ bản

// 1. Tạo constructor function
function User(name) {
  this.name = name
}
// Một function prototype mặc định có thuộc tính "constructor",
// Vậy nên không cần tạo nó

// 2. Thêm phương thức vào prototype
User.prototype.sayHi = function () {
  alert(this.name)
}

// Sử dụng:
let user = new User('John')
user.sayHi()
```

Kết quả tương tự với class. Nhưng vẫn có một số điểm khác biệt:

1. Đầu tiên, một function được tạo bởi `class` thì được gắn nhãn như là một function đặc biệt, nó chỉ được gọi bằng `new`

```
class User {
  constructor() {}
}

alert(typeof User) // function
User() // Error: Class constructor User cannot be invoked without 'new'
```

Cũng như chuỗi đại diện cho một class constructor trong hầu hết JS engine bắt đầu là "class..."

```
class User {
  constructor() {}
}

alert(User) // class User { ... }
```

2. Các phương thức class thì non-enumerable (không liệt kê). Một class set cờ `enumerable` là `false` cho tất cả phương thức trong `"prototype"`. Điều này là tốt vì nếu `for...in` qua một object, chúng ta thường không muốn nó lặp qua các phương thức class.

3. Các class luôn luôn `use strict`. Tất cả code bên trong class thì tự động ở chế độ strict mode.

Bên cạnh đó, cú pháp `class` mang đến nhiều tính năng khác mà chúng ta sẽ khám phá sau.

Class Expression

Giống như function, class cũng có thể được khai báo như thế này.

```
let User = class {
  sayHi() {
    alert('Hello')
  }
}
```

Nếu một class expression có một tên, nó chỉ hiện bên trong class

```
// "Named Class Expression"
// (no such term in the spec, but that's similar to Named Function Expression)
let User = class MyClass {
  sayHi() {
    alert(MyClass) // MyClass name is visible only inside the class
  }
```

```
}

new User().sayHi() // works, shows MyClass definition

alert(MyClass) // error, MyClass name isn't visible outside of the class
```

Chúng ta có thể tạo class theo kiểu này

```
function makeClass(phrase) {
  // declare a class and return it
  return class {
    sayHi() {
      alert(phrase)
    }
  }
}

// Create a new class
let User = makeClass('Hello')

new User().sayHi() // Hello
```

Getter/setter

Giống như literal object, class cũng có getter/setter, thuộc tính computed

Đây là ví dụ cho `user.name` được thực hiện bằng sử dụng `get/set`

```
class User {
  constructor(name) {
    // invokes the setter
    this.name = name
  }

  get name() {
    return this._name
  }

  set name(value) {
    if (value.length < 4) {
      alert('Name is too short.')
      return
    }
    this._name = value
  }
}

let user = new User('John')
alert(user.name) // John
```

```
user = new User('') // Name is too short.
```

Computed names [...]

Đây là một ví dụ với một tên phương thức computed sử dụng `[...]`

```
class User {
  ['say' + 'Hi']() {
    alert('Hello')
  }
}

new User().sayHi()
```

Class fields - Các trường trong class

Đây là một tính năng mới được thêm gần đây (các trình duyệt cũ có thể phải cần một polyfill)

Trước đó thì các class của chúng ta chỉ có các phương thức, class fields cho phép chúng ta thêm bất kỳ thuộc tính.

Ví dụ, thêm thuộc tính `firstName` vào `class User`:

```
class User {
  firstName = 'John'
  constructor() {
    this.age = 24
    this.name = 'Alan'
  }
  sayHi() {
    alert(`Hello, ${this.firstName}!`)
  }
}
new User().sayHi() // Hello, John!
```

Vậy nên chúng ta chỉ cần khai báo "=", thế là xong. Điều quan trọng là các class field chỉ được set trên cá nhân của object, không phải `User.prototype`

```
class User {
  name = 'John'
}

let user = new User()
alert(user.name) // John
alert(User.prototype.name) // undefined
```

Cảnh giác với this trong class

Như chúng ta được biết thì `this` phụ thuộc vào ngữ cảnh được gọi. Vậy nên đoạn code dưới đây sẽ show `undefined`:

```
class Button {
  constructor(value) {
    this.value = value
  }

  click() {
    alert(this.value)
  }
}

let button = new Button('hello')

setTimeout(button.click, 1000) // undefined
```

Vấn đề là chúng ta bị mất `this`. Có 2 cách tiếp cận để fix điều này (như đã mô tả trong Function binding):

1. Truyền vào một wrapper function, như là `setTimeout(() => button.click(), 1000)`
2. Bind một phương thức vào object, ví dụ trong constructor.

```
class Button {
  constructor(value) {
    this.value = value
    this.click = this.click.bind(this)
  }

  click() {
    alert(this.value)
  }
}

let button = new Button('hello')

setTimeout(button.click, 1000) // undefined
```

Class fields cung cấp chúng ta một cách khác, khá dễ dàng:

```
class Button {
  constructor(value) {
    this.value = value
  }
  click = () => {
    alert(this.value)
```

```
  }
}

let button = new Button('hello')

setTimeout(button.click, 1000) // hello
```

Class field `click = () => {...}` được tạo trên mỗi object, nó là một function riêng biệt trên mỗi `Button` object với `this` bên trong tham chiếu đến object đó. Chúng ta có thể truyền `button.click` bất kỳ đâu mà không cần lo lắng về `this`, vì nó luôn đúng.

Đây là một cách rất hữu hiệu trong môi trường trình duyệt, ví dụ lắng nghe các sự kiện.

Tóm lại

Cú pháp class cơ bản như thế này

```
class MyClass {
  prop = value; // property

  constructor(...) { // constructor
    // ...
  }

  method(...) {} // method

  get something(...) {} // getter method
  set something(...) {} // setter method

  [Symbol.iterator]() {} // method with computed name
  // ...
}
```

Về mặt kỹ thuật, `MyClass` là một function (hàm mà chúng ta cung cấp dưới dạng constructor function), trong khi các phương thức, getter/setter được viết ở Myclass.prototype

Trong chương tiếp theo chúng ta sẽ học thêm về class, bao gồm kế thừa và các tính năng khác.