[Bài đọc] 3 cách định nghĩa lớp trong JavaScript

Javascript là một ngôn ngữ hướng đối tượng rất linh hoạt khi nói đến cú pháp. Trong bài này sẽ trình bày 3 cách để định nghĩa và khởi tạo một đối tượng trong Javascript. Ngay cả khi bạn đã chọn cho mình một cách sử dụng yêu thích nhất, bài viết này vẫn sẽ hữu ích cho bạn trong trường hợp muốn đọc code của người khác viết.

Một lưu ý quan trọng là không có CLASS trong Javascript. Hàm có thể được dùng để giả lập CLASS, nhưng một điều chắc chắn rằng JavaScript là một ngôn ngữ không có CLASS. Mọi thứ trong Javascript đều là đối tượng. Và khi áp dụng kế thừa, chỉ có đối tượng kế thừa đối tượng, không phải CLASS kế thừa một CLASS khác như trong các ngôn ngữ thuần lớp khác.

1. Sử dụng hàm

Đây gần như là một cách phổ biến nhất. Bạn định nghĩa một hàm, sau đó tạo đối tượng sử dụng từ khóa **new**. Để định nghĩa các thuộc tính và phương thức cho một đối tượng sử dụng từ khóa **function()** và **this** để truy xuất. Hãy xem ví dụ sau:

*//Khai báo lớp Apple với các thuộc tính type, color,*

*//phương thức getAppleInfo()*

**function** Apple (type) {

**this**.type = type;

**this**.color = "red";

**this**.getInfo = getAppleInfo;

}

*//Khai báo hàm getAppleInfo()*

*//Khuyến nghị: đây là cách không nên dùng,*

*//đọc tiếp* sẽ rõ

**function** getAppleInfo() {

**return** **this**.color + ' ' + **this**.type + ' apple';

}

Để khởi tạo một đối tượng sử dụng hàm khởi tạo **Apple**, gán giá trị cho các thuộc tính và gọi phương thức bạn có thể làm như sau:

**let** apple = **new** Apple('macintosh');

apple.color = "reddish";

**alert**(apple.getInfo());

1.1 Sử dụng hàm bên trong

Trong ví dụ trên, các bạn có thể thấy rằng hàm **getInfo()** của lớp **Apple** được định nghĩa một cách tách biệt bên ngoài lớp **Apple** thông qua hàm **getAppleInfo()**. Về hình thức, cách này vẫn hoạt động trơn tru, tuy nhiên có một bất lợi là chúng ta tạo ra rất nhiều hàm mà chúng đều ở phạm vi toàn cục. Tức là bạn sẽ phải đối mặt với vấn đề xung đột về tên hàm khi bạn (hoặc một thư viện khác bạn đang dùng) muốn tạo một hàm khác trùng tên. Để khắc phục điều này, bạn có thể khai báo phương thức của bạn ngay bên trong hàm khởi tạo như sau:

**function** Apple (type) {

**this**.type = type;

**this**.color = "red";

**this**.getInfo = function() {

**return** **this**.color + ' ' + **this**.type + ' apple';

    };

}

Sử dụng cách này không làm thay đổi cách bạn khởi tạo đối tượng và sử dụng những thuộc tính hoặc phương thức của nó.

**1.2 Khai báo hàm bằng thuộc tính prototype**

Một hạn chế của 1.1 là hàm **getInfo()** được tạo ra mỗi lần bạn tạo lại một đối tượng mới. Thỉnh thoảng đó có thể là cách bạn muốn nhưng rất hiếm gặp, điều này đồng nghĩa là nó sẽ tiêu tốn nhiều tài nguyên bộ nhớ hơn. Một cách làm tiết kiệm hơn là thêm phương thức **getInfo()** vào bên trong prototype của hàm khởi tạo.

**function** Apple (type) {

**this**.type = type;

**this**.color = "red";

}

Apple.**prototype**.getInfo = **function**() {

**return** **this**.color + ' ' + **this**.type + ' apple';

};

Một lần nữa bạn có thể sử dụng đối tượng mới như trong phần 1. và 1.1

2. Sử dụng object literals

Literals là một cách tạo đối tượng và mảng trong Javascript ngắn nhất trong Javascript. Để tạo ra một đối tượng rỗng, bạn có thể làm như sau:

**let** o = {};

thay cho các thông thường:

**let** o = **new** Object();

Đối với mảng, bạn khai báo như sau:

**let** a = [];

Thay cho cách thông thường:

**let** a = **new** Array();

Vậy bạn có thể bỏ qua cách khai báo lớp rườm rà và tạo ra một thể hiện (đối tượng) ngay tức khắc. Cách này hoạt động giống như những ví dụ trước nhưng sử dụng cú pháp như sau:

**let** apple = {

    type: "macintosh",

    color: "red",

    getInfo: **function** () {

**return** **this**.color + ' ' + **this**.type + ' apple';

    }

}

Với cách này, bạn không cần (và không thể) khởi tạo đối tượng của lớp vì đơn giản là nó đã tồn tại. Vì vậy bạn chỉ cần bắt đầu sử dụng đối tượng này thôi.

apple.color = "reddish";

alert(apple.getInfo());

Một đối tượng như thế thường được gọi là Singleton. Trong những ngôn ngữ thuần lớp như Java, singleton có nghĩa là bạn chỉ có thể có duy nhất một thể hiện (đối tượng) của một lớp tại bất kỳ thời điểm nào, bạn không thể tạo ra nhiều hơn một đối tượng của cùng một lớp. Trong Javascript (nhớ rằng, không có khái niệm lớp), khái niệm này không có ý nghĩa nữa khi tất cả đối tượng đều là duy nhất ngay từ khi khai báo.

3. Singleton sử dụng một hàm

Cách thứ ba được trình bày trong bài viết này là sự kết hợp của hai cách trên mà bạn đã thấy. Bạn có thể sử dụng một hàm để xác định một đối tượng singleton. Đây là cú pháp:

**let** apple = **new** **function**() {

**this**.type = "macintosh";

**this**.color = "red";

**this**.getInfo = **function** () {

**return** **this**.color + ' ' + **this**.type + ' apple';

    };

}

Cách này có vẻ giống với 1.1 mô tả ở trên, nhưng cách để sử dụng đối tượng thì lại giống cách 2.

apple.color = "reddish";

alert(apple.getInfo());

Vậy **new** **function**(){...} thực hiện 2 điều sau cùng một lúc: định nghĩa một hàm (một hàm tạo không tên) và gọi nó thông qua từ khóa new. Với cách này, có thể gây ra cho bạn chút nhầm lẫn bởi nó không quá phổ biến, nhưng đây là một lựa chọn khi bạn thật sự muốn có một hàm khởi tạo mà bạn sẽ chỉ sử dụng một lần duy nhất và không cần thiết phải khai báo tên cho nó.

Tổng kết

Bạn đã biết 3 cách để tạo ra một đối tượng trong Javascript. Nhưng hãy luôn nhớ rằng Javascript là một ngôn ngữ không có lớp. Hy vọng bài viết cho bạn cái nhìn thân thiện và dễ dàng hơn khi bạn làm việc với JavaScript.