Object Prototype Javascript – Công cụ hỗ trợ OOP cho JS

Bởi **Sơn Dương** - Tháng Chín 6, 2019

Bài này thuộc phần 5 của 8 phần trong series Tự học Javascript cơ bản

Khi nghe người ta nói về Javascript, bạn chỉ biết Javascript là ngôn ngữ lập trình hướng sự kiện, kiểu script... Thế nhưng, có một điều thú vị rằng, Javascript cũng có thể viết code theo hướng đối tượng. Để làm được điều này, bạn cần phải hiểu về Object Prototype Javascript.

Bạn sẽ không thể nào trở thành chuyên gia về Javascript nếu bạn không nắm được khái niệm Object trong Js. Chúng là nền tảng cho mọi khía cạnh của ngôn ngữ lập trình Javascript.

Trên thực tế, với bất kỳ ai, bất kì ngôn ngữ nào, thì học cách tạo và làm việc với Object là việc đầu tiên bạn nghiên cứu và thực hành.

Tuy nhiên, chắc do Javascript là ngôn ngữ kiểu "lai tạp" , cú pháp lỏng lẻo, dễ tiếp cận... nên nhiều bạn thường bỏ qua việc tìm hiểu Object, cứ nghĩ JS không có Object. Thực ra bạn đã nhầm, rất nhầm ☺

Để tìm hiểu một cách có hiệu quả về Prototype, chúng ta sẽ quay lại những khái niệm cơ bản nhất: "back to basic".

Nằm trong series: **Lập trình Javascript cơ bản**, bài viết này mình sẽ chia sẻ những kiến thức cần thiết về Prototype Javascript.

Nội dung chính của bài viết

#Object javascript là gì – Các cách tạo Object

Qua bài viết Kiểu dữ liệu và hàm trong JS, bạn đã biết được rằng, JS có kiểu dữ liệu cơ bản gồm: Number, String, Boolean, Undefined và Object.

Object là kiểu dữ liệu tổng hợp các kiểu dữ liệu nguyên thủy. Theo định nghĩa thì Object là một danh sách các property (thuộc tính). Mỗi Property là một cặp key-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).

Trong Javascript, mình có 3 cách để định nghĩa, tạo một Object. Chúng ta sẽ lần lượt điểm qua từng cách một.

>> Đọc thêm: Các kiểu dữ liệu và hàm trong Javascript

1. Sử dụng Object literal

Ví dụ như bên dưới là cách tạo đối tượng sử dụng Object literal. Đây là cách mà mình hay sử dụng nhất, vừa đơn giản, lại dễ hiểu.

```
var human = {
    firstName: "Duong",
    lastName: "Anh Son",
    age: 30,
    fullName: function(){
        return this.firstName + " " + this.lastName
    }
}
```

2. Sử dụng từ khóa new Object()

Với các ngôn ngữ khác, từ khóa new dùng để tạo một instance của một Object. Thì với Javascript, từ khóa new vừa tạo Object, vừa tạo instance.

Ví dụ về cách sử từ khóa `new Object()` :

```
var human = new Object()
human
console.log(human);// Creates an empty object
```

Để thêm thuộc tính cho Object, bạn có thể sử dụng ký hiệu dot (.) và gán giá trị luôn cho thuộc tính đó. Ví dụ cho dễ hình dung nhé:

```
var human = new Object()
human.firstName = "Duong"
human.age = 30
console.log(human);// Creates an empty object
```

3. Sử dụng Object Constructor

Khái niệm hàm khởi tạo (constructor) có lẽ không còn quá xa lạ với các bạn đã biết C# hay Java.

Contructor trong Javascript cũng có vai trò tương tự như vậy.

Để tạo một Object bằng hàm constructor, cần phải qua 2 bước:

- Định nghĩa các thuộc tính của một prototype javascript class bằng một hàm constructor. Theo Javascript convention, thì tên của hàm khởi tạo nên bắt đầu bằng chữ in hoa.
- Tạo đối tượng được định nghĩa bằng hàm constructor bằng từ khóa new.

Nhìn các bước tạo object này, mình lại có sự liên tưởng tới Kotlin. Cách làm rất giống nhau.

Đây là một ví dụ:

```
// Định nghĩa một class bằng hàm khởi tạo.
function Human(firstName, lastName) {
    this.firstName = firstName,
    this.lastName = lastName,
    this.fullName = function() {
        return this.firstName + " " + this.lastName;
    }
}

var person1 = new Human("Virat", "Kohli");

console.log(person1)
// Tạo instance bằng từ khóa new
var anhson = new Human("Duong", "Anh Son")
var sachinTendulkar = new Human("Sachin", "Tendulkar")
```

>> Tham khảo thêm: 3 cách định nghĩa hàm Javascript – bạn có biết?

3. Vấn đề khi tạo Object bằng hàm khởi tạo constructor

Chúng ta sẽ xem xét đoạn mã tạo Object bên dưới đây:

```
var person1 = new Human("Virat", "Kohli");
var person2 = new Human("Sachin", "Tendulkar");
```

Khi đoạn code thực thi, Javascript engine sẽ tạo ra 2 bản sao của hàm constructor tương ứng với person1 và person2.

Điều này có nghĩa là mọi đối tượng được tạo bằng hàm constructor sẽ có bản sao tương ứng các thuộc tính và method riêng.

Việc này dẫn đến việc phung phí tài nguyên khi mà hàm `fullName()` làm cùng một việc như nhau ở mọi bản sao.

Đề giải quyết bài toán này, chúng ta sử dụng khái niệm Prototype.

Prototypes Javascript là gì?

Prototype có hai khái niệm mà bạn cần phân biệt:

- Prototype object là một đối tượng trong Javascript.
- Thuộc tính Prototype của function.

Có một điều kì cục trong Javascript đó là: function cũng là một object. Function có một thuộc tính đặc biệt là prototype property. Và bản thân thuộc tính prototype này lại mang một giá trị object.

Ngoài ra, vì chúng ta sử dụng function để tạo hàm khởi tạo đối tượng, nên thuộc tính prototype của function cũng có khả năng đặc biệt. Đó là bạn có thể thêm các thuộc tính hoặc phương thức vào thuộc tính prototype để thực hiện kế thừa.

Cuối cùng, khi bạn tạo một object, Javascript engine sẽ mặc định thêm một thuộc tính propotype. **Thuộc tính này có giá trị trỏ tới prototype object mà nó kế thừa**. Ta dùng thuộc tính "__proto__" để truy cập tới prototype object.

1. Cách tạo propotype

Do hàm khởi tạo đối tượng cũng được xem là 1 đối tượng prototype, do đó các đơn giản để tạo ra 1 đối tượng prototype là khai báo một hàm khởi tạo:

```
function Student() {
    this.name = 'Duong Anh Son';
    this.gender = 'M';
}

Student.prototype.age = 20;

var sinhvien1 = new Student();
console.log(sinhvien1.age); // 20

var sinhvien2 = new Student();
console.log(sinhvien2.age); // 20
```

2. Changing Prototype

Như mình đã nói ở trên, mỗi một prototype object sẽ link tới một prototype object của hàm.

Nếu bạn thay đổi prototype của hàm thì chỉ các đối tượng tạo mới sau này mới bị ảnh hưởng. Còn các đối tượng đã được tạo trước đó vẫn giữ nguyên prototype cũ.

Để dễ hình dung, mời bạn xem đoạn code bên dưới đây:

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
    this.website = 'vntalking.com'
}

Student.prototype.age = 15;

var sinhvien1 = new Student();
console.log('sinhvien1.age = ' + sinhvien1.age); // 15

var sinhvien2 = new Student();
console.log('sinhvien2.age = ' + sinhvien2.age); // 15

Student.prototype = { age : 20 };

var sinhvien3 = new Student();
console.log('sinhvien3.age = ' + sinhvien3.age); // 20

console.log('sinhvien1.age = ' + sinhvien1.age); // 15
console.log('sinhvien2.age = ' + sinhvien2.age); // 15
```

3. Prototype Javascript sử dụng như nào?

Tổng kết lại thì prototype object được sử dụng nhằm 2 mục đích:

- Để tìm các thuộc tính và phương thức của một object.
- Thực hiện kế thừa trong Javascript.

```
function Student() {
    this.name = 'John';
    this.gender = 'M';
}

Student.prototype.sayHi = function(){
    console.log("Hi");
};

var studObj= new Student();
studObj.toString();
```

Ở ví dụ trên, bạn thấy rằng hàm `toString()` không được định nghĩa trong Student.

Vậy đoạn code trên có bị crash không? Làm thế nào và chương trình tìm thấy hàm `toString()` ?

Để trả lời cho câu hỏi trên, bạn cần tham khảo hình bên dưới:

Đầu tiên, JavaScript engine sẽ kiểm tra xem hàm `toString()` có sẵn trong `studObj` hay không?

Nếu nó không tìm thấy, nó sẽ sử dụng thuộc tính prototype của `studObj` để tìm tới Prototype của Student.

Cuối cùng, nếu nó vẫn không thấy hàm `toString()` được định nghĩa trong Student, nó sẽ kiểm tra trong đối tượng Object (đây là đối tượng mà tất các đối tượng khác đều kế thừa).

Như ví dụ ở đoạn code trên, chương trình sẽ gọi hàm `toString()` của đối tượng Object, và nó không in ra gì cả (vì hàm `toString()` trong Object không làm gì).

Qua cách này, bạn có thể thấy là prototypechỉ giữ duy nhất một bản sao các hàm cho tất cả các đối tượng (instances).

Và đây cũng chính là câu trả lời cho câu hỏi ở đầu bài viết: tại sao sử dụng prototype lại tiết kiệm tài nguyên hơn so với cách sử dụng hàm khởi tạo constructor.

4. Kế thừa trong Javascript bằng cách sử dụng Propotype

Kế thừa là một khái niệm quan trọng trong lập trình hướng đối tượng. Trong kế thừa cổ điển (ví dụ như Java), các hàm trong class cha sẽ được copy vào lớp con.

Với Javascript thì có vẻ nhiều người cứ nghĩ ngôn ngữ lập trình này không có khái niệm kế thừa. Nhưng thực tế thì có đấy.

Trong Javascript, kế thừa được hỗ trợ bởi Propotype. Đó là lý do mà một số người gọi kế thừa trong javascript là "Prototypal Inheriatance".

Để dễ hiểu hơn, chúng ta sẽ bắt đầu bằng một ví dụ:

```
function Person(firstName, lastName) {
    this.FirstName = firstName || "unknown";
    this.LastName = lastName || "unknown";
};

Person.prototype.getFullName = function () {
    return this.FirstName + " " + this.LastName;
}
```

Ở ví dụ này, mình định nghĩa một class Person với 2 thuộc tính là FirstName và LastName, và một hàm `getFullName()`. Hàm `getFullName()` được thêm vào class thông qua prototype object.

Và giờ chúng ta sẽ tạo một class Student, kế thừa từ class Person.

```
function Student(firstName, lastName, schoolName, grade)
{
    Person.call(this, firstName, lastName);

    this.SchoolName = schoolName || "unknown";
    this.Grade = grade || 0;
}
//Student.prototype = Person.prototype;
Student.prototype = new Person();
Student.prototype.constructor = Student;
```

Các bạn để ý rằng, mình đã đặt Student.prototype là một Person.

Từ khóa new tạo một đối tượng của lớp Person và cũng gán Person.prototype cho prototype object của đối tượng mới. Và cuối cùng gán đối tượng được tạo cho Student.prototype

Bây giờ bạn có thể tạo một đối tượng student sử dụng các thuộc tính và hàm của lớp Person như sau:

```
function Person(firstName, lastName) {
    this.FirstName = firstName || "unknown";
    this.LastName = lastName || "unknown";
}

Person.prototype.getFullName = function () {
    return this.FirstName + " " + this.LastName;
}
function Student(firstName, lastName, schoolName, grade)
{
    Person.call(this, firstName, lastName);

    this.SchoolName = schoolName || "unknown";
    this.Grade = grade || 0;
}
//Student.prototype = Person.prototype;
Student.prototype = new Person();
Student.prototype.constructor = Student;

var std = new Student("James","Bond", "XYZ", 10);

console.log(std.getFullName()); // James Bond
console.log (std instanceof Student); // true
console.log (std instanceof Person); // true
```

Nhìn cũng giống với kế thừa như các ngôn ngữ kinh điển (Java, C#) nhỉ!

#Tạm kết

Như vậy là chúng ta đã hiểu rõ hơn về prototype javascript object rồi đúng không? Quả thực, đây là khái niệm mà rất nhiều người cảm thấy khó khăn khi mới tiếp cận. Nhưng hãy cứ bình tĩnh và đừng nản chí.

Khi bạn đã quen với prototype javascript, bạn sẽ cảm thấy đây là một khái niệm tuyệt vời. Bạn có cảm thấy thế không? Để lại bình luận chia sẻ cảm nghĩ với mọi người nhé.

Bài viết sau, chúng ta sẽ cùng nhau tìm hiểu một thế mạnh khác của Javascript đó là Ajax. Đừng quên đón đọc đấy!

Xem tiếp các bài trong Series

Phần trước: Tìm hiểu cách sử dụng promise trong Javascript

Phần kế tiếp: Ajax là gì? Sử dụng Ajax jquery cho web app như thế nào?

Đăng kí nhận sách dạy kiếm tiền từ ứng dụng di động

Bạn đã bao giờ tự hỏi Hà Đông kiếm tiền tỷ từ Flappy Bird như nào không? Bạn biết cách lập trình nhưng lại không biết làm sao để kiếm được tiền từ nó?
Hiện cuốn sách đang bán rất chạy trên Amazon với giá 9.99$. Bạn có muốn nhận cuốn sách này để học kiếm tiền không? Đăng kí để nhận miễn phí nhé

Email

DOWNLOAD

Sơn Dương

Tên đầy đủ là Dương Anh Sơn. Tốt nghiệp ĐH Bách Khoa Hà Nội. Mình bắt đầu nghiệp coder khi mà ra trường chẳng xin được việc đúng chuyên ngành. Mình tin rằng chỉ có chia sẻ kiến thức mới là cách học tập nhanh nhất. Các bạn góp ý bài viết của mình bằng cách comment bên dưới nhé !