Prototypal inheritance

Trong lập trình, chúng ta thường muốn lấy cái gì đó và kế thừa nó. Ví dụ chúng ta có object `user` có vài thuộc tính và phương thức, chúng ta muốn tạo một object `admin` cũng có vài thuộc tính và phương thức như `user` mà không cần tạo mới lại chúng, vậy nên chúng ta kế thừa chúng.

Prototypal inheritance (kế thừa nguyên mẫu) là tính năng giúp chúng ta làm được điều này.

[[Prototype]]

Trong Javascript, Object có một thuộc tính ẩn là gọi là `[[Prototype]]`(nó được giới thiệu như vậy). Dù bị ẩn nhưng chúng ta vẫn có nhiều cách để sử dụng nó. Một trong những cách đó là thông qua một getter/setter là `__proto__`. Ví dụ tạo object `rabbit` kế thừa thuộc object `animal`

```
let animal = {
  eats: true
}
let rabbit = {
  jumps: true
}

rabbit.__proto__ = animal // // sets rabbit.[[Prototype]] = animal

alert(rabbit.eats) // true
alert(rabbit.jumps) // true
```

Javascript Engine tìm thuộc tính `eat` trong `rabbit` nhưng không có, vì thế nó vào tìm trong `[[Prototype]]` của `rabbit`, bay giờ `[[Prototype]]` của `rabbit` là `animal`, vì thế nó lấy từ `animal`

Ví dụ về prototype chain

```
let animal = {
  eats: true,
  walk() {
    alert('Animal walk')
  }
}

let rabbit = {
  jumps: true,
  __proto__: animal
}

let longEar = {
  earLength: 10,
  __proto__: rabbit
}
```

```
// walk được lấy từ prototype chain
longEar.walk() // Animal walk
alert(longEar.jumps) // true (từ rabbit)
```

Lưu ý với `__proto__`

- Trước ECMAScript 2015 không có cách chính thống nào truy cập trực tiếp đến prototype của một object. Vậy nên hầu hết các trình duyệt thêm vào một thuộc tính bộ truy cập (accessor property) được gọi là `__proto__`
- `__proto__` không phải là thuộc tính `[[Prototype]]`, nó chỉ là getter/setter của `[[Prototype]]`
- `__proto__` gần đây đã bị loại bỏ và không được khuyên dùng nữa, nhưng vì một số lý do một số trình duyệt và môi trường vẫn giữ chúng lại. Chúng ta nên sử dụng `Object.getPrototypeOf/Object.setPrototypeOf` (ECMAScript 2015)

Giá trị của 'this'

Ví dụ đoạn code phía dưới

```
let user = {
  name: 'John',
  surname: 'Smith',

  set fullName(value) {
    ;[this.name, this.surname] = value.split(' ')
  },

  get fullName() {
    return `${this.name} ${this.surname}`
  }
}

let admin = {
  __proto__: user,
  isAdmin: true
}

alert(admin.fullName) // John Smith (*)

// setter triggers!
admin.fullName = 'Alice Cooper' // (**)

alert(admin.fullName) // Alice Cooper, trạng thái của admin đã bị thay đổi
alert(user.fullName) // John Smith, trạng thái của user vẫn giữ nguyên
```

Lý giải cho điều này là `this` trong `this.name` và `this.surname` không đại diện cho prototype

Không quan trọng phương thức ở đâu: trong object hay prototype của nó. Trong phương thức được gọi, `this` luôn đại diện cho object trước dấu chấm

Vì thế `admin.fullName=` sử dụng `admin` như `this`, không phải `user`

```
// animal has methods
let animal = {
  walk() {
    if (!this.isSleeping) {
      alert(`I walk`)
    }
  },
  sleep() {
    this.isSleeping = true
  }
}

let rabbit = {
  name: 'White Rabbit',
  __proto__: animal
}

// modifies rabbit.isSleeping
rabbit.sleep()

alert(rabbit.isSleeping) // true
alert(animal.isSleeping) // undefined (no such property in the prototype)
```

for...in loop

Vòng lặp `for...in` cũng lặp các thuộc tính kế thừa

```
let animal = {
  eats: true
}

let rabbit = {
  jumps: true,
  __proto__: animal
}

// Object.keys chỉ return các key
alert(Object.keys(rabbit)) // jumps

// for..in lặp luôn cả key và key kế thừa
for (let prop in rabbit) alert(prop) // jumps, sau đó eats
```

Nếu chúng ta muốn loại bỏ các thuộc tính kế thừa thì có thể dùng phương thức có sẵn là obj.hasOwnProperty(key): nó return `true` nếu key đó thuộc object (không phải từ kế thừa)

```
let animal = {
  eats: true
}

let rabbit = {
  jumps: true,
  __proto__: animal
}

for (let prop in rabbit) {
  let isOwn = rabbit.hasOwnProperty(prop)

  if (isOwn) {
    alert(`Our: ${prop}`) // Our: jumps
  } else {
    alert(`Inherited: ${prop}`) // Inherited: eats
  }
}
```

Ở đây ta có prototype chain như sau: `rabbit` kế thừa từ `animal`, `animal` kế thừa từ `Object.prototype` (bởi vì `animal` là một literal object `{...}`, vì thế nó là mặc định), cuối cùng `Object.prototype` kế thừa từ `null`

Có một điều khá thú vị là phương thức `rabbit.hasOwnProperty` đến từ đâu? Chúng ta không định nghĩa nó mà. À, hóa ra nó đến từ `Object.prototype.hasOwnProperty`. Chúng ta đã kế thừa nó.

..Nhưng tại sao `hasOwnProperty` không xuất hiện trong `for...in` như `eats` và `jumps` Câu trả lời rất đơn giản: nó không được liệt kê. Tất cả thuộc tính từ `Object.prototype` thì được set cờ là `enumerable:false`.

Lưu ý: **Hầu hết các phương thức get key/value sẽ bỏ qua thuộc tính kế thừa, ví dụ như Object.keys, Object.values**

Tham khảo: https://javascript.info/prototype-inheritance