Property flags and descriptors

Như chúng ta biết thì Object lưu trữ các thuộc tính, các thuộc tính chỉ đơn giản là các cặp "key-value". Nhưng ngoài ra thì object còn chứa nhiều điều hay ho phía sau.

Property flags - các cờ thuộc tính

Thuộc tính object, bên cạnh một `value` còn có một số thuộc tính đặc biệt (gọi là các cờ):

- **`writable`** - nếu true, giá trị có thể thay đổi, còn không thì chỉ đọc (read-only)
- **`enumerable`** - nếu true, thì thuộc tính được liệt kê khi lặp, còn không thì không được liệt kê
- **`configurable`** - nếu true, thuộc tính có thể bị xóa, `writable` và `enumerable` có thể bị thay đổi, còn không thì không thể

Một thuộc tính sẽ có đầy đủ 3 cờ và value, người ta gọi 4 cờ đó là bộ mô tả thuộc tính (property descriptor)

Phương thức `Object.getOwnPropertyDescriptor` cho phép chúng ta lấy đầy đủ thông tin về thuộc tính

```
let descriptor = Object.getOwnPropertyDescriptor(obj, propertyName)
```

```
let user = {
  name: 'John'
}
let descriptor = Object.getOwnPropertyDescriptor(user, 'name')
console.log(descriptor)
/* property descriptor:
{
  configurable: true,
  enumerable: true,
  value: "John",
  writable: true
}
*/
```

Để thay đổi cờ, chúng ta có thể sử dụng `Object.defineProperty`

```
Object.defineProperty(obj, propertyName, descriptor)
```

`descriptor` : Một object miêu tả các thuộc tính, nếu không cung cấp thì mặc định là false

```
let user = {}
Object.defineProperty(user, 'name', {
  value: 'John'
```

```
})
let descriptor = Object.getOwnPropertyDescriptor(user, 'name')
console.log(descriptor)
/*
{
  configurable: false,
  enumerable: false,
  value: "John",
  writable: false
}
*/
```

Non-writable - Không thể thay đổi

Cùng làm `user.name` không thể bị assign lại bằng cách thay đổi cờ `writable` sang `false`

```
let user = {
  name: 'John'
}

Object.defineProperty(user, 'name', {
  writable: false
})

user.name = 'Pete' // Error: Cannot assign to read only property 'name'
```

Lưu ý: Nếu không trong chế độ strict-mode thì sẽ không xảy ra lỗi, nhưng quá trình trên sẽ không thực hiện thành công

Đây là ví dụ thay đổi cờ `writable` ngay từ lúc tạo

```
let user = {}

Object.defineProperty(user, 'name', {
  value: 'John',
  enumerable: true,
  configurable: true
})

console.log(user.name) // John
user.name = 'Pete' // Error
```

Non-enumerable - Không thể liệt kê

Cùng custom `toString` của `user`

```
let user = {
  name: 'John',
  toString() {
    return this.name
  }
}

// Mặc định thì cả 2 thuộc tính sẽ được liệt kê trong vòng lặp
for (let key in user) console.log(key) // name, toString
```

Bây giờ chúng ta không thích `toString` xuất hiện khi trong vòng lặp, ta có thể set `enumerable:false`

```
let user = {
  name: 'John',
  toString() {
    return this.name
  }
}

Object.defineProperty(user, 'toString', {
  enumerable: false
})

// Bây giờ thì toString biến mất
for (let key in user) console.log(key) // name
```

non-enumerable cũng có hiệu lực trong `Object.keys`

```
alert(Object.keys(user)) // name
```

Non-configurable - Không thể cấu hình

Ý tưởng của `configurable: false` là để ngăn chặn thay đổi cờ thuộc tính và xóa thuộc tính, trong khi đó bạn vẫn có thể thay đổi value của thuộc tính. Cờ non-configurable (`configurable:false`) thỉnh thoảng được set mặc định trong các Object xây dựng sẵn. Ví dụ `Math.PI` thì non-writable, non-enumerable, non-configurable

```
let descriptor = Object.getOwnPropertyDescriptor(Math, 'PI');

console.log(descriptor)
/*
{
  value: 3.141592653589793,
  writable: false,
  enumerable: false,
  configurable: false
```

```
}
*
```

Vì thế chúng ta không thể thay đổi giá trị của `Math.PI` hoặc viết lại nó

```
Math.PI = 3 // Error

// Không thể xóa Math.PI
delete Math.PI
```

Một khi chúng ta set một thuộc tính là non-configurable (`configurable:false`) thì chúng ta không thể thay đổi lại bằng `defineProperty`

Khi `configurable: false` thì chúng ta sẽ bị một số hạn chế trên `defineProperty`

1. Không thể thay đổi cờ `configurable`
2. Không thể thay đổi cờ `enumerable`
3. Không thể thay đổi `writable: false` sang `true` (ngược lại thì ok)
4. Không thể thay đổi `getter/setter`

Đây là `user.name` non-configurable, nhưng chúng ta còn có thể thay đổi (vì nó writable)

```
let user = {
  name: 'John'
}

Object.defineProperty(user, 'name', {
  configurable: false
})

user.name = 'Pete' // Chạy ok
delete user.name // Error
```

Và bây giờ chúng ta làm cho `user.name` mãi mãi bị "niêm phong"

```
let user = {
  name: 'John'
}

Object.defineProperty(user, 'name', {
  writable: false,
  configurable: false
})

// Không thể thay đổi user.name và cờ của nó
// Ngay cả việc xóa user.name
user.name = 'Pete'
```

```
delete user.name
Object.defineProperty(user, 'name', { value: 'Pete' })
```

Object.defineProperties

Object.defineProperties(obj, descriptors) cho phép chúng ta định nghĩa nhiều thuộc tính cùng 1 lúc Cú pháp:

```
Object.defineProperties(obj, {
  prop1: descriptor1,
  prop2: descriptor2
  // ...
})
```

Ví dụ

```
Object.defineProperties(user, {
  name: { value: 'John', writable: false },
  surname: { value: 'Smith', writable: false }
  // ...
})
```

Object.getOwnPropertyDescriptors

Để lấy tất cả property descriptor cùng một lần, chúng ta có thể sử dụng Object.getOwnPropertyDescriptors(obj)

Kết hợp với `Object.defineProperties` nó có thể được sử dụng như một cách để clone một object

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

Bình thường khi clone một object, chúng ta thường copy thuộc tính như thế này

```
for (let key in user) {
  clone[key] = user[key]
}
```

Nhưng điều này không copy cờ. Vì thế nếu muốn một cách clone tốt hơn thì `Object.defineProperties` là một sự lựa chọn tối ưu. Một sự khác biệt khác là `for...in` bỏ qua các thuộc tính tượng trưng, nhưng `Object.getOwnPropertyDescriptors` sẽ return về tất cả property descriptors bao gồm loại tượng trưng.

Sealing an object globally - Niêm phong một object toàn bộ

Object.preventExtensions(obj) Ngăn cấm thêm thuộc tính mới vào object

Object.seal(obj) Ngăn cấm thêm/xóa các thuộc tính. Set `configurable: false` cho tất cả thuộc tính đang tồn tại

Object.freeze(obj) Ngăn cấm thêm/xóa/thay đổi các thuộc tính. Set `configurable: false, writable: false` cho tất cả các thuộc tính đang tồn tại

Object.isExtensible(obj) Return `false` nếu việc thêm thuộc tính bị cấm, còn không thì `true`.

Object.isSealed(obj) Return `true` nếu thêm/xóa thuộc tính bị cấm, và tất cả thuộc tính tồn tại có `configurable: false`

Object.isFrozen(obj) Return `true` nếu thêm/xóa/sửa thuộc tính bị cấm, và tất cả thuộc tính hiện tại là `configurable: false, writable: false`

Những phương thức này thì hiếm khi được sử dụng trong thực tế

Tham khảo

https://javascript.info/property-descriptors#sealing-an-object-globally