Kiểu Symbol

Bài gốc: Kiểu Symbol trong Javascript, có thể bạn chưa từng nghe thấy

Theo như mô tả thì các key của object chỉ có thể là kiểu string hoặc symbol. Không phải number, không phải boolean, chỉ là string và symbol.

Thường thì chúng ta chỉ sử dụng string. Bây giờ cùng xem lợi ích mà symbol có thể mang lại cho chúng ta.

Symbol là gì

Một "symbol" đại diện cho một định danh duy nhất.

Một giá trị của kiểu này có thể được tạo bằng cách sử dụng `Symbol()`

```
// id thuộc kiểu symbol
let id = Symbol()
```

Chúng ta có thể truyền một đoạn mô tả vào symbol (gọi là tên symbol), việc này rất hữu hiệu cho việc debug

```
// id là một symbol với mô tả là "id"
let id = Symbol('id')
```

Các symbol thì được cam kết duy nhất. Ngay cả khi chúng ta tạo nhiều symbol với cùng mô tả, chúng cũng khác nhau về giá trị. Mô tả chỉ như là dán nhãn mác thôi, không có tác động gì cả.

Ví dụ 2 symbol cùng mô tả – chúng cũng không bằng nhau.

```
let id1 = Symbol('id')
let id2 = Symbol('id')
console.log(id1 == id2) // false
```

Lưu ý: `Symbol` không tự động chuyển sang `string`.

Hầu hết giá trị trong JS hỗ trợ convert ngầm hiểu sang string. Ví dụ khi dùng `alert` cho bất cứ giá trị nào, nó sẽ gọi đến phương thức `toString()` của giá trị đó. Nhưng symbol thì đặc biệt, chúng không tự động convert.

```
let id = Symbol('id')
alert(id) // TypeError: Cannot convert a Symbol value to a string
```

Nếu chúng ta thực sự muốn hiển thị một symbol, chúng ta cần gọi `toString()` trên nó, như thế này

```
let id = Symbol('id')
alert(id.toString()) // Symbol(id), bây giờ thì hoạt động được rồi
```

Hoặc muốn show mô tả bằng cách

```
let id = Symbol('id')
alert(id.description) // id
```

Thuộc tính "ẩn"

Symbol cho phép tạo những thuộc tính ẩn của object, các thành phần khác không thuộc code chúng ta không thể truy cập hoặc ghi đè được.

Ví dụ:

```
const initState = () => ({
  id: 1
})
const state = initState()
// Vô tình một đoạn code nào đó làm thay đổi id
// Có thể là từ User hoặc thư viện ngoài
state.id = 2
// và thuộc tính id của chúng ta bị thay đổi
```

`state` thuộc về một đoạn code khác, nếu vì lý do nào đó (có thể từ người dùng hoặc thư viện ngoài) mà nó bị thay đổi thì có thể dẫn đến việc tính toán bị sai lệch. Điển hình là các game chơi trên trình duyệt khá là dễ bị hack hay cheat vì chỉ cần viết một đoạn tool nhỏ can thiệp vào 1 biến chủ đạo trong code và làm thay đổi giá trị nó là được.

Ví dụ bạn thay đổi biến `health = 100` thành `health = 99999` trong game và bạn bất tử cmnr.

Điều này rõ ràng không an toàn tí nào.

Cách fix vấn đề trên:

```
const initState = () => {
  const result = {}
  const id = Symbol('id')
  result[id] = id
  return result
}
const state = initState()
// Bây giờ chúng ta không thể truy cập được vào thuộc tính Symbol('id') nữa
// Bởi vì ở ngoài không biết giá trị id là bao nhiêu
```

Chúng ta cũng có thể tạo key là symbol trong object literal như thế này

```
const initState = () => ({
  [Symbol('id')]: 1
})
```

Thực ra về mặt kỹ thuật thì symbol không 100% là ẩn. Một phương thức có sẵn là Object.getOwnPropertySymbols(obj) cho phép chúng ta lấy tất cả các symbol. Cũng có một phương thức là Reflect.ownKeys(obj) return tất cả các key của object bao gồm cả kiểu symbol. Vì thế chúng không thực sự ẩn. Nhưng hầu hết các thư viện ngoài thì không dùng đến những phương thức này.

Symbol bị bỏ qua ở for...in

Thuộc tính symbol không tham gia vòng lặp `for...in`

Ví dụ:

```
let id = Symbol('id')
let user = {
  name: 'John',
  age: 30,
  [id]: 123
}
for (let key in user) alert(key) // name, age (không có symbol)
// Truy cập trực tiếp thì ok
alert('Direct: ' + user[id])
```

`Object.keys(user)` cũng bỏ qua chúng.

Đối ngược với điều đó, `Object.assign` copy cả thuộc tính có key là string và symbol

```
let id = Symbol('id')
let user = {
  [id]: 123
}
let clone = Object.assign({}, user)
alert(clone[id]) // 123
```

Không có nghịch lý nào ở đây cả. Đó là bởi vì thiết kế ngôn ngữ như vậy. Ý tưởng là khi chúng ta clone một object hoặc merge object, chúng ta thường muốn tất cả các thuộc tính được copy (bao gồm cả symbol như id).

Global symbol

Như chúng ta đã thấy, thường thì tất cả các symbol thì khác nhau, ngay cả khi nếu chúng có cùng tên. Nhưng thỉnh thoảng chúng ta muốn những symbol cùng tên thì cùng giá trị. Ví dụ, những phần khác nhau của app

muốn truy cập đến symbol `"id"`, điều này có nghĩa là nó phải cùng giá trị.

Để làm điều này, chúng ta có một thứ gọi là **global symbol registry**. Chúng ta có thể tạo symbol trước và truy cập sau, và nó đảm bảo rằng việc truy cập lại có thể chính xác.

Ví dụ:

```
// đọc từ global registry
let id = Symbol.for('id') // nếu symbol chưa tồn tại, nó sẽ được tạo
// đọc lại lần nữa, có thể từ một nơi nào đó trong code
let idAgain = Symbol.for('id')
// cùng một symbol
alert(id === idAgain) // true
```

Symbol bên trong registry được gọi là global symbol. Nếu chúng ta muốn một symbol toàn app, có thể truy cập ở bất kỳ đâu trong code – global symbol là tất cả những gì ta cần.

Symbol.keyFor

Với global symbol, không chỉ `Symbol.for(key)` return một symbol bằng tên, có một phương thức ngược lại là `Symbol.keyFor(sym)`: return một tên bởi global symbol

Ví dụ:

```
// get symbol bằng tên
let sym = Symbol.for('name')
let sym2 = Symbol.for('id')
// get tên bằng symbol
alert(Symbol.keyFor(sym)) // name
alert(Symbol.keyFor(sym2)) // id
```

`Symbol.keyFor` chỉ hoạt động với global symbol registry, vì thế nó không hoạt động với symbol thông thường. Nếu symbol không phải là global, nó sẽ không tìm thấy và return `undefined`.

Ngoài ra thì bất cứ symbol nào cũng có thuộc tính `description`.

Ví dụ:

```
let globalSymbol = Symbol.for('name')
let localSymbol = Symbol('name')
alert(Symbol.keyFor(globalSymbol)) // name, global symbol
alert(Symbol.keyFor(localSymbol)) // undefined, not global
alert(localSymbol.description) // name
```

Hệ thống symbol

Có nhiều phương thức symbol mà Javascript sử dụng bên trong, chúng ta có thể sử dụng chúng để tinh chỉnh các khía cạnh khác nhau của object.

Chúng thì được liệt kê trong bảng Well-know symbols :

- `Symbol.hasInstance`
- `Symbol.isConcatSpreadable`
- `Symbol.iterator`
- `Symbol.toPrimitive`
- ... và vô số khác

Tóm lại

`Symbol` là một kiểu dữ liệu nguyên thủy (primitive type) cho các định danh duy nhất.

Symbol được tạo với `Symbol()` có thể tùy chọn tham số mô tả (name)

Symbol luôn luôn cho ra các giá trị khác nhau, ngay cả khi nó cùng tên. Nếu chúng ta muốn các symbol cùng tên bằng nhau, thì hãy sử dụng global registry.

Symbol có 2 trường hợp sử dụng chính:

1. Thuộc tính ẩn của object. Các thuộc tính có key là kiểu symbol thì không xuất hiện trong `for...in`, vì thế nó không thể vô tình bị xử lý cùng với các thuộc tính khác. Ngoài ra, nó sẽ không được truy cập trực tiếp, bởi vì các đoạn code khác không có được giá trị symbol. Vì thế thuộc tính sẽ được bảo vệ bởi các yếu tố tình cờ hoặc cố ý ghi đè.
2. Có nhiều hệ thống symbol được sử dụng bởi Javascript mà ta có thể truy cập thông qua `Symbol.*`. Ví dụ sử dụng `Symbol.iterator` cho iterable object.

Tham khảo

Symbol