Bài: Kiểu dữ liệu số trong JavaScript (Phần 3) - Khái quát về kiểu dữ liệu BigInt

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu số trong JavaScript (Phần 3) - Khái quát về kiểu dữ liệu BigInt

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Ở các bài trước, Kteam đã giới thiệu tới các bạn về Các thuộc tính và phương thức với kiểu dữ liệu Number trong Javascript.

Ở bài này, các bạn sẽ được biết thêm về một kiểu dữ liệu cơ bản khác trong Javascript – **BigInt**

Nội dung

Để nắm vững nội dung bài này, các bạn cần có kiến thức về:

- Kiểu dữ liệu số (Number) trong Javascript

Nội dung mà chúng ta sẽ được tiếp cận:

- Giới thiệu về kiểu dữ liệu BigInt
- Các đặc điểm của bigInt

Giới thiệu về bigint

Về cơ bản, **BigInt** cũng tương tự như **number**, đều là số. Nhưng giữa chúng có một sự khác biệt đáng kể.

Khởi tạo một bigint

Có hai cách để khởi tạo **BigInt**

- Cách 1: Thêm kí tự "**n**" vào phía sau một số nguyên.

Javascript:

```
a = 123n
// 123n
b = -1238884n
// -1238884n
c = 1.23n // Lỗi vì BigInt chỉ có thể là các số nguyên
// c = 1.23n
    ^^^^

// Uncaught SyntaxError: Invalid or unexpected token
```

- Cách 2: Sử dụng **Constructor** bigint

Cú pháp:

> **BigInt(<value>)**

Yêu cầu:

- **<value>** phải là một giá trị nguyên, hoặc là một **Number-String**, hay có dạng một chuỗi giá trị hợp lệ.

Ví dụ:

Javascript:

```
a = BigInt(123)
// 123n
a = BigInt(-123884)
// -123884n
a = BigInt(1.23)
// Uncaught:
RangeError: The number 1.23 cannot be converted to a BigInt because it is not an integer
    at BigInt (<anonymous>)
BigInt("0xf") // chuyển từ hệ hexa sang bigint
// 15n
typeof(a)
// 'bigint'
```

Đặc điểm của bigint

Đặc điểm cơ bản nhất của **bigint** là nó cho phép làm việc với các số nguyên mà **không bị giới hạn** về độ lớn. Ngoài ra, **bigint** cũng cho phép thực hiện hầu hết các toán tử giống như trong number.

Javascript:

```
a = 12345n
// 12345n
b = 5n
// 5n
a + b
// 12350n
a * b
// 61725n
a - b
// 12340n
a % b
// 0n
a ** b
// 286718338524635465625n
```

> **Lưu ý:** Đối với phép chia, thì bigint sẽ cho kết quả khác với number (vì bigint chỉ cho phép số nguyên)

Javascript:

```
15n / 4n
// 3n
```

Bên cạnh đó, các toán tử so sánh cũng có thể được dùng một cách bình thường giữa number và bigint:

Javascript:

```
10n == 10
// true
11n < 9
// false
10n === 10n
// true
8n == 24/3
// true
```

Tuy nhiên, việc dùng các toán tử số học giữa bigint và number sẽ dẫn đến **lỗi**:

Javascript:

```
a = BigInt(100)
// 100n
a + 1
// Uncaught TypeError: Cannot mix BigInt and other types, use explicit conversions
a + 1n
// 101n
a * 2
// Uncaught TypeError: Cannot mix BigInt and other types, use explicit conversions
a * 2n
// 200n
a / 2
// Uncaught TypeError: Cannot mix BigInt and other types, use explicit conversions
```

Mở rộng:

Các toán tử trên bit cũng có thể được dùng trên **bigint** giống như là trên một number, ngoại trừ toán tử >>> (unsigned right shift operator)

Vì việc ép kiểu giữa các giá trị **Number** và giá trị **BigInt** có thể dẫn đến mất độ chính xác, nên:

- Chỉ sử dụng giá trị **BigInt** khi các giá trị lớn hơn 2^{53} được mong đợi một cách hợp lý.
- Không ép kiểu giữa giá trị **BigInt** và giá trị **Number**.

Kết luận

Ở bài này, các bạn đã được làm quen với một kiểu dữ liệu mới: **bigint**

Bài tiếp theo, chúng ta sẽ cùng nhau tìm hiểu một kiểu dữ liệu mới: Bài tập về kiểu dữ liệu số trong Javascript

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".