Bài: Kiểu dữ liệu số trong JavaScript (Phần 2) - Các thuộc tính và phương thức đối với số

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu số trong JavaScript (Phần 2) - Các thuộc tính và phương thức đối với số

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 bài trước, chúng ta đã cùng tìm hiểu về Number trong JavaScript

Bài này, Kteam xin giới thiệu đến các bạn **các thuộc tính, constructor và các phương thức của số trong JavaScript**

Nội dung

Những yêu cầu về mặt kiến thức:

- Có hiểu biết về Number trong JavaScript
- Cài đặt sẵn môi trường nodejs

Và nội dung mà chúng ta sẽ cùng tìm hiểu:

- Các thuộc tính của Number
- Constructor Number
- Các phương thức đối với số trong JavaScript

Các thuộc tính của kiểu dữ liệu Number

Kiểu dữ liệu Number cho phép tính toán chính xác nhất với các số trong khoảng $-(2^{53}-1)$ đến $(2^{53}-1)$ (tức là, với các số lớn hơn, hãy sử dụng **bigint** – một kiểu dữ liệu số khác mà Kteam sẽ giới thiệu tới các bạn ở bài sau)

Các giá trị này lần lượt là Number_MIN_SAFE_INTEGER và Number_MAX_SAFE_INTEGER.

Các bạn có thể chạy code để chứng thực điều đó.

Javascript:

```
Number.MAX_SAFE_INTEGER
// 9007199254740991
Number.MIN_SAFE_INTEGER
// -9007199254740991
```

Trong JavaScript, có 2 giá trị đặc trưng cho dương vô cùng $+\infty$ và âm vô cùng $-\infty$. Đó lần lượt là Number.POSITIVE_INFINITY và Number.NEGATIVE_INFINITY. Trong JavaScript , 2 giá trị trên được kí hiệu là **Infinity** và **-Infinity**

Javascript:

```
Number.NEGATIVE_INFINITY
// -Infinity
Number.POSITIVE_INFINITY
// Infinity
```

Constructor Number

Cú pháp:

> **Number(<value>)**

Tác dụng: tạo một giá trị kiểu **Number** mới từ **<value>**. Đó có thể là một số, hay là NaN

Một giá trị Number sẽ được tạo ra nếu **<value>** là Number-String (Quy ước những chuỗi chỉ gồm các kí tự số là **Number-String**), trong các trường hợp khác là **NaN**.

:

```
Number("1223")
// 1223
Number([1, 2, 3])
// NaN
Number({1: 1, 2:2})
// NaN
```

Các phương thức với số trong JavaScript

Ở đây Kteam sẽ chỉ đề cập tới những phương thức cơ bản nhất, để các bạn dễ hiểu và cũng dễ nắm bắt trọng tâm hơn.

Các phương thức xác định

Phương thức isNaN

Cú pháp:

> **Number.isNaN(<value>)**

Tác dụng:

- Xác định xem **<value>** có phải là **NaN** hay không.

Javascript:

```
Number.isNaN(Infinity)
// False
Number.isNaN(NaN)
// true
Number.isNaN(Number("str"))
// true
```

Phương thức isFinite

Cú pháp:

> **Number.isFinite(<value>)**

Tác dụng:

- Xác định xem giá trị được truyền có phải là một số hữu hạn hay không.
- Nếu **<value>** thuộc một trong 2 giá trị: **Infinity** hoặc -**Infinity** thì trả về false. Ngược lại trả về true. Riêng đối với NaN, thì phương thức này trả về false.

Ví dụ:

Javascript:

```
Number.isFinite(Infinity)
// false
Number.isFinite(-Infinity)
// false
Number.isFinite(1234)
// true
Number.isFinite(NaN)
// False
Number.isFinite(12)
// true
```

Phương thức isInteger

Cú pháp:

> **Number.isInteger(<value>)**

Tác dụng:

- Xác định giá trị được truyền có phải là số nguyên hay không.
- Trả về **true** nếu **<value>** là một số nguyên, ngược lại trả về false. Đối với **Infinity** và **NaN**, phương thức này cũng trả về false.

Ví dụ:

Javascript:

```
Number.isInteger(123)
// true
Number.isInteger(123.0)
// true
Number.isInteger(123.000001)
// false
Number.isInteger(NaN)
// false
Number.isInteger(Infinity)
// false
```

Các phương thức định dạng

Phương thức toExponential

Cú pháp:

> **<Number>.toExponential(<fractionDigits>)**

Tác dụng:

- Trả về một giá trị chuỗi (Number-String), là số **<Number>** ban đầu được viết dưới dạng thu gọn thành tích của một số thực nhân với lũy thừa của 10, với **<fractionDigits>** là số lượng chữ số được lấy sau dấu phẩy (trong JavaScript là dấu chấm).

> **Lưu ý:** giá trị của **<fractionDigits>** phải nằm giữa 0 và 100, giá trị này được mặc định là số chữ số thuộc phần thập phân của **<Number>**

Ví dụ:

Javascript:

```
var a = 123.1236574;
// undefined
a.toExponential() // fractionDigits mặc định
// '1.231236574e+2'
a.toExponential(3) // Lấy 3 số sau phần thập phân của a (được làm tròn). Giá trị này tương đương với 1.231*(10^2)
// '1.231e+2'
a.toExponential(5) // Lấy 5 số sau phần thập phân của a (được làm tròn). Giá trị này tương đương với 1.23124*(10^2)
// '1.23124e+2'
```

Phương thức toFixed

Cú pháp:

> **<Number>.toFixed(<digits>)**

Tác dụng:

- Làm tròn số theo **digits** (làm tròn và lấy **<digits>** số ở phần thập phân), giá trị trả về cũng là một **chuỗi**. Theo mặc định, nếu không được người dùng xác định, thì **<digits>** bằng 0.

Ví dụ:

Javascript:

```
var a = 123.1236574;
// Undefined
a.toFixed(5)
// '123.12366'
a.toFixed(4)
// '123.1237'
a.toFixed()
// '123'
```

Phương thức parseFloat

Cú pháp:

> **Number.parseFloat(<string>)**

Tác dụng:

- Tách và xuất ra một số từ chuỗi **<string>** cho trước. Chương trình sẽ lấy các kí tự từ bên trái sang bên phải của **<string>**, và lấy nhiều kí tự số nhất có thể. Nếu không thể lấy được ít nhất một kí tự số, phương thức trả về **NaN**.

Ví dụ:

Javascript:

```
Number.parseFloat("12.123tf") // Lấy 12.123, sau đó dừng vì thấy chữ cái t
// 12.123
Number.parseFloat("12.1.23tf") // Lấy 12.1, vì sau đó có dấu chấm (.) (lấy tối đa một dấu chấm)
// 12.1
Number.parseFloat(".12.123tf") // Tương tự như trên, trước dấu chấm mặc định có số 0
// 0.12
Number.parseFloat("tf") // Không lấy được bất kì kí tự số nào
// NaN
Number.parseFloat("ff1") // Không lấy được bất kì kí tự số nào
// NaN
```

Phương thức parseInt

Cú pháp:

> **Number.parseInt(<string>, <radix>)**

Tác dụng:

Phương thức này thực hiện 2 công việc sau:

- Phân tích **<string>** giống như cách phân tích của **parseFloat** (nhưng không lấy dấu chấm, tức là chỉ lấy các kí tự số từ trái sang), sau đó chuyển thành số. Nếu không lấy được bất kì kí tự nào, tự động trả về **NaN**.
- Từ số mới được chuyển và hệ cơ số **<radix>**, chuyển về giá trị tương ứng ở hệ thập phân.

> **Chú ý:** **<radix>** nằm trong khoảng 2 ...16.

Ví dụ:

Javascript:

```
Number.parseInt("1001", 2)
// 9
Number.parseInt("15ff", 16)
// 5631
Number.parseInt("15.129", 10)
// 15
Number.parseInt(".t", 10)
// NaN
```

Math object trong JavaScript

Để có thể làm việc một cách hiệu quả hơn với số, Javascript hỗ trợ một **built-in object** (đối tượng dựng sẵn) được gọi là **Math**

Vài thuộc tính của Math:

- **Math.E**: Hằng số euler, xấp xỉ 2.718
- **Math.PI**: Số Pi, xấp xỉ 3.14159
- **Math.SQRT1_2**: Căn bậc 2 của $\frac{1}{2}$, xấp xỉ 0.707.

Javascript:

```
Math.E
// 2.718281828459045
Math.PI
// 3.141592653589793
Math.SQRT1_2
// 0.7071067811865476
```

Bên cạnh các thuộc tính cơ bản, Math cũng có các phương thức giúp các bạn đơn giản hóa hơn khi làm việc với toán.

PHƯƠNG THỨC	NỘI DUNG
Math.abs(x)	Trả về $\|x\|$
Math.ceil(x)	Trả về số nguyên nhỏ nhất lớn hơn hoặc bằng x.

Math.cbrt(x)	Trả về $\sqrt[3]{x}$
Math.max(n1, n2, n3, n4, n5, ...)	Trả về số lớn nhất trong một dãy các số
Math.min(n1, n2, n3, n4, n5, ...)	Trả về số bé nhất trong một dãy các số

Ví dụ:

Javascript:

```
Math.abs(-1111);
// 1111
Math.ceil(1.23);
// 2
Math.cbrt(8);
// 2
Math.max(-1, 2, 3, -4, 1)
// 3
Math.min(-1, 2, 3, -4, 1)
// -4
```

> **Lưu ý:** Math hoạt động với **Number**, không hoạt động với **BigInt**.

Ở trên chỉ là một vài phương thức cơ bản nhất của Math. Để tìm hiểu thêm, các bạn có thể tham khảo tại: Math in JavaScript

Kết luận

Ở bài này, chúng ta đã cùng tìm hiểu kĩ hơn về số trong Javascript

Ở bài sau, chúng ta sẽ cùng tìm hiểu một kiểu dữ liệu khác: BigInt. Tuy là "khác" nhưng nó vẫn là số =))

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**".