Bài: Kiểu dữ liệu số trong JavaScript (Phần 1) - Number

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu số trong JavaScript (Phần 1) - Number

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 cùng bạn làm Bài tập về biến trong JavaScript

Ở bài này, chúng ta sẽ cùng tìm hiểu một thứ mới mẻ hơn, nhưng cũng khá quen thuộc đó là số. Cụ thể là **kiểu dữ liệu số trong JavaScript**

Nội dung

Để có thể hiểu được nội dung của bài, bạn cần có:

- Hiểu được các kiến thức về biến trong JavaScript
- Có các kiến thức về số học

Ở bài này, chúng ta sẽ cùng tìm hiểu về....

- Số trong JavaScript
- Các toán tử cơ bản đối với số trong JavaScript
- Độ ưu tiên các toán tử (+, - , *, /)
- Sự nhầm lẫn khi tính toán với số thực

Nhắc lại các kiến thức về số học

Hàng ngày, chúng ta tiếp xúc rất nhiều với số, ví dụ như: Hôm nay là ngày bao nhiêu? Bạn mua đồ ăn sáng hết bao nhiêu tiền ? Bài kiểm tra vừa rồi được bao nhiêu điểm ? .... Tất cả đều là những con số.

Với số, ta có thể thực hiện các phép so sánh, cộng, trừ, nhân, chia, và rất nhiều phép tính khác.

Số cũng được chia thành nhiều loại, trong phạm vi bài này Kteam sẽ đề cập đến **số nguyên** và **số thực**.

Số trong JavaScript

Các giá trị số trong JavaScript đều thuộc một kiểu dữ liệu duy nhất, bất kể nó là số thực hay số nguyên.

Javascript:

```
typeof(1.3)
// 'number'
typeof(10)
// 'number'
typeof(10.25)
// 'number'
typeof(0.6)
// 'number'
typeof(0)
// 'number'
```

Các toán tử đối với số trong JavaScript

Các toán tử số học

Trong bài trước, chúng ta đã cùng tìm hiểu về các toán tử so sánh.

Đối với các toán tử dùng để tính toán, thì nó thực chất cũng giống như các phép tính số học mà ta được học. Các toán tử này lần lượt là **+, -, *, /, %, **.

Javascript:

```
3 + 4
// 7
0.2 + 0.3
// 0.5
12 * 3
// 36
8 / 4
// 2
7 / 4
// 1.75
3 - 9
// -6
```

Dưới đây là một số toán tử cơ bản đối với số trong JavaScript

Kí hiệu trong JavaScript	Kí hiệu toán học	Ví dụ	Ghi chú
+	+	3 + 2 // 5	Phép cộng
-	-	1 - 2 // -1	Phép trừ
*	x	2 * 2.5 // 5	Phép nhân
/	÷	10 / 2 // 5 6 / 4 // 1.5	Phép chia
%	MOD	6 % 4 // 2 15 % 4 // 3	Chia lấy dư
**	^	2**3 // 8 4**0.5 // 2	Phép lũy thừa

Bạn cần lưu ý một số điểm sau:

- Trong phép chia, nếu đó là phép chia hết thì kết quả trả về là số nguyên, còn nếu không chia hết, thì trả về số thập phân
- Đối với phép chia cho số 0, kết quả trả về là **Infinity** hoặc -**Infinity**, phụ thuộc vào dấu của số bị chia
- Đối với phép chia lấy dư cho 0, kết quả luôn là **NaN** (Not a Number)
- Các phép tính có độ ưu tiên giống như khi ta thực hiện bên toán, riêng phép chia lấy dư được ưu tiên giống với các phép nhân, chia

Ngoài ra, nếu muốn thực hiện một phép toán với một biến và gán kết quả cho chính biến đó, ta chỉ cần dùng toán tử tính toán kèm theo toán tử gán:

Javascript:

```
> let t = 9;
undefined
> t += 9
18
> t
18
> let g = 12;
undefined
> g *= 2
24
> g
24
```

Bitwise operator.

> Đây là một phần "mở rộng". Để có thể rõ hơn, các bạn có thể tham khảo tại: Expressions and operators

Trước khi đến với các **bitwise operator**, thì mình sẽ nói sơ qua về khái niệm hệ nhị phân dành cho những ai chưa biết.

Cụ thể, có nhiều cách biểu thị các giá trị số theo nhiều hệ đếm khác nhau. Ví dụ như:

- Hệ thập phân dùng các con số 0..9 để biểu thị các số. Ví dụ: 21, 98, ...
- Hệ tam phân thì dùng các con số 0..2 để biểu diễn các giá trị, ví dụ: 210, 21.
- Hệ thập lục phân thì dùng các số từ 0..9 và các kí tự a, b, c, d, e, f để biểu diễn các số. Ví dụ: 12f, 3c, ...

Từ những ví dụ trên, chắc các bạn cũng biết cách mà hệ nhị phân biểu diễn các con số rồi phải không?

Hệ nhị phân dùng 2 số 0 và 1 để biểu diễn các giá trị. Ví dụ: 011, 100, 111, 10101, ...

Và, một giá trị số có thể được chuyển qua lại giữa các hệ đếm, ví dụ với số 50:

HỆ ĐẾM	GIÁ TRỊ
Hệ thập phân	50
Hệ nhị phân	0110010
Hệ bát phân	062
Hệ thập lục phân	32

Trong JavaScript, các số được biểu diễn dưới dạng các dãy bit dài 32 kí tự. Do đó, trong JavaScript, thì:

- 1001 được biểu diễn là 0000..0000(28 số 0)1001.
- 101 được biểu diễn là 000..00(29 số 0)101.

Đối với các **bitwise operator**, các thao tác sẽ được thực hiện trên các bit (tương tự như hệ nhị phân). Cụ thể, đối với các toán tử **& (AND), | (OR), ^ (XOR),** thì chúng sẽ so sánh từng cặp bit của 2 toán hạng (sau khi đã chuyển toàn bộ về hệ nhị phân), và trả về kết quả tương ứng và thêm vào dãy kết quả. Cuối cùng, dãy kết quả này lại được chuyển về hệ thập phân.

Công dụng của chúng được thể hiện như sau:

TÊN TOÁN TỬ	TÁC DỤNG
& (AND)	Trả về 1 nếu cả 2 bit đều là 1.
| (OR)	Trả về 1 nếu có ít nhất một trong 2 bit bằng 1.
^ (XOR)	Trả về 1 nếu như một trong 2 bit là 1, và bit còn lại bằng 0.

Còn đối với các toán tử **~(NOT), << (Zero fill left shift), >> (Signed right shift) và >>> (Zero fill right shift),** thì chúng sẽ tác động trực tiếp lên dãy bit đang được chỉ định và trả về kết quả dưới dạng thập phân.

TÊN TOÁN TỬ	TÁC DỤNG
~ (NOT)	Chuyển các bit trong dãy từ 1 à 0, và ngược lại (từ 0 à 1)
<<	Dịch trái (loại bỏ một vài bit ở đầu dãy, và thêm các số 0 vào cuối dãy sao cho độ dài dãy bằng với ban đầu)
>>>	Dịch phải (tương tự như dịch trái, nhưng loại bỏ các bit bên phải và thêm vào đầu dãy)

Ví dụ:

Javascript:

```
~2
// -3
12 & 10
// 8
5 | 12
// 13
6 ^ 12
// 10
7 >> 2
// 1
8 >>> 2
// 2
12 << 12
// 49152
```

Các toán tử một ngôi

JavaScript cũng hỗ trợ các toán tử một ngôi và nó được sử dụng trong trường hợp mà ta muốn thay đổi giá trị của một biến, với các thao tác cộng thêm một (++) hay trừ đi một (--).

Ví dụ:

Javascript:

```
let t = 1;
// undefined
t++
// 1
t
// 2
++t
// 3
t
// 3
--t
// 2
t
// 2
t--
// 2
t
// 1
```

Cụ thể:

- Toán tử đặt trước biến, sẽ thay đổi giá trị của biến, sau đó mới thực hiện câu lệnh chứa nó
- Toán tử đặt sau biến, sẽ thực hiện câu lệnh chứa nó trước, sau đó mới thay đổi giá trị của biến

Javascript:

```
var t = 5;
// undefined
console.log(t--);
5
// undefined
t
// 4
console.log(++t);
// 5
// undefined
```

Ngoài ra, đối với JavaScript ta có thể đặt dấu + trước một giá trị nào đó để chuyển nó thành số, ví dụ:

Javascript:

```
+'a'
// NaN
+'5'
// 5
+['-1']
// -1
+[1]
// 1
```

Dấu trừ cũng có thể được sử dụng, nhưng nó sẽ trả về giá trị đối so với khi mà ta dùng dấu +.

Javascript:

```
-'a'
// NaN
-['1']
// -1
-[1]
// -1
-'5'
// -5
```

Tránh nhầm lẫn khi tính toán với số thực

Trong quá trình tính toán với các số, có những lúc chúng ta sẽ gặp phải những lỗi như sau:

Javascript:

```
0.1 + 0.2
// 0.30000000000000004
0.2 - 0.15
// 0.05000000000000002
0.3 * 3
// 0.8999999999999999
```

Đây là một lỗi rất cơ bản. Nguyên nhân của nó có thể được hiểu như sau: Để lưu trữ các số, máy tính sẽ chuyển các số này về dạng số nhị phân để tiến hành lưu trữ. Nhưng không phải số nào cũng có thể được biểu diễn một cách chính xác dưới dạng số nhị phân, và điều này tạo ra sai số (Giống các trường hợp trên).

Kết luận

Qua bài này, các bạn đã có kiến thức về kiểu dữ liệu số (Number) trong JavaScript

Ở bài sau, Kteam sẽ hướng dẫn cho các bạn về các thuộc tính của Number cũng như là các phương thức với 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**".