BigInt 数据类型

1. 简介

JavaScript 的数值表示有两大限制：

> - 数值的精度只能到 53 个二进制位（相当于 16 个十进制位），大于这个范围的整数，JavaScript 是无法精确表示。
> - 大于或等于 2 的 1024 次方的数值，JavaScript 无法表示，会返回 `Infinity`。

```
Math.pow(2, 53) === Math.pow(2, 53) + 1 // true, 超过 53 个二进制位的数值，无法保持精度
Math.pow(2, 1024) // Infinity, // 超过 2 的 1024 次方的数值，无法表示
```

`BigInt`（大整数）是第八种数据类型，用来表示整数，没有位数的限制，任何位数的整数都可以精确表示。

```
const a = 2172141653n;
const b = 15346349309n;

a * b // 33334444555566667777n, BigInt 可以保持精度
Number(a) * Number(b) // 33334444555566670000, 普通整数无法保持精度
2n ** 1024n // 179...216n, BigInt 可以表示 2 的 1024 次方的数值
```

为了与 Number 类型区别，BigInt 类型的数据必须添加后缀 `n`。

```
1234 // 普通整数
1234n // BigInt
1n + 2n // 3n, // BigInt 的运算
```

BigInt 同样可以使用各种进制表示，都要加上后缀n。

```
0b1101n // 二进制，输出十进制：13n
0o777n // 八进制，输出十进制：511n
0xFFn // 十六进制，输出十进制：255n
```

BigInt 与普通整数是两种值，它们之间并不相等。

```
42n === 42 // false
```

typeof 运算符对于 BigInt 类型的数据返回 `bigint`。

```
typeof 123n // 'bigint'
```

BigInt 可以使用负号（-），但是不能使用正号（+），因为会与 asm.js 冲突。

```
-42n // 正确
+42n // Cannot convert a BigInt value to a number
```

JavaScript 以前不能计算70的阶乘（即70!），因为超出了可以表示的精度。

```
let p = 1;
for (let i = 1; i <= 70; i++) {
  p *= i;
}
console.log(p); // 1.197857166996989e+100
```

现在支持大整数了，就可以算了，浏览器的开发者工具运行下面代码，就OK。

```
let p = 1n;
for (let i = 1n; i <= 70n; i++) {
  p *= i;
}
console.log(p); // 11978571...00000000n
```

2. BigInt 函数

`BigInt()` 函数，可以用它生成 BigInt 类型的数值。转换规则基本与 `Number()` 一致，将其他类型的值转为 `BigInt`。

```
BigInt(123) // 123n
BigInt('123') // 123n
BigInt(false) // 0n
BigInt(true) // 1n
```

`BigInt()` 函数必须有参数，而且参数必须可以正常转为数值：

```
// 下面的用法都会报错
new BigInt() // TypeError: BigInt is not a constructor
BigInt(undefined) //TypeError: Cannot convert undefined to a BigInt
BigInt(null) // TypeError: Cannot convert null to a BigInt
BigInt('123n') // SyntaxError: Cannot convert 123n to a BigInt
BigInt('abc') // SyntaxError: Cannot convert abc to a BigInt
```

参数如果是小数，也会报错。

```
BigInt(1.5) // RangeError
BigInt('1.5') // SyntaxError
```

BigInt 继承了 Object 对象的两个实例方法。`BigInt.prototype.toString()`，`BigInt.prototype.valueOf()`。

```
// Number.parseInt() 与 BigInt.parseInt() 的对比
Number.parseInt('9007199254740993', 10); // 9007199254740992
BigInt.parseInt('9007199254740993', 10); // 9007199254740993n
```

`Number.parseInt()` 方法返回的结果是不精确的，由于有效数字超出了最大限度。而 `BigInt.parseInt()` 方法正确返回了对应的 `BigInt`。

3. 转换规则

可以使用 `Boolean()`、`Number()` 和 `String()`，将 `BigInt` 可以转为布尔值、数值和字符串类型。

```
Boolean(0n) // false
Boolean(1n) // true
Number(1n)  // 1
String(1n)  // "1"，转为字符串时后缀n会消失。
```

取反运算符（!）也可以将 BigInt 转为布尔值。

```
!0n // true
!1n // false
```

4. 数学运算

数学运算方面，BigInt 类型的 `+`、`-`、`*` 和 `**` 这四个二元运算符，与 Number 类型的行为一致。除法运算 `/` 会舍去小数部分，返回一个整数：

```
9n / 5n // 1n
```

BigInt 不能与普通数值进行混合运算。

```
1n + 1.3 // 报错
```

上面代码报错是因为无论返回的是 BigInt 或 Number，都会导致丢失精度信息。比如 `(2n**53n + 1n) + 0.5` 这个表达式，如果返回 BigInt 类型，`0.5` 这个小数部分会丢失；如果返回 Number 类型，有效精度只能保持 53 位，导致精度下降。

同样的原因，如果一个标准库函数的参数预期是 Number 类型，但是得到的是一个 BigInt，就会报错。

```
Math.sqrt(4n) // Cannot convert a BigInt value to a number
Math.sqrt(Number(4n)) // 2，正确的写法
```

`Math.sqrt` 的参数预期是 Number 类型，如果是 BigInt 就会报错，必须先用 `Number()` 方法转一下类型，才能进行计算。

asm.js 里面，`|0` 跟在一个数值的后面会返回一个 32 位整数。根据不能与 Number 类型混合运算的规则，**BigInt 如果与 `|0` 进行运算会报错。**

```
1n | 0 // 报错
```

5. 其他运算

BigInt 对应的布尔值，与 Number 类型一致，即 `0n` 会转为 false，其他值转为 true。

```
if (0n) {
  console.log('if');
} else {
  console.log('else');
}
// else
```

比较运算符（比如 `>`）和相等运算符（`==`）允许 BigInt 与其他类型的值混合计算，因为这样做不会损失精度：

```
0n < 1 // true
0n < true // true
0n == 0 // true
0n == false // true
0n === 0 // false
```

BigInt 与字符串混合运算时，会先转为字符串，再进行运算。

```
'' + 123n // "123"
```