数据类型-Number

《JavaScript 教程》作者：阮一峰

1 概述

1. JavaScript 中没有整数

   在 JavaScript 内部没有整数，所有数字都是小数，以64位浮点数形式储存。所以，1 与 1.0 在 JS 中是同一个数。

   ```
   1 === 1.0 // true
   ```

   容易造成混淆的是，某些运算只有整数才能完成，此时 JavaScript 会自动把64位浮点数转成32位整数，然后再进行运算。

   由于浮点数不是精确的值，所以涉及小数的比较和运算要特别小心。

   ```
   0.1 + 0.2 === 0.3
   // false

   0.3 / 0.1
   // 2.9999999999999996

   (0.3 - 0.2) === (0.2 - 0.1)
   // false
   ```

2. 数值精度

   安全整数

   Javascript 的数字存储使用了IEEE 754中规定的双精度浮点数数据类型，所以这一数据类型能够安全表示 `-(2^53 - 1)` 到 `2^53 - 1` 之间的数值（包含边界值）。

   这里的 “安全” 意思是能够准确表示整数并且能够正确地进行比较。

   例如，

```
12345678901234567890
// 12345678901234568000
// JS 不能正确表示超出安全整数范围内的整数

Math.pow(2, 53) === Math.pow(2, 53) + 1
// true
// 比较结果显然是错误的
```

Number 对象的 Number.MAX_SAFE_INTEGER 和 Number.MIN_SAFE_INTEGER 属性，保存着最大安全整数 `2^53 - 1` 和最小安全整数 `-(2^53 - 1)`。

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

最大值和最小值

如果一个数大于等于 `2^1024`，那么就会发生“正向溢出”，即 JavaScript 会返回 `Infinity`。

```
Math.pow(2, 1024) // Infinity
```

如果一个数小于等于 `2^-1075` （指数部分最小值-1023，再加上小数部分的52位），那么就会发生为“负向溢出”，即 JavaScript 会返回 0。

```
Math.pow(2, -1075) // 0
```

Number 对象的 MAX_VALUE 和 MIN_VALUE 属性，返回可以表示的最大值 `2^1024` 和最小值 `2^-1075`。

```
Number.MAX_VALUE // 1.7976931348623157e+308
Number.MIN_VALUE // 5e-324
```

2 数值的表示法

1. 使用进制表示
    - 十进制 无前缀 `0`
    - 十六进制 加前缀 `0x` 或 `0X`
    - 二进制 加前缀 `0b` 或 `0B`
    - 八进制 加前缀 `0o` 或 `0O`

```
35     // 35

0xff // 255
```

```
0b11 // 3

0o377 // 255
```

2. 使用科学计数法表示

```
123e3 // 123000
123e-3 // 0.123
-3.1E+12 // -3100000000000
.1e-23 // 1e-24
```

以下两种情况，JavaScript 会自动将数值转为科学计数法表示，其他情况都采用字面形式直接表示。

- 小数点前的数字多于21位。

```
123456789012345678901
// 1.2345678901234568e+21

12345678901234567890
// 12345678901234568000
```

- 小数点后的零多于5个。

```
// 小数点后紧跟5个以上的零，
// 就自动转为科学计数法
0.0000003 // 3e-7

// 否则，就保持原来的字面形式
0.000003 // 0.000003
```

3 特殊数值

1. `+0` 和 `-0`

JavaScript 内部存在两个 `0` ： `+0` 和 `-0` ，区别就是64位浮点数表示法的符号位不同。它们是等价的。

```
-0 === +0 // true
0 === -0 // true
0 === +0 // true
```

几乎所有场合， `+0` 和 `-0` 都会被当作正常的 `0` 。

```
+0 // 0
-0 // 0
(-0).toString() // '0'
(+0).toString() // '0'
```

运算：

`+0` 或 `-0` 作分母时，返回值不相等。

```
1 / +0 // Infinity
1 / -0 // -Infinity
-Infinity === Infinity // false

0 / +0 // NaN
0 / -0 // NaN
```

2. 非数字 `NaN`

含义

`NaN` 是 JavaScript 的特殊值，表示“非数字”（Not a Number）。

主要出现在将字符串解析成数字出错的场合:

```
5 - 'x' // NaN
```

另外，一些数学运算结果会出现 `NaN` 。

```
Math.acos(2) // NaN
Math.log(-1) // NaN
Math.sqrt(-1) // NaN

0/0 //NaN
```

`NaN` 不是独立的数据类型，而是一个特殊数值，它的数据类型依然属于Number:

```
typeof NaN //"number"
```

运算规则

`NaN` 不等于任何值，包括它本身。

```
NaN === NaN // false
```

数组的 `indexOf` 方法内部使用的是严格相等运算符 `===` ，所以无法判断数组中是否存在 `NaN` 。

```
[NaN].indexOf(NaN) // -1
```

`NaN` 在布尔运算时被当作 `false` 。

```
Boolean(NaN) // false
```

`NaN` 与任何数（包括它自己）的运算，得到的都是 `NaN` 。

```
NaN + 32 // NaN
NaN - 32 // NaN
NaN * 32 // NaN
NaN / 32 // NaN
```

3. Infinity 和 -Infinity

`Infinity` 表示正无穷大。

```
Math.pow(2, 1024)
// Infinity

0 / 0
// NaN

1 / 0
// Infinity
```

`-Infinity` 表示负无穷大。

```
Infinity === -Infinity // false

1 / -0 // -Infinity
-1 / -0 // Infinity
```

`Infinity` 大于一切数值（除了 `NaN` ），-`Infinity` 小于一切数值（除了 `NaN` ）。

```
Infinity > NaN // false
Infinity < NaN // false

-Infinity > NaN // false
-Infinity < NaN // false
```

运算规则

`Infinity` 的四则运算，符合无穷的数学计算规则。

```
5 * Infinity // Infinity
5 - Infinity // -Infinity
Infinity / 5 // Infinity
5 / Infinity // 0
```

`0` 与 `Infinity` ：

```
0 * Infinity // NaN
0 / Infinity // 0
Infinity / 0 // Infinity
```

`Infinity` 与 `Infinity`：

```
Infinity + Infinity // Infinity
Infinity * Infinity // Infinity

Infinity - Infinity // NaN
Infinity / Infinity // NaN
```

`Infinity` 与 `null` 计算时，`null` 会转成 `0`：

```
null * Infinity // NaN
null / Infinity // 0
Infinity / null // Infinity
```

`Infinity` 与 `undefined`：

```
undefined + Infinity // NaN
undefined - Infinity // NaN
undefined * Infinity // NaN
undefined / Infinity // NaN
Infinity / undefined // NaN
```

4 与数值相关的方法

1. `parseInt()`

```
parseInt('123') // 123
```

描述： 用于将字符串转为整数。

传入参数： 字符串和可选的进制（2~36）。

返回值： `NaN` 或十进制整数。

++++++++++++++++++

对单个传入参数的处理：

1. 传入参数不是字符串，会先将其转换为字符串。

```
parseInt(1.23) // 1
// 等同于
parseInt('1.23') // 1
```

parseInt 不会解析科学计数法表示的字符串。

```
parseInt(10000000000000000000.5) // 1
// 等同于
parseInt('1e+21') // 1
```

```
parseInt(0.0000008) // 8
// 等同于
parseInt('8e-7') // 8
```

2. 传入字符串有前导空格，空格会被自动去除。

```
parseInt('   81') // 81
```

3. 传入字符串的第一个字符不能转化为数字（后面跟着数字的正负号除外），返回 NaN。

```
parseInt('abc') // NaN
parseInt('.3') // NaN
parseInt('') // NaN
parseInt('+') // NaN
parseInt('+1') // 1
```

否则，返回可以转为数字的部分。

```
parseInt('8a') // 8
parseInt('12**') // 12
parseInt('12.34') // 12
parseInt('15e2') // 15
parseInt('15px') // 15
```

4. 如果字符串以 `0x` 或 `0X` 开头，parseInt 会将其按照十六进制数解析。

```
parseInt('0x10') // 16
```

如果字符串以0开头，将其按照10进制解析。

```
parseInt('011') // 11
```

第二个传入参数

1. parseInt方法还可以接受第二个参数（2 到 36 之间），表示被解析的值的进制，默认为 10。

```
parseInt('1000') // 1000
// 等同于
parseInt('1000', 10) // 1000

parseInt('1000', 2) // 8
parseInt('1000', 6) // 216
parseInt('1000', 8) // 512
parseInt('1000', 16) //4096
```

2. 如果第二个传入参数超出（2~36）的范围则返回 `NaN` 。如果第二个参数是 `0`、`undefined` 或 `null`，则直接忽略。

```
parseInt('10', 37) // NaN
parseInt('10', 1) // NaN
parseInt('10', 0) // 10
parseInt('10', null) // 10
parseInt('10', undefined) // 10
```

3. 如果字符串第一个字符不能用指定进制转换，返回 `NaN`；否则，返回可以解析的字符，直到不能被解析为止。

```
parseInt('1546', 2) // 1
parseInt('546', 2) // NaN
```

4. 第一个参数不是字符串，这可能会导致一些令人意外的结果。

```
parseInt(0x11, 36) // 43
parseInt(0x11, 2) // 1

// 等同于
parseInt(String(0x11), 36)
parseInt(String(0x11), 2)

// 等同于
parseInt('17', 36)
parseInt('17', 2)
```

这种处理方式，对于八进制的前缀0，尤其需要注意。

```
parseInt(011, 2) // NaN

// 而
parseInt('011', 2)

// 等同于
parseInt(String(011), 2)

// 等同于
parseInt(String(9), 2)

parseInt('011', 2) // 3
```

JavaScript 不再允许将带有前缀0的数字视为八进制数，而是要求忽略这个0。但是，为了保证兼容性，大部分浏览器并没有部署这一条规定。

2. parseFloat()

描述： 用于将一个字符串转为浮点数。

传入参数： 字符串。

返回值： `NaN` 或浮点数。

1. 如果参数不是字符串，或者字符串的第一个字符不能转化为浮点数，则返回 `NaN` 。

```
parseFloat([]) // NaN
parseFloat('FF2') // NaN
parseFloat('') // NaN
```

尤其值得注意，parseFloat会将空字符串转为 `NaN` 。

2. 传入的字符串符合科学计数法，则会进行相应的转换。

```
parseFloat('314e-2') // 3.14
parseFloat('0.0314E+2') // 3.14
```

如果字符串包含不能转为浮点数的字符，则不再进行往后转换，返回已经转好的部分。

```
parseFloat('3.14more non-digit characters') // 3.14
```

3. parseFloat方法会自动过滤字符串前导的空格。

```
parseFloat('\t\v\r12.34\n ') // 12.34
```

parseFloat 与 Number 函数的区别

```
        parseFloat(true)  // NaN
        Number(true) // 1

        parseFloat(null) // NaN
        Number(null) // 0

        parseFloat('') // NaN
        Number('') // 0

        parseFloat('123.45#') // 123.45
        Number('123.45#') // NaN
```

3. isNaN()

```
typeof NaN // "number"

isNaN(NaN) // true
isNaN(123) // false
```

描述： 判断一个值是否为 `NaN` 。注意， `NaN` 是 `number` 类型。

传入参数： 数值 ，即 `number` 类型的值。

返回值： `true` 或 `false` 。

1. 如果传入其他值，会自动调用 Number 转换函数将其转换成数值。

`Number()` 转换函数：

```
Number() // 0

Number(0) // 0
Number(39) // 39

Number('') // 0
Number('Dante') // NaN

Number([]) // 0
Number([1,2,3]) // NaN

Number({}) // NaN
Number({a:4, b:5}) // NaN

Number(true) // 1
Number(false) // 0

Number(null) // 0
Number(undefined) // NaN
```

由上述 `Number()` 的转换结果可知， `isNaN` 函数传入非空字符串、非空数组、对象以及 `undefined` 也会返回 `true` 。

2. 比 `isNaN` 更准确的方式

    1. 使用之前，判断一下数据类型

    ```
    function myIsNaN(value) {
            return typeof value === 'number' && isNaN(value);
    }
    ```

    2. 利用 `NaN` 不等于自身的特点

    ```
    NaN !== NaN //true

    function myIsNaN(value) {
            return value !== value;
    }
    ```

4. isFinite()

描述： 用于判断某个值是否为正常的数值。

传入参数： 数值。

返回值： `true` 或 `false` 。

功能： 传入 `Infinity` 、 `-Infinity` 、 `NaN` 或 `undefined` 返回 `false` ；否则返回 `true` 。

除了Infinity、-Infinity、NaN和undefined这四个值会返回false，isFinite对于其他的数值都会返回true。

```
isFinite(Infinity) // false
isFinite(-Infinity) // false
isFinite(NaN) // false
isFinite(undefined) // false

isFinite(null) // true
isFinite(-1) // true
```