

Number 对象

`Number` 对象是数值对应的包装对象，可以作为构造函数使用，也可以作为工具函数使用。

作为构造函数时，它主要用于生成值为数值的对象。

```
let n = new Number(1);  
n; // Number {1}  
typeof n; // "object"
```

作为工具函数时，她可以将任意类型的值转换为数值。

有两个值转换为 `NaN`, 有四个转换为 `0`,

```
Number(0);
```

1. 静态属性

```
// Number.POSITIVE_INFINITY 表示无穷大的数值  
Number.POSITIVE_INFINITY; // Infinity  
  
// Number.NEGATIVE_INFINITY 表示负无穷大的数值  
Number.NEGATIVE_INFINITY; // -Infinity  
  
// Number.NaN 表示非数值  
Number.NaN; // NaN  
isNaN(Number.NaN); // true  
  
// Number.MAX_SAFE_INTEGER 表示能够精确表示的最大整数  
Number.MAX_SAFE_INTEGER; // 9007199254740991  
Math.pow(2, 53) - 1 === Number.MAX_SAFE_INTEGER; // true  
  
// Number.MIN_SAFE_INTEGER; 表示能够精确表示的最小整数  
Number.MIN_SAFE_INTEGER; // -9007199254740991  
Number.MIN_SAFE_INTEGER === -Math.pow(2, 53) + 1; // true  
  
// Number.MAX_VALUE 表示最大的正数  
Number.MAX_VALUE; // 1.7976931348623157e+308  
Number.MAX_VALUE < Infinity; // true  
  
// Number.MIN_VALUE 表示最小的正数  
Number.MIN_VALUE; // 5e-324
```

2. 实例方法

`Number` 对象有 4 个实例方法，这四个方法都将数值转换成指定格式。

2.1. Number.prototype.toString()

`Number` 部署了自己的 `toString()` 方法，用来将一个数值转换成字符串形式。

```
(10).toString(); // "10"
```

`toString()` 还可以接受一个参数，表示输出的进制。

```
(10).toString(2); // "1010"  
(10).toString(8); // "12"  
(10).toString(16); // "a"
```

通过方括号运算符也可以调用 `toString()` 方法。

```
(10)["toString"](2); // "1010"
```

2.2. Number.prototype.toFixed()

`toFixed` 先将一个数转换成指定的小数，然后返回这个小数对应的字符串。

```
(10).toFixed(2); // "10.00"  
(10.123).toFixed(2); // "10.12"
```

`toFixed()` 方法的参数为小数位数，有效范围为 0 到 100，超出这个范围将抛出 `RangeError` 错误。

```
(100.0).toFixed(120); // Uncaught RangeError: toFixed() digits argument must be  
between 0 and 100
```

由于浮点数的原因，小数 5 的四舍五入是不确定的，使用的时候必须小心。

```
(10.055).toFixed(2); // "10.05"  
(10.056).toFixed(2); // "10.06"  
(10.005).toFixed(2); // "10.01"
```

2.3. Number.prototype.toExponential()

`toExponential()` 方法将一个转换为科学计数法。

```
(10).toExponential(2); // "1.00e+1"  
(123.4567).toExponential(2); // "1.23e+2"  
(1234).toExponential(); // "1.234e+3"
```

`toExponential` 方法的参数是小数点后有效数字的位数，范围为 0 到 100，超出这个范围，会抛出一个 `RangeError` 错误。

```
(345.234).toExponential(120); // Uncaught RangeError: toExponential() argument  
must be between 0 and 100  
(345.234).toExponential(10); // "3.4523400000e+2"
```

2.4. Number.prototype.toPrecision()

`toPrecision` 用于将一个数转换为有效的数字。

```
(12.34).toPrecision(1); // "1e+1"  
(12.34).toPrecision(2); // "12"  
(12.34).toPrecision(3); // "12.3"  
(12.34).toPrecision(4); // "12.34"  
(12.34).toPrecision(5); // "12.340"
```

该方法的参数为有效数字的位数，范围是 1 到 100，超出这个范围会抛出 `RangeError` 错误。

```
(345.234).toExponential(0); // Uncaught RangeError: toPrecision() argument must be  
between 1 and 100
```

该方法用于四舍五入时不太可靠，跟浮点数不是精确储存有关。

```
(12.35).toPrecision(3); // "12.3"  
(12.25).toPrecision(3); // "12.3"  
(12.15).toPrecision(3); // "12.2"  
(12.45).toPrecision(3); // "12.4"
```

2.5. Number.prototype.toLocaleString()

`toLocaleString()` 方法接受一个地区码作为参数，返回一个字符串，表示当前数字在该地区的当地书写形式。

该方法还可以接受第二个参数配置对象，用来定制指定用途的返回字符串。该对象的 `style` 属性指定输出样式，默认值是 `decimal`，表示输出十进制形式。如果值为 `percent`，表示输出百分数。

```
(123).toLocaleString("zh-Hans-CN", { style: "percent" });  
// "12,300%"
```

如果 `style` 属性的值为 `currency`，则可以搭配 `currency` 属性，输出指定格式的货币字符串形式。

```
(123).toLocaleString("zh-Hans-CN", { style: "currency", currency: "CNY" });  
// "¥123.00"  
  
(123).toLocaleString("de-DE", { style: "currency", currency: "EUR" });  
// "123,00 €"  
  
(123).toLocaleString("en-US", { style: "currency", currency: "USD" });  
// "$123.00"
```

3.3. 自定义方法

与其他对象一样，`Number.prototype` 对象上面可以自定义方法，被 `Number` 的实例继承。

```
Number.prototype.add = function (x) {  
  return this + x;  
};  
  
(8)["add"](2); // 10  
  
Number.prototype.double = function () {  
  return this + this;  
};  
(8)["double"](); // 16  
(10).double(); // 20
```

注意，数值的自定义方法，只能定义在它的原型对象 `Number.prototype` 上面，数值本身是无法自定义属性的。

```
let n = 1;  
n.x = 1;  
n.x; // undefined
```