Number 对象

概述

`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，它用于生成值为数值的对象。

```javascript
var n = new Number(1);
typeof n // "object"
```

上面代码中，`Number`对象作为构造函数使用，返回一个值为`1`的对象。

作为工具函数时，它可以将任何类型的值转为数值。

```javascript
Number(true) // 1
```

上面代码将布尔值`true`转为数值`1`。`Number`作为工具函数的用法，详见《数据类型转换》一章。

静态属性

`Number`对象拥有以下一些静态属性（即直接定义在`Number`对象上的属性，而不是定义在实例上的属性）。

- `Number.POSITIVE_INFINITY`：正的无限，指向`Infinity`。
- `Number.NEGATIVE_INFINITY`：负的无限，指向`-Infinity`。
- `Number.NaN`：表示非数值，指向`NaN`。
- `Number.MIN_VALUE`：表示最小的正数（即最接近0的正数，在64位浮点数体系中为`5e-324`），相应的，最接近0的负数为`-Number.MIN_VALUE`。
- `Number.MAX_SAFE_INTEGER`：表示能够精确表示的最大整数，即`9007199254740991`。
- `Number.MIN_SAFE_INTEGER`：表示能够精确表示的最小整数，即`-9007199254740991`。

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

Number.MAX_VALUE
// 1.7976931348623157e+308
Number.MAX_VALUE < Infinity
// true
```

```
Number.MIN_VALUE
// 5e-324
Number.MIN_VALUE > 0
// true

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

实例方法

`Number`对象有4个实例方法，都跟将数值转换成指定格式有关。

Number.prototype.toString()

`Number`对象部署了自己的`toString`方法，用来将一个数值转为字符串形式。

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

`toString`方法可以接受一个参数，表示输出的进制。如果省略这个参数，默认将数值先转为十进制，再输出字符串；否则，就根据参数指定的进制，将一个数字转化成某个进制的字符串。

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

上面代码中，`10`一定要放在括号里，这样表明后面的点表示调用对象属性。如果不加括号，这个点会被 JavaScript 引擎解释成小数点，从而报错。

```javascript
10.toString(2)
// SyntaxError: Unexpected token ILLEGAL
```

只要能够让 JavaScript 引擎不混淆小数点和对象的点运算符，各种写法都能用。除了为`10`加上括号，还可以在`10`后面加两个点，JavaScript 会把第一个点理解成小数点（即`10.0`），把第二个点理解成调用对象属性，从而得到正确结果。

```javascript
10..toString(2)
// "1010"

// 其他方法还包括
10 .toString(2) // "1010"
10.0.toString(2) // "1010"
```

这实际上意味着，可以直接对一个小数使用`toString`方法。

```javascript
10.5.toString() // "10.5"
10.5.toString(2) // "1010.1"
10.5.toString(8) // "12.4"
10.5.toString(16) // "a.8"
```

通过方括号运算符也可以调用`toString`方法。

```javascript
10['toString'](2) // "1010"
```

`toString`方法只能将十进制的数，转为其他进制的字符串。如果要将其他进制的数，转回十进制，需要使用`parseInt`方法。

Number.prototype.toFixed()

`toFixed()`方法先将一个数转为指定位数的小数，然后返回这个小数对应的字符串。

```javascript
(10).toFixed(2) // "10.00"
10.005.toFixed(2) // "10.01"
```

上面代码中，`10`和`10.005`先转成2位小数，然后转成字符串。其中`10`必须放在括号里，否则后面的点会被处理成小数点。

`toFixed()`方法的参数为小数位数，有效范围为0到20，超出这个范围将抛出 RangeError 错误。

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。

```javascript
(10.055).toFixed(2) // 10.05
(10.005).toFixed(2) // 10.01
```

Number.prototype.toExponential()

`toExponential`方法用于将一个数转为科学计数法形式。

```javascript
(10).toExponential()  // "1e+1"
(10).toExponential(1) // "1.0e+1"
(10).toExponential(2) // "1.00e+1"

(1234).toExponential()  // "1.234e+3"
```

```
(1234).toExponential(1) // "1.2e+3"
(1234).toExponential(2) // "1.23e+3"
```

`toExponential`方法的参数是小数点后有效数字的位数，范围为0到20，超出这个范围，会抛出一个 RangeError 错误。

Number.prototype.toPrecision()

`toPrecision`方法用于将一个数转为指定位数的有效数字。

```javascript
(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"
```

`toPrecision`方法的参数为有效数字的位数，范围是1到21，超出这个范围会抛出 RangeError 错误。

`toPrecision`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。

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

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。

```javascript
Number.prototype.add = function (x) {
  return this + x;
};

8['add'](2) // 10
```

上面代码为`Number`对象实例定义了一个`add`方法。在数值上调用某个方法，数值会自动转为`Number`的实例对象，所以就可以调用`add`方法了。由于`add`方法返回的还是数值，所以可以链式运算。

```javascript
Number.prototype.subtract = function (x) {
  return this - x;
```

```
};

(8).add(2).subtract(4)
// 6
```

上面代码在`Number`对象的实例上部署了`subtract`方法，它可以与`add`方法链式调用。

我们还可以部署更复杂的方法。

```javascript
Number.prototype.iterate = function () {
  var result = [];
  for (var i = 0; i <= this; i++) {
    result.push(i);
  }
  return result;
};

(8).iterate()
// [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

上面代码在`Number`对象的原型上部署了`iterate`方法，将一个数值自动遍历为一个数组。

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。

```javascript
var n = 1;
n.x = 1;
n.x // undefined
```

上面代码中，`n`是一个原始类型的数值。直接在它上面新增一个属性`x`，不会报错，但毫无作用，总是返回`undefined`。这是因为一旦被调用属性，`n`就自动转为`Number`的实例对象，调用结束后，该对象自动销毁。所以，下一次调用`n`的属性时，实际取到的是另一个对象，属性`x`当然就读不出来。