JavaScript 的类型系统

作者：lingyundu
原文链接：https://ld246.com/article/1603018708892
来源网站：链滴

当前 ECMAScript 标准定义了8种数据类型，包括7个原始（primary）类型，还有一个是对象类型。

类型	含义	说明
Undefined	未定义	未声明的变，或者声明过但未赋值的变量的值。该类型只有一个值：undefined。
Null	空值	该类型只有一个值：null。
Number	数值	表示整数和小数。
Bigint	整数	ES2020 引入的一个的数据类型，可以表示任意精度格式的整数。
String	字符串	不能直接读取或改字符串中的单个字符。
Boolean	布尔值	true/false
Symbol	符号	ES2015（ES6）入了一个新的数据类型，表示独一无二的值。
Object	对象	一组属性（propert）的无序集合。

Null和Undefined类型

据说在最初设计 JavaScript 时，像 Java 一样，只设置了 null 表示“无”。根据 C 语言的传统，null 可以自动转为 0。

但是，JavaScript 的设计者 Brendan Eich，觉得这样做还不够。首先，Brendan Eich 觉得表示“无的值最好不是对象。其次，那时的 JavaScript 不包括错误处理机制，Brendan Eich 觉得，如果 null 动转为0，很不容易发现错误。

```
Number(null) // 0
5 + null // 5
```

因此，他又设计了一个undefined。区别是这样的：null是一个表示“空”的对象，转为数值时为0；ndefined是一个表示"此处无定义"的原始值，转为数值时为NaN。

```
Number(undefined) // NaN
5 + undefined // NaN
```

Number类型

存储结构

在 JavaScript 内部，所有数字都是以64位浮点数形式储存，即使整数也是如此。

因此1与1.0是相同的，是同一个数。

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

字面量

数值字面量支持十进制、八进制、十六进制以及科学计数法。

```
// 十进制：无前缀，或者有前缀0但是后面有8或9
var num1 = 21;
var num2 = 0789; // 这是十进制，因为这里有8和9

// 八进制：
// 使用前缀0表示（ES5的严格模式不再允许使用该写法）。
var num3 = 0123;
// 使用前缀0o或0O表示（ES6引入的新写法）。
var num4 = 0o123;

// 十六进制：使用前缀0x或0X表示。
var num5 = 0xff;

// 科学计数法
var num6 = 25e2; // 2500
var num7 = 25e-2; // 0.25

// 二进制：使用前缀0b或0B表示（ES6引入的新写法）。
var num8 = 0b111110111; // 503
```

前缀0表示八进制，处理时很容易造成混乱。ES5 的严格模式和 ES6，已经废除了这种表示法。

特殊数值

NaN

NaN表示“非数字”（not a number），主要用在将字符串解析成数值出错等场合。

```
5 - 'x' // NaN
0 / 0 // NaN
```

Infinity

Infinity表示“无穷”，用来表示两种场景。一种是一个正的数值太大，或一个负的数值太小，无法表；另一种是非0数值除以0，得到Infinity。

Infinity有正负之分，Infinity表示正的无穷，-Infinity表示负的无穷。

Bigint类型

JavaScript 所有数字都保存成 64 位浮点数，这给数值的表示带来了两大限制。一是数值的精度只能到 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 是无法精确表示的，这得 JavaScript 不适合进行科学和金融方面的精确计算。二是大于或等于2的1024次方的数值，JavaScript 无法表示，会返回Infinity。

```
// 超过 53 个二进制位的数值，无法保持精度
Math.pow(2, 53) === Math.pow(2, 53) + 1 // true

// 超过 2 的 1024 次方的数值，无法表示
Math.pow(2, 1024) // Infinity
```

ES2020 引入了一种新的数据类型 BigInt（大整数），来解决这个问题，这是 ECMAScript 的第八种据类型。BigInt 只用来表示整数，没有位数的限制，任何位数的整数都可以精确表示。

为了与 Number 类型区别，BigInt 类型的数据必须添加后缀n。

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

// BigInt 可以保持精度
a * b // 33334444555566667777n
```

String类型

字符串是包含零个或多个字符的序列。

字面量

```
// 字符串需放在单引号或者双引号之中。
var str1 = 'abc';
var str2 = "abc";

// 单引号字符串内部，可以使用双引号。双引号字符串内部可以使用单引号。
var str1 = 'key = "value"'
var str2 = "It's a long journey"

// 在单（双）引号字符串内部使用单（双）引号，需用反斜杠来进行转义。
var str1 = 'Did she say \'Hello\'?'
var str2 = "Did she say \"Hello\"?"

// 当字符串太长时，可以使用反斜杠进行折行，效果与写在同一个行完全一样。
// 需注意的是：
//    1. 从第二行开始，不要添加多余的空格，否则这些空格也会成为字符串的一部分。
//    2. 反斜杠后边必须是换行符，不能有其他字符（比如空格）。
var longString = 'Long \
long \
long \
string';
```

模板字符串

使用+号可以将多个字符串拼接起来。但是如果有很多变量时，使用+就比较麻烦了。

因此，ES6 引入了模板字符串。模板字符串（template string）是增强版的字符串，用反引号（`标识，可以使用${}嵌入变量或表达式。

```
// 使用拼接符
$('#result').append(
  'There are <b>' + basket.count + '</b> ' +
  'items in your basket, ' +
  '<em>' + basket.onSale +
  '</em> are on sale!'
);
// 使用模板字符串
$('#result').append(`
  There are <b>${basket.count}</b> items
   in your basket, <em>${basket.onSale}</em>
  are on sale!
`);
```

转义

对于那些有特殊含义的字符（比如单引号）或者没有字符实体（比如换行符）的字符，可以采用下面种方法来表示：

- 转义符（反斜杠）+ 普通字符
- 转义符 + Unicode码点

特殊字符	转义字符+普通字符	义符+Unicode码点（十进制）
null	\0	\u0000
后退键	\b	\u0008
换页符	\f	\u000C
换行符	\n	\u000A
回车键	\r	\u000D
制表符	\t	\u0009
垂直制表符	\v	\u000B
单引号	\'	\u0027
双引号	\"	\u0022
反斜杠	\\	\u005C

转义符 + Unicode码点的形式同样可以表示其他普通字符。

另外，Unicode码点也可以使用三位八进制（000到377）或者两位十六进制（00到FF）表示。但这种方法只能表示256个字符。

字符串与数组

字符串可以被视为字符数组，因此可以使用数组的方括号运算符，用来返回某个位置的字符（位置编从0开始）。但是，无法改变或者删除字符串内的单个字符。

字符串长度

length属性返回字符串的长度，该属性也是无法改变的。

```
var s = 'hello';
s.length // 5

s.length = 3;
s.length // 5
```

字符集

JavaScript 使用 Unicode 字符集。每个字符在 JavaScript 内部都是以16位（即2个字节）的 UTF-16 格式储存。

Symbol类型

ES5 的对象属性名都是字符串，这容易造成属性名的冲突。比如，你使用了一个他人提供的对象，但想为这个对象添加新的方法，新方法的名字就有可能与现有方法产生冲突。为了从根本上防止属性名冲突，ES6 引入了 Symbol 类型。

Symbol 类型是一种新的原始数据类型，它表示独一无二的值。Symbol 值可以用作对象的属性名

Symbol 值通过Symbol函数生成。该值不是对象，因此Symbol函数前不能使用new命令，否则会报。

Symbol函数可以接受一个字符串作为参数，表示对 Symbol 实例的描述，主要是为了在控制台显示或者转为字符串时，比较容易区分。该参数可以是原始类型，也可以是对象，并且不会对 Symbol 值生影响。

```
let s1 = Symbol('foo');
let s2 = Symbol('bar');

s1 // Symbol(foo)
s2 // Symbol(bar)

s1.toString() // "Symbol(foo)"
s2.toString() // "Symbol(bar)"

let s3 = Symbol('foo');
s1 === s3 // false
```

Symbol 值可以显式转为字符串或者布尔值：

```
let sym = Symbol('My symbol');

String(sym) // 'Symbol(My symbol)'
sym.toString() // 'Symbol(My symbol)'

let sym = Symbol();
Boolean(sym) // true
!sym  // false
```

Symbol 值作为对象属性名时，不能用点运算符，只能使用方括号。

```
let mySymbol = Symbol();

// 第一种写法
let a = {};
a[mySymbol] = 'Hello!';

// 第二种写法
let a = {
  [mySymbol]: 'Hello!'
};

// 第三种写法
let a = {};
Object.defineProperty(a, mySymbol, { value: 'Hello!' });

// 以上写法都得到同样结果
a[mySymbol] // "Hello!"
```

Object类型

简单的说，在 JavaScript 中对象就是一组属性（property）的集合。

属性包含属性名（key）和属性值（value）以及属性的一些特征（比如：属性是否可被修改、属性是可被遍历等）组成。

对象字面量

对象的字面量由一组键值对和{}构成，其中键和值之间由冒号（:）分隔。

对象的所有键名都是字符串（ES6 又引入了 Symbol 值也可以作为键名），所以加不加引号都可以。果键名是数值，会被自动转为字符串。

```
// 这是一个空的对象
var obj = {};

// 创建一个Student对象
var student = {
  name: 'Tom',
  age: 12
}
```

构造函数

JavaScript 语言使用构造函数（constructor）作为对象的模板，通过new命令执行构造函数来生成例对象。比如：Object就是一个构造函数。

```
// 创建一个空的对象
var obj = new Object();

// 定义一个Student构造函数
var Student = function(){
  this.name = 'Tom',
  this.age = 12
};
// 创建一个Student对象实例
var s = new Student();
```

Class（类）

ES6 引入了 Class（类）这个概念，作为对象的模板，通过class关键字，可以定义类。

```
// 定义一个Student类
class Student {
  // 构造方法，生成对象实例时被调用
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  // 定义“类”的方法时，前面不需要添加function关键字
  toString() {
    return '(name: ' + this.name + ', age: ' + this.age + ')';
  }
}
```

```
// 创建一个Student对象实例
var s = new Student('Tom', 12);
```

对象的分类

ECMAScript 规范明确定义了各种对象的类别，包括：

- 常规对象（ordinary object）拥有 JavaScript 对象所有的默认行为。
- 特异对象（exotic object）的某些内部行为和默认的有所差异。
- 标准对象（standard object）是 ECMAScript 6 中定义的对象，例如 Array, Date 等，它们既能是常规也可能是特异对象。
- 内置对象（built-in object）指 JavaScript 执行环境开始运行时已存在的对象。标准对象均为内置象。

数据类型判断

使用typeof运算符可以判断一个值的数据类型。

```
typeof 123; // 'number'
typeof NaN; // 'number'
typeof 'str'; // 'string'
typeof true; // 'boolean'
typeof undefined; // 'undefined'
typeof Math.abs; // 'function'，函数也是对象
typeof null; // 'object'
typeof []; // 'object'
typeof {}; // 'object'
```

null的类型是object，这是由于历史原因造成的。1995年的 JavaScript 语言第一版，只设计了五种据类型（对象、整数、浮点数、字符串和布尔值），没考虑null，只把它当作object的一种特殊值。来null独立出来，作为一种单独的数据类型，为了兼容以前的代码，typeof null返回object就没法改了。

相关资料

《JavaScript语言精髓与编程实践》

JavaScript全栈教程

JavaScript 教程

ES6 入门教程

Understanding ECMAScript 6(简体中文版)

ECMAScript® 2020 Language Specification