语言基础

数据类型

ES6有6中简单数据类型（原始类型）

- Undefined
- Null
- Boolean
- Number
- String
- Symbol ES6
- BigInt ES11

还有一种复杂数据类型Object（对象）

typeof

ES类型系统是松散的，所以需要一种手段来确定变量的数据类型
typeof是操作符，返回值如下

- undefined 表示值未定义
- boolean 表示值位布尔值
- string 表示值为字符串
- number 表示值为数字
- object 表示置为对象（而不是函数）或null或者数组
- function 表示值为函数
- symbol 表示值为符号

typeof不能区分对象/数组/Null
可以使用Object.prototype.toString.call(value)

Undefined

Undefined 类型只有一个值，就是特殊值 undefined
声明变量没有初始化就是undefined
一般不会去显式的赋值undefine，因为ES3之前是没有undefined,出现的作用就是为了明确Null与未初始化变量

undefined是声明未初始化，与未声明不同

```
let message; // 这个变量被声明了，只是值为 undefined
// 确保没有声明过这个变量
// let age
```

```
console.log(message); // "undefined"
console.log(age); // 报错
```

但是typeof未声明返回结果还是undefined，用来条件声明

Null

Null 类型同样只有一个值，即特殊值 null
Null表示一个空对象指针，所以typeof null返回值是object

Boolean

数据类型	转换为 true 的值	转换为 false 的值
Boolean	true	false
String	非空字符串	""（空字符串）
Number	非零数值（包括无穷值）	0、NaN（参见后面的相关内容）
Object	任意对象	null
Undefined	N/A（不存在）	undefined

转成Boolean后为false的值 false,"",0,NaN,null,undefined

Number

Number用了IEEE 754格式表示整数和浮点值（双精度值）
不同数值类型有不同的字面量

- 10进制 直接写出数字
- 8进制 0开头，后面接0-7。若超出则会忽略前缀0
- 16进制 0x前缀，16进制数字

由于JS保存数值的方式，存在+0与-0，一般在所有情况下都被认为是等同，如果一定要区分可以使用 Object.is()

浮点值

```
let floatNum1 = 1.1;
let floatNum2 = 0.1;
let floatNum3 = .1; // 有效，但不推荐
let floatNum1 = 1.; // 小数点后面没有数字，当成整数 1 处理
let floatNum2 = 10.0; // 小数点后面是零，当成整数 10 处理
let floatNum = 3.125e7; // 等于 31250000
```

0.1+0.2!=0.3
可以用以下代码来判断浮点数是否相等

```
Math.abs(a-b)<0.0000000001
```

- Number.MIN_VALUE = 5e-324

- Number.MAX_VALUE = (Math.pow(2, 53) - 1) * Math.pow(2, 971) = 1.7976931348623157e+308
- JS的精度最大整数为2^53即9007199254740992，当 x 大于等于 9007199254740992时，x === x + 1。
- 小于Number.MIN_VALUE的数为0
- IEEE754规定只有大于1.797 693 134 862 315 8e+308才会被round到Infinity， 所以Number.MAX_VALUE+1不是Infinity

NaN
不是数值，Not a Number

- 其他语言被0除都是报错，ES被0除会返回NaN
- NaN==NaN; //false
- isNaN(value)测试值能否转成数值，不能为true，能则返回false
- isNaN无法严格区分字符串与NaN，所以后添加了Number.isNaN(),只有NaN返回true

数值转换

1. Number()
    - Boolean，true->1;false->0
    - Number，直接返回
    - Null，0
    - Undefined，NaN
    - 字符串
        - 为数值字符，转为10进制
        - 为有效浮点值，转为浮点
        - 为有效16进制，转为10进制整数
        - 空字符串，0
        - 如果包含上述情况之外的字符，NaN
    - 对象，调用valueOf()方法，并按上述规则转换返回值。如果是NaN则用toString，再用字符串规则转换
2. parseInt()
   一般使用parseInt()，来进行字符串转数值。parseInt()更专注是否包含数值
   第一个非空字符，如果不是+-数值符号，返回NaN
   为了避免解析出错，建议传第二个参数

```
let num1 = parseInt("1234blue"); // 1234
let num2 = parseInt(""); // NaN
let num3 = parseInt("0xA"); // 10，解释为十六进制整数
let num4 = parseInt(22.5); // 22
let num5 = parseInt("70"); // 70，解释为十进制值
let num6 = parseInt("0xf"); // 15，解释为十六进制整数
```

3. parseFloat()
   与parseInt类似，解析到字符串末尾或者第一个无效浮点数值为止
   区别：只能解析10进制，所以忽略开头0（八进制），16进制返回0

```
let num1 = parseFloat("1234blue"); // 1234，按整数解析
let num2 = parseFloat("0xA"); // 0
let num3 = parseFloat("22.5"); // 22.5
```

```
let num4 = parseFloat("22.34.5"); // 22.34
let num5 = parseFloat("0908.5"); // 908.5
let num6 = parseFloat("3.125e7"); // 31250000
```

String

String（字符串）数据类型表示零或多个 16 位 Unicode 字符序列

字面量

字面量	含义
\n	换行
\t	制表
\b	退格
\r	回车
\f	换页
\\	反斜杠（\）
\'	单引号（'），在字符串以单引号标示时使用，例如'He said, \'hey.\''

图灵社区会员 aSINKz(1561821892@qq.com) 专享 尊重版权

（续）

字面量	含义
\"	双引号（"），在字符串以双引号标示时使用，例如"He said, \"hey.\""
\`	反引号（`），在字符串以反引号标示时使用，例如`He said, \`hey.\``
\xnn	以十六进制编码 *nn* 表示的字符（其中 *n* 是十六进制数字 0~F），例如\x41 等于"A"
\unnnn	以十六进制编码 *nnnn* 表示的 Unicode 字符（其中 *n* 是十六进制数字 0~F），例如\u03a3 等于希腊字符"Σ"

特点：不可变（immutable）。要修改只能赋值（必须先销毁原始的字符串，然后将包含新值的另一个字符串保存到该变量）

转换为字符串

- toString(),字符串也有toString方法，返回自身的一个副本
- null,undefined没有toString(),可以用String()
- 数值调用toString()可以传参，表示进制

模板字面量
ES6新增，单引号。

- 跨行定义字符串

- 字符串插值 `${value}`
  所有插入的值都会调用toString(),也可以插入自己
- 标签函数

```
let a = 6;
let b = 9;
// ...剩余操作符 rest operator
function simpleTag(strings, ...expressions) {
 console.log(strings);
 for(const expression of expressions) {
 console.log(expression);
 }
 return 'foobar';
}
let taggedResult = simpleTag`${ a } + ${ b } = ${ a + b }`;
// ["", " + ", " = ", ""]
// 6
// 9
// 15
console.log(taggedResult); // "foobar"
```

- 原始字符串
  用模板字面量也可以直接获取原始的模板字面量内容（如换行符或 Unicode字符），而不是被转换后的字符表示。为此，可以使用默认的 String.raw 标签函数

```
// Unicode 示例
// \u00A9 是版权符号
console.log(`\u00A9`); // ©
console.log(String.raw`\u00A9`); // \u00A9
// 换行符示例
console.log(`first line\nsecond line`);
// first line
// second line
console.log(String.raw`first line\nsecond line`); // "first line\nsecond line"
// 对实际的换行符来说是不行的
// 它们不会被转换成转义序列的形式
```

Symbol

ES6新加的数据类型。
确保对象属性使用唯一标识符，不会发生属性冲突
（我个人理解每一个symbol实例就是从内存中拿一个地址，唯一标示）

用法

- 符号需要使用 Symbol()函数初始化
- 调用Symbol()可以传一个字符串，这是对符号的描述，与符号的定义标识完全无关
- Symbol()函数不能与new一起作为构造函数使用
- 主要是用来当对象的属性名
- 如果不同部分需要共享和重用符号实例，可以用字符串为key,全局符号注册表中创建。

```
let fooGlobalSymbol = Symbol.for('foo'); // 创建新符号
let otherFooGlobalSymbol = Symbol.for('foo'); // 重用已有符号
console.log(fooGlobalSymbol === otherFooGlobalSymbol); // true

let o = {
    [fooGlobalSymbol]:'123'
}
```

Symbol值作为属性名特性：

- 共有属性，可在类外部访问
- 无法遍历 enumerable:false，即不能for...in/of，Object.Keys()。
  用Object.getOwnPropertySymbols() 和 Reflect.ownKeys() 取到
  Object.getOwnPropertyDescriptors()会返回同时包含常规和符号属性描述符的对象

```
let s1 = Symbol('foo'),
 s2 = Symbol('bar');
let o = {
 [s1]: 'foo val',
 [s2]: 'bar val',
 baz: 'baz val',
 qux: 'qux val'
};
console.log(Object.getOwnPropertySymbols(o));
// [Symbol(foo), Symbol(bar)]
console.log(Object.getOwnPropertyNames(o));
// ["baz", "qux"]
console.log(Object.getOwnPropertyDescriptors(o));
// {baz: {...}, qux: {...}, Symbol(foo): {...}, Symbol(bar): {...}}
console.log(Reflect.ownKeys(o));
// ["baz", "qux", Symbol(foo), Symbol(bar)]
```

TODO:

- 常用内置符号
- Symbol.asyncIterator
- Symbol.hasInstance
- Symbol.isConcatSpreadable
- Symbol.iterator
- Symbol.match
- Symbol.replace
- Symbol.search
- Symbol.species
- Symbol.split
- Symbol.toPrimitive
- Symbol.toStringTag
- Symbol.unscopables

Object

```
let o = new Object();
let o = new Object; // 合法，但不推荐
```

每个Object拥有的属性和方法

- constructor:用于创建当前对象的函数
- hasOwnProperty(propertyName)：是否是自己的属性，不查原型，不查方法
- isPrototypeOf(object)：判断当前对象是否为另一个对象的原型
- propertyIsEnumerable(propertyName)：属性是否可枚举
- toLocaleString()::返回对象的字符串表示，该字符串反映对象所在的本地化执行环境
- toString():返回对象的字符串表示
- valueOf():返回对象对应的字符串、数值或布尔值表，通常与 toString()的返回值相同