链滴

JavaScrpit 概念快速概览

作者：limanting

原文链接：https://ld246.com/article/1629819466438

来源网站：链滴

快速入门

javaScript 每个语句以 ;结束，语句块用 {...}。

注释

以 //开头直到行末的字符被视为行注释，块注释是用 /*...*/把多行字符包裹起来，把一大“块”视为一个注释。

大小写

JavaScript严格区分大小写，如果弄错了大小写，程序将报错或者运行不正常。

类型

1. **number**（不区分整数和浮点数）1.23e3 =1230 ； NaN 无法计算结果时（not a Num）； Infinity 无限大；

```
NaN === NaN; // false 唯一能判断NaN的方法是通过isNaN()函数：
1 / 3 === (1 - 2 / 3); // false
Math.abs(1 / 3 - (1 - 2 / 3)) < 0.0000001; // true
这不是JavaScript的设计缺陷。浮点数在运算过程中会产生误差，因为计算机无法精确表示无限循环数。要比较两个浮点数是否相等，只能计算它们之差的绝对值，看是否小于某个阈值
```

2. **string**

```
`这是一个
多行
字符串`;
// ES6新增了一种模板字符串
var message = `你好, ${name}, 你今年${age}岁了!`;
对字符串的某个索引赋值，不会有任何错误，但是，也没有任何效果.
var s = 'Test';
s[0] = 'X';
alert(s); // s仍然为'Test'
```

3. **boolean**

4. **null**和**undefined**。 用null表示一个空的值，而undefined表示值未定义。大多数情况下，我们都该用null。undefined仅仅在判断函数参数是否传递的情况下有用。

5. **数组**。直接给Array的length赋一个新的值会导致Array大小的变化，如果通过索引赋值时，索引超了范围，同样会引起Array大小的变化

6. **对象**。一组由键-值组成的无序集合。不是一个有效的变量，就需要用''括起来。访问这个属性也无使用.操作符，必须用['xxx']来访问。检测xiaoming是否拥有某一属性，可以用in操作符，这个属性不定是xiaoming的，它可能是xiaoming继承得到的，要判断一个属性是否是xiaoming自身拥有的，而是继承得到的，可以用hasOwnProperty()方法。对象有个小问题，就是键必须是字符串。

7. **变量**。变量名是大小写英文、数字、$和_的组合，且不能用数字开头。变量名也不能是JavaScript关键字，如if、while等。变量本身类型不固定的语言称之为动态语言。JavaScript的函数可以嵌套，时，内部函数可以访问外部函数定义的变量，反过来则不行。

strict模式

非严格模式：如果一个变量没有通过 var申明就被使用，那么该变量就自动被申明为全局变量。在同个页面的不同的JavaScript文件中，如果都不用 var申明，恰好都使用了变量 i，将造成变量 i互相影，产生难以调试的错误结果。

使用 var申明的变量则不是全局变量，它的范围被限制在该变量被申明的函数体内（函数的概念将稍讲解），同名变量在不同的函数体内互不冲突

ECMA在后续规范中推出了strict模式

启用strict模式的方法是在JavaScript代码的第一行写上：

```
'use strict';
```

在strict模式下运行的JavaScript代码，强制通过 var申明变量，未使用 var申明变量就使用的，将导运行错误。

Map

Map是一组键值对的结构，具有极快的查找速度。

```
var m = new Map([['Michael', 95], ['Bob', 75], ['Tracy', 85]]); // 初始化Map需要一个二维数组，者直接初始化一个空Map
m.get('Michael'); // 95

var m = new Map(); // 空Map
m.set('Adam', 67); // 添加新的key-value
m.set('Bob', 59);
m.has('Adam'); // 是否存在key 'Adam': true
m.get('Adam'); // 67
m.delete('Adam'); // 删除key 'Adam'
m.get('Adam'); // undefined
```

由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉：

Set

是一组key的集合，但不存储value。由于key不能重复，所以，在 Set中，没有重复的key。重复元素 Set中自动被过滤。

```
创建一个Set，需要提供一个Array作为输入，或者直接创建一个空Set：
var s1 = new Set(); // 空Set
var s2 = new Set([1, 2, 3]); // 含1, 2, 3
s; // Set {1, 2, 3, 4}
s.add(4);
s.delete(3);
```

iterable

具有 iterable类型的集合可以通过新的 for ... of循环来遍历。

ES6标准引入了新的 iterable类型，Array、Map和 Set都属于 iterable类型。

for ... in遍历的实际上是对象的属性名称。 一个 Array数组实际上也是一个对象，它的每个元素的索被视为一个属性。 （但 Array的 length属性却不包括在内）

for ... of循环则完全修复了这些问题，它只循环集合本身的元素。

直接使用 iterable内置的 forEach方法，它接收一个函数，每次迭代就自动回调该函数。（但是forEac需要注意里面不能有异步操作）。

Set与 Array类似，但 Set没有索引，因此回调函数的前两个参数都是元素本身：

```
var s = new Set(['A', 'B', 'C']);
s.forEach(function (element, sameElement, set) {
    console.log(element);
});
```

Map的回调函数参数依次为 value、key和 map本身：

```
var m = new Map([[1, 'x'], [2, 'y'], [3, 'z']]);
m.forEach(function (value, key, map) {
    console.log(value);
});
```

函数

如果没有 return语句，函数执行完毕后也会返回结果，只是结果为 undefined。

传入参数

JavaScript还有一个免费赠送的关键字 arguments，它只在函数内部起作用，并且永远指向当前函数调用者传入的所有参数。arguments类似 Array但它不是一个 Array，利用 arguments，你可以获得用者传入的所有参数。也就是说，即使函数不定义任何参数，还是可以拿到参数的值，arguments最用于判断传入参数的个数。

ES6标准引入了rest参数，上面的函数可以改写为：

```
function foo(a, b, ...rest) {
    console.log('a = ' + a);
    console.log('b = ' + b);
    console.log(rest);
}

foo(1, 2, 3, 4, 5);
// 结果:
// a = 1
// b = 2
// Array [ 3, 4, 5 ]
```

变量提升

var声明的变量会有变量提升，但是const和let没有。变量提升首先是udefined，只有在赋值语句的置才会赋值。

当使用一个没有赋值的变量时，是undefined，使用没被声明的变量会直接报错。

名字空间

全局作用域：window。基本所有的都挂在window下。

不同的JavaScript文件如果使用了相同的全局变量，或者定义了相同名字的顶层函数，都会造成命名突，并且很难被发现。减少冲突的一个方法是把自己的所有变量和函数全部绑定到一个全局变量中

局部作用域

块级作用域的变量：let 和 const

解构

```
let [x, [y, z]] = ['hello', ['JavaScript', 'ES6']];
let [, , z] = ['hello', 'JavaScript', 'ES6']; // 忽略前两个元素，只对z赋值第三个元素
var person = {
    name: '小明',
    age: 20,
    gender: 'male',
    passport: 'G-12345678',
    school: 'No.4 middle school',
    address: {
        city: 'Beijing',
        street: 'No.1 Road',
        zipcode: '100001'
    }
};
var {name, address: {city, zip}} = person;
name; // '小明'
city; // 'Beijing'
zip; // undefined, 因为属性名是zipcode而不是zip
// 注意: address不是变量，而是为了让city和zip获得嵌套的address对象的属性:
address; // Uncaught ReferenceError: address is not defined

var {name, single=true} = person;

// 声明变量:
var x, y;
// 解构赋值:
{x, y} = { name: '小明', x: 100, y: 200};
// 语法错误: Uncaught SyntaxError: Unexpected token =
这是因为JavaScript引擎把{开头的语句当作了块处理，于是=不再合法。解决方法是用小括号括起来：

({x, y} = { name: '小明', x: 100, y: 200});
```

交换变量

```
var x=1, y=2;
[x, y] = [y, x]
```

如果一个函数接收一个对象作为参数，那么，可以使用解构直接把对象的属性绑定到变量中,使用解构值可以减少代码量。

方法

```
var xiaoming = {
    name: '小明',
    birth: 1990,
    age: function () {
        var y = new Date().getFullYear();
        return y - this.birth;
    }
};

xiaoming.age; // function xiaoming.age()
xiaoming.age(); // 今年调用是25,明年调用就变成26了

function getAge() {
    var y = new Date().getFullYear();
    return y - this.birth;
}

var xiaoming = {
    name: '小明',
    birth: 1990,
    age: getAge
};

xiaoming.age(); // 25, 正常结果
getAge(); // NaN
```

在一个独立的函数调用中，根据是否是strict模式，this指向 undefined或 window，不过，我们还是以控制 this的指向的。

函数本身的 apply方法，它接收两个参数，第一个参数就是需要绑定的 this变量，第二个参数是 Array表示函数本身的参数。call()把参数按顺序传入。对普通函数调用，我们通常把 this绑定为 null

装饰器

利用 apply()，我们还可以动态改变函数的行为。JavaScript的所有对象都是动态的，即使内置的函数我们也可以重新指向新的函数。

```
'use strict';

var count = 0;
var oldParseInt = parseInt; // 保存原函数

window.parseInt = function () {
    count += 1;
    return oldParseInt.apply(null, arguments); // 调用原函数
};
```

高阶函数

一个函数就可以接收另一个函数作为参数，这种函数就称之为高阶函数。高阶函数除了可以接受函数为参数外，还可以把函数作为结果值返回。

map/reduce

map()方法定义在JavaScript的 Array中,返回一个新的函数

Array的 reduce()把一个函数作用在这个 Array的 [x1, x2, x3...]上，这个函数必须接收两个参数，reduce()把结果继续和序列的下一个元素做累积计算

filter

把 Array的某些元素过滤掉，然后返回剩下的元素。根据返回值是 true还是 false决定保留还是丢弃元素。

Sort

对于两个元素 x和 y，如果认为 x < y，则返回 -1，如果认为 x == y，则返回 0，如果认为 x > y，返回 1，这样，排序算法就不用关心具体的比较过程，而是根据比较结果直接排序。

字符串根据ASCII码进行排序，而小写字母 a的ASCII码在大写字母之后。sort()方法会直接对 Array进修改，它返回的结果仍是当前 Array

Array的方法

every 每个都满足 返回true ； some 存在满足条件的 返回true；find ，查找符合条件的第一个元素如果找到了，返回这个元素，否则，返回 undefined；findIndex， findIndex()和 find()类似，也是找符合条件的第一个元素，不同之处在于 findIndex()会返回这个元素的索引，如果没有找到，返回 -1 ；forEach ,forEach()和 map()类似，它也把每个元素依次作用于传入的函数，但不会返回新的数组。

闭包

内部函数可以引用外部函数的参数和局部变量。

当一个函数返回了一个函数后，其内部的局部变量还被新函数**引用**。

返回闭包时牢记的一点就是：返回函数不要引用任何循环变量，或者后续会发生变化的变量。

如果一定要引用循环变量怎么办？方法是再创建一个函数，用该函数的参数绑定循环变量当前的值，论该循环变量后续如何更改，已绑定到函数参数的值不变

```
function count() {
    var arr = [];
    for (var i=1; i<=3; i++) {
        arr.push(function () {
            return i * i;
        });
    }
    return arr;
}

var results = count();
var f1 = results[0];
var f2 = results[1];
var f3 = results[2];

f1(); // 16
```

```
f2(); // 16
f3(); // 16


function count() {
    var arr = [];
    for (var i=1; i<=3; i++) {
        arr.push((function (n) {
            return function () {
                return n * n;
            }
        })(i));
    }
    return arr;
}

var results = count();
var f1 = results[0];
var f2 = results[1];
var f3 = results[2];

f1(); // 1
f2(); // 4
f3(); // 9
```

箭头函数

ES6标准新增了一种新的函数：Arrow Function（箭头函数）。

但实际上，箭头函数和匿名函数有个明显的区别：箭头函数内部的 this是词法作用域，由上下文确定所以，用 call()或者 apply()调用箭头函数时，无法对 this进行绑定，即传入的第一个参数被忽略。

generator

generator（生成器）是ES6标准引入的新的数据类型。一个generator看上去像一个函数，但可以返多次。

generator和函数不同的是，generator由 function*定义（注意多出的 *号），并且，除了 return语，还可以用 yield返回多次。

函数在执行过程中，如果没有遇到 return语句（函数末尾如果没有 return，就是隐含的 return undefned;），控制权无法交回被调用的代码。

直接调用一个generator和调用函数不一样，仅仅是创建了一个generator对象，还没有去执行它。次遇到 yield x;就返回一个对象 {value: x, done: true/false}，然后“暂停”。如果 done为 true，则alue就是 return的返回值。

直接用 for ... of循环迭代generator对象，这种方式不需要我们自己判断 done

```
'use strict'

function* fib(max) {
```

```
    var
        t,
        a = 0,
        b = 1,
        n = 0;
    while (n < max) {
        yield a;
        [a, b] = [b, a + b];
        n ++;
    }
    return;
}

for (var x of fib(10)) {
    console.log(x); // 依次输出0, 1, 1, 2, 3, ...
}
```

generator和普通函数相比,可以在执行过程中多次返回，所以它看上去就像一个可以记住执行状态的数，利用这一点，写一个generator就可以实现需要用面向对象才能实现的功能。

generator还有另一个巨大的好处，就是把异步回调代码变成“同步”代码。

```
try {
    r1 = yield ajax('http://url-1', data1);
    r2 = yield ajax('http://url-2', data2);
    r3 = yield ajax('http://url-3', data3);
    success(r3);
}
catch (err) {
    handle(err);
}
```

标准对象

```
typeof 123; // 'number'
typeof NaN; // 'number'
typeof 'str'; // 'string'
typeof true; // 'boolean'
typeof undefined; // 'undefined'
typeof Math.abs; // 'function'
typeof null; // 'object'
typeof []; // 'object'
typeof {}; // 'object'
```

包装对象

```
var n = new Number(123); // 123,生成了新的包装类型
var b = new Boolean(true); // true,生成了新的包装类型
var s = new String('str'); // 'str',生成了新的包装类型

typeof new Number(123); // 'object'
new Number(123) === 123; // false
```

```
typeof new Boolean(true); // 'object'
new Boolean(true) === true; // false

typeof new String('str'); // 'object'
new String('str') === 'str'; // false
```

此时，Number()、Boolean和 String()被当做普通函数，把任何类型的数据转换为 number、boolean 、string类型（注意不是其包装类型）.

```
var n = Number('123'); // 123，相当于parseInt()或parseFloat()
typeof n; // 'number'

var b = Boolean('true'); // true
typeof b; // 'boolean'

var b2 = Boolean('false'); // true! 'false'字符串转换结果为true！因为它是非空字符串！
var b3 = Boolean(''); // false

var s = String(123.45); // '123.45'
typeof s; // 'string'
```

规则

- 不要使用 new Number()、new Boolean()、new String()创建包装对象
- 用 parseInt()或 parseFloat()来转换任意类型到 number
- 用 String()来转换任意类型到 string，或者直接调用某个对象的 toString()方法
- 通常不必把任意类型转换为 boolean再判断，因为可以直接写 if (myVar) {...}
- typeof操作符可以判断出 number、boolean、string、function和 undefined
- 判断 Array要使用 Array.isArray(arr)
- 判断 null请使用 myVar === null
- 判断某个全局变量是否存在用 typeof window.myVar === 'undefined'
- 函数内部判断某个变量是否存在用 typeof myVar === 'undefined'

任何对象都有 toString()方法吗？null和 undefined就没有！

number对象调用 toString()报SyntaxError

```
123.toString(); // SyntaxError
123..toString(); // '123', 注意是两个点!
(123).toString(); // '123'
```

Data

Date对象用来表示日期和时间。

```
var now = new Date(); // 获取系统当前时间
now; // Wed Jun 24 2015 19:49:22 GMT+0800 (CST)
now.getFullYear(); // 2015, 年份
now.getMonth(); // 5, 月份，注意月份范围是0~11，5表示六月
```

```
now.getDate(); // 24, 表示24号
now.getDay(); // 3, 表示星期三
now.getHours(); // 19, 24小时制
now.getMinutes(); // 49, 分钟
now.getSeconds(); // 22, 秒
now.getMilliseconds(); // 875, 毫秒数
now.getTime(); // 1435146562875, 以number形式表示的时间戳
```

当前时间是浏览器从本机操作系统获取的时间，所以不一定准确，因为用户可以把当前时间设定为任值。

```
var d = new Date(2015, 5, 19, 20, 15, 30, 123);
d; // Fri Jun 19 2015 20:15:30 GMT+0800 (CST)

var d = Date.parse('2015-06-24T19:49:22.875+08:00');
d; // 1435146562875

var d = new Date(1435146562875);
d; // Wed Jun 24 2015 19:49:22 GMT+0800 (CST)
d.getMonth(); // 5
```

Date对象表示的时间总是按浏览器所在时区显示的，不过我们既可以显示本地时间，也可以显示调整的UTC时间

```
var d = new Date(1435146562875);
d.toLocaleString(); // '2015/6/24 下午7:49:22'，本地时间（北京时区+8:00），显示的字符串与操系统设定的格式有关
d.toUTCString(); // 'Wed, 24 Jun 2015 11:49:22 GMT'，UTC时间，与本地时间相差8小时
```

RegExp

正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串义一个规则，凡是符合规则的字符串，我们就认为它“匹配”了，否则，该字符串就是不合法的。

```
var re = /^\d{3}\-\d{3,8}$/;
re.test('010-12345'); // true

'a b   c'.split(/\s+/); // ['a', 'b', 'c']
```

正则匹配默认是贪婪匹配，也就是匹配尽可能多的字符。

```
var re = /^(\d+)(0*)$/;
re.exec('102300'); // ['102300', '102300', '']
```

加个 ?就可以让 \d+采用非贪婪匹配

```
var re = /^(\d+?)(0*)$/;
re.exec('102300'); // ['102300', '1023', '00']
```

g，表示全局匹配

```
var r1 = /test/g;
// 等价于:
var r2 = new RegExp('test', 'g');
```

全局匹配可以多次执行 exec()方法来搜索一个匹配的字符串。当我们指定 g标志后，每次运行 exec()正则表达式本身会更新 lastIndex属性，表示上次匹配到的最后索引

```
var s = 'JavaScript, VBScript, JScript and ECMAScript';
var re=/[a-zA-Z]+Script/g;

// 使用全局匹配:
re.exec(s); // ['JavaScript']
re.lastIndex; // 10

re.exec(s); // ['VBScript']
re.lastIndex; // 20

re.exec(s); // ['JScript']
re.lastIndex; // 29

re.exec(s); // ['ECMAScript']
re.lastIndex; // 44

re.exec(s); // null，直到结束仍没有匹配到
```

JSON

JSON是JavaScript Object Notation的缩写，它是一种数据交换格式。

在JSON中，一共就这么几种数据类型：

- number：和JavaScript的 number完全一致；
- boolean：就是JavaScript的 true或false；
- string：就是JavaScript的 string；
- null：就是JavaScript的 null；
- array：就是JavaScript的 Array表示方式——[]；
- object：就是JavaScript的 { ... }表示方式。

JSON还定死了字符集必须是UTF-8

JSON的字符串规定必须用双引号 ""，Object的键也必须用双引号 ""