


金三银四！我带你复习50个JavaScript「基础」知识点

原创 林三心不学挖掘机 前端之神 2022-02-25 09:37

收录于话题

#JavaScript 50 #面试 48 #ES6 18 #前端 46

 **前端之神**
现已有100+篇原创文章，全网粉丝高达1w+，面试过超过100+个前端程序员，全网获赞2w+，全网阅读量播放量超过60w，B站「面试进...」>
59篇原创内容

公众号

前言

大家好，我是林三心，用最通俗易懂的话讲最难的知识点是我的座右铭，基础是进阶的前提是我的初心

金三银四

金三银四快要到来了，希望各位想找工作的朋友们要做好准备啊，今天就带着大家来复习一下JavaScript的50个「基础」知识点哦~~~

开始复习

1、JavaScript有几种数据类型？

- number：数字类型
- string：字符串类型
- boolean：布尔值类型
- undefined：未定义类型
- null：空类型
- object：对象类型
- symbol：symbol类型
- bigint：大数字类型

2、JavaScript最大安全数字与最小安全数字？

```
console.log(Number.MAX_SAFE_INTEGER)
// 9007199254740991

console.log(Number.MIN_SAFE_INTEGER)
// -9007199254740991
```

3、深拷贝与浅拷贝的区别？

- 浅拷贝：只拷贝第一层，深层的依然是引用，改变深层会影响原对象
- 深拷贝：每一层都拷贝了，改变数据不会影响原对象

4、闭包是什么？

闭包是一个函数，是一个能让外部访问到函数内部的函数

- 优点：使外部能访问内部，延长内部变量寿命
- 缺点：滥用闭包造成内存泄漏

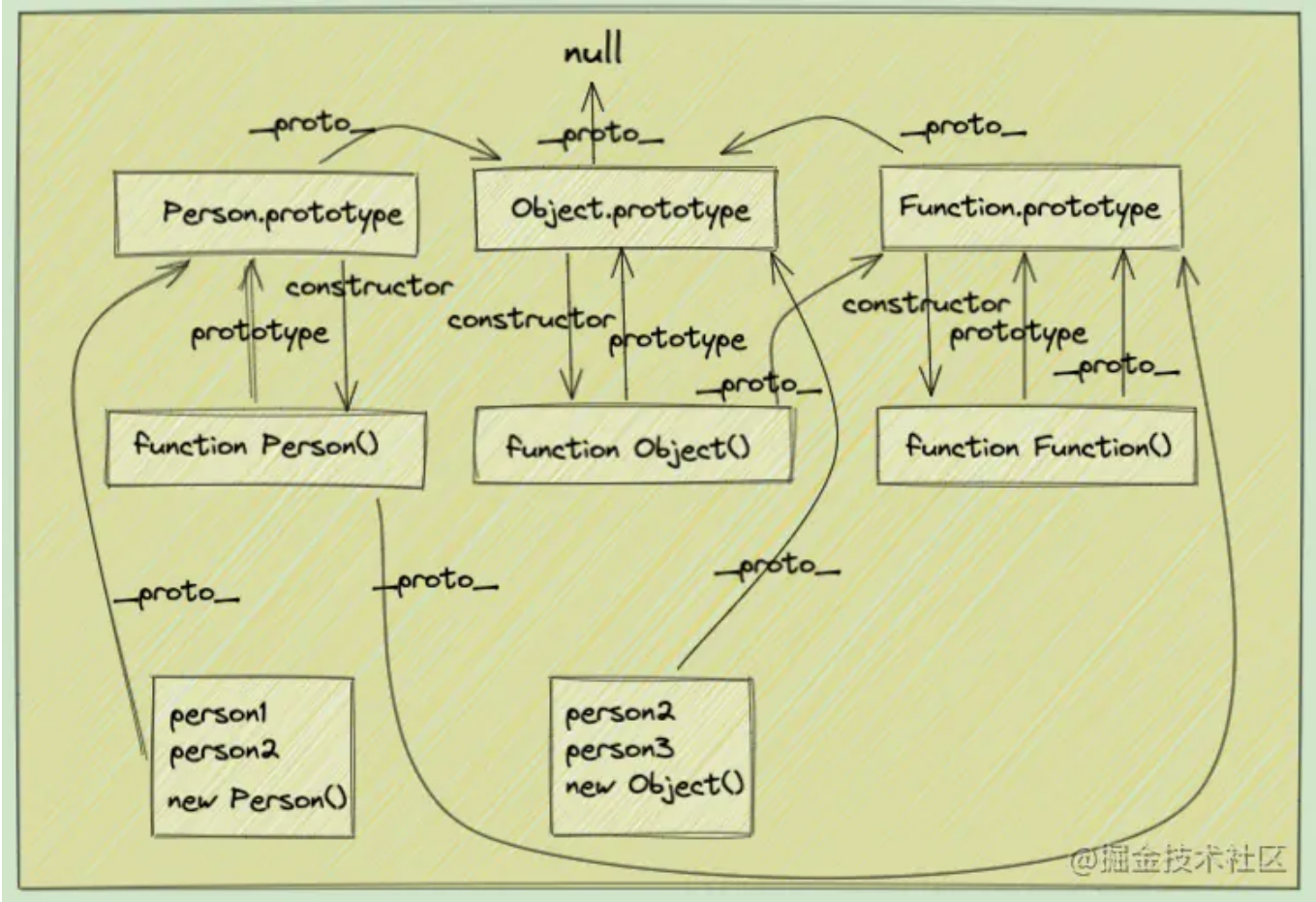
闭包的例子

```
function a () {
  let num = 0

  // 这是个闭包
  return function () {
    return ++num
  }
}
const b = a()
console.log(b()) // 1
console.log(b()) // 2
```

5、原型链是什么呀？

原型链是一条引用的链，实例的隐式原型指向构造函数的显式原型，可以使用 `A instanceof B` 来判断B是否在A的原型链上。



6、什么是变量提升？函数提升？

- 变量提升

```
console.log(name) // undefined
var name = 'Sunshine_Lin'

if (false) {
  var age = 23
}

console.log(age) // undefined 不会报错
```

- 函数提升

```
console.log(fun) // function fun() {}
function fun() {}

if (false) {
  function fun2(){}
}

console.log(fun2) // undefined 不会报错
```

- 函数提升优先级 > 变量提升优先级

```
console.log(fun) // function fun() {}
```

```
var fun = 'Sunshie_Lin'

function fun() {}

console.log(fun) // 'Sunshie_Lin'
```

7、isNaN 与 Number.isNaN的区别？

- `isNaN`：除了判断NaN为true外，还会把不能转成数字判断为true，例如'dasd'
- `Number.isNaN`：只会判断NaN为true

8、解决遍历对象时，把原型上的属性遍历出来了咋办？

使用 `hasOwnProperty` 判断

```
function Person(name) {
  this.name = name
}
Person.prototype.age = 23

const person = new Person('Sunshine_lin')

for (const key in person) { console.log(key) } // name age

// 使用 hasOwnProperty

for (const key in person) {
  person.hasOwnProperty(key) && console.log(key)
} // name
```

9、valueOf 与 toString？

- `valueOf` 比较偏向于计算，`toString` 偏向于显示
- 对象转换时，优先调用 `toString`
- 强转字符串时优先调用 `toString`，强转数字时优先调用 `valueOf`
- 正常情况下，优先调用 `toString`
- 运算操作符情况下优先调用 `valueOf`

10、JavaScript变量在内存中具体存储形式？

- `基本数据类型`：存在栈内存里
- `引用数据类型`：指针存栈内存，指向堆内存中一块地址，内容存在堆内存中
- 也有说法说其实JavaScript所有数据都存堆内存中，我也比较赞同这种说法

11、讲一讲JavaScript的装箱和拆箱？

装箱：把基本数据类型转化为对应的引用数据类型的操作

看以下代码，s1只是一个基本数据类型，他是怎么能调用indexOf的呢？

```
const s1 = 'Sunshine_Lin'

const index = s1.indexOf('_')

console.log(index) // 8
```

原来是JavaScript内部进行了装箱操作

- 1、创建String类型的一个实例；
- 2、在实例上调用指定的方法；
- 3、销毁这个实例；

```
var temp = new String('Sunshine_Lin')

const index = temp.indexOf('_')

temp = null

console.log(index) // 8
```

拆箱：将引用数据类型转化为对应的基本数据类型的操作

通过valueOf或者toString方法实现拆箱操作

```
var objNum = new Number(123);
var objStr =new String("123");
console.log( typeof objNum ); //object
console.log( typeof objStr ); //object
console.log( typeof objNum.valueOf() ); //number
console.log( typeof objStr.valueOf() ); //string

console.log( typeof objNum.toString() ); // string
console.log( typeof objStr.toString() ); // string
```

12、null和undefined的异同点有哪些？

相同点

- 1、都是空类型
- 2、转布尔值都是false，都是假值
- 3、null == undefined 为 true

不同点

- 1、typeof，前者为object，后者为undefined
- 2、null转数字为0，undefined转数字为NaN
- 3、null === undefined 为 false

13、如何判断数据类型？

- typeof：能判断string、number、undefined、boolean、function、object（null是object）
- Object.prototype.toString.call()：能判断大部分数据类型
- instanceof

14、为什么typeof null 是object？

不同数据类型底层都是用二进制来表示的，二进制前三位为 000 则会被判断为object，而null二进制全是0，所以被判断成object

15、== 与 === 的区别？

- ==：比较过程会进行隐式转换
- ===：值相同，类型相同才会为true

16、JavaScript的隐式转换规则？

- 转string类型：+（字符串连接符）
- 转number类型：++/--(自增自减运算符) + - * / %(算术运算符) > < >= <= == != === !== (关系运算符)
- 转boolean：！（逻辑非运算符）

17、双等号左右两边的转换规则？

- 1、null == undefined 为 true
- 2、如果有一个操作数是布尔值，则在比较相等性之前先将其转换为数值——false转换为0，而true转换为1；
- 3、如果一个操作数是字符串，另一个操作数是数值，在比较相等性之前先将字符串转换为数值
- 4、如果一个操作数是对象，另一个操作数不是，则调用对象的toString()方法，用得到的基本类型值按照前面的规则进行比较

18、undefined >= undefined 为什么是 false ？

隐式转换，变成 NaN >= NaN ， NaN 不等于自身也不大于自身

19、`null >= null` 为什么是 `true`？

隐式转换，变成 `0 >= 0`，为`true`

20、`[] == ![]` 为什么是 `true` ？

- 第一步：转为 `[] == false`
- 第二步：转为 `[] == 0`
- 第三步：转为 `'' == 0`
- 第四步：转为 `0 == 0`

21、`0.1 + 0.2 === 0.3`，对吗？

不对，JavaScript存在 `精度丢失` 问题，由于有些小数无法用二进制表示，所以只能取近似值，解决方法有：

- 先转大数，再变小数
- 使用 `toFixed`

22、什么是匿名函数？

匿名函数，就是没有名字的函数，比如：

```
(function(x, y){
    alert(x + y);
})(2, 3)
```

23、绑定点击事件有几种方式？

三种

- `xxx.onclick = function (){}`
- `<xxx onclick=""></xxx>`
- `xxx.addEventListener('click', function(){}), false)`

24、`addEventListener`的第三个参数是干嘛的？

决定事件是 `捕获阶段` 执行还是 `冒泡阶段` 执行

- `true`：捕获
- `false`：默认，冒泡

25、函数声明和函数表达式的区别？

- `函数声明`：享受函数提升
- `函数表达式`：归类于变量声明，享受变量提升
- `函数提升优先级 > 变量提升优先级`

```
console.log(fun) // fun () {}
// 函数表达式
var fun = function(name) {}
// 函数声明
function fun () {}
console.log(fun) // fun (name) {}
```

26、JavaScript的事件流模型有哪些？

- `事件冒泡`：由最具体的元素接收，并往上传播
- `事件捕获`：由最不具体的元素接收，并往下传播
- `DOM事件流`：事件捕获 -> 目标阶段 -> 事件冒泡

27、Ajax、Axios、Fetch有啥区别？

- **Ajax**：是对XMLHttpRequest(XHR)的封装
- **Axios**：是基于Promise对XHR对象的封装
- **Fetch**：是window的一个方法，基于Promise，与XHR无关，不兼容IE

28、load、\$(document).ready、DOMContentLoaded的区别？

- **\$(document).ready、DOMContentLoaded**：DOM树构建完毕，但还没有请求静态资源
- **load**：静态资源请求完毕

29、如何阻止事件冒泡？

```
function stopBubble(e) {  
  if (e.stopPropagation) {  
    e.stopPropagation()  
  } else {  
    window.event.cancelBubble = true;  
  }  
}
```

30、如何阻止事件默认行为？

```
function stopDefault(e) {  
  if (e.preventDefault) {  
    e.preventDefault();  
  } else {  
    window.event.returnValue = false;  
  }  
}
```

31、什么是事件委托？

当子元素都需要绑定相同事件时，可以将事件绑在父元素上，优点有：

- 绑定在父元素，则只需绑定一次，节省性能
- 后续新增的子元素也可以触发父元素绑定的事件

32、如何实现数组去重？

```
// 使用 Map 去重  
function quchong1(arr) {  
  const newArr = []  
  arr.reduce((pre, next) => {  
    if (!pre.get(next)) {  
      pre.set(next, 1)  
      newArr.push(next)  
    }  
    return pre  
  }, new Map())  
  return newArr  
}  
  
// 使用 Set 去重  
function quchong (arr) {  
  return [...new Set(arr)]  
}
```

33、Set与Array的区别是什么？

- Set使用has判断有无元素，数组使用索引

- Set添加元素使用方法add，数组用push、unshift
- Set长度为size，数组为length
- Set会自动把同样的基础数据类型去重，数组不能
- Set删除元素用delete，数组用splice、pop、shift
- Set可以使用clear清空，数组需要重新赋值[]
- 数组可以传入new Set(array)，实现数组转Set
- Set可以使用keys、value方法，转数组
- Set自带forEach方法进行遍历

34、Map与Object的区别是什么？

- Map使用set设置属性，对象使用obj[key] = value
- Map使用get获取属性值，对象使用obj[key]
- Map使用has判断属性存在与否，对象只能obj[key]
- Map删除元素使用delete方法，对象使用delete关键字
- Map使用clear进行情空，对象需要重新赋值{}
- Map和对象都可以使用entries方法转数组键值对
- Map自带forEach方法进行遍历

35、NaN是什么？有什么特点？

- typeof NaN 为 number
- NaN不等于自身，不大于自身，不小于自身
- NaN可以使用 Number.isNaN 判断
- NaN是假值，转布尔值为false

36、处理异步的方法有哪些？

- 回调函数
- Promise
- 事件监听
- 发布订阅
- async/await

37、JavaScript继承方式有几种？

前置工作

```
// 定义一个动物类
function Animal (name) {
  // 属性
  this.name = name || 'Animal';
  // 实例方法
  this.sleep = function(){
    console.log(this.name + '正在睡觉! ');
  }
}
// 原型方法
Animal.prototype.eat = function(food) {
  console.log(this.name + '正在吃: ' + food);
};
```

1、原型链继承

核心：将父类的实例作为子类的原型

```
function Cat(){
```

```
}
Cat.prototype = new Animal();
Cat.prototype.name = 'cat';

var cat = new Cat();
console.log(cat.name); // cat
cat.eat('fish') // cat正在吃: fish
cat.sleep() // cat正在睡觉!

console.log(cat instanceof Animal); //true
console.log(cat instanceof Cat); //true
```

优点：

- 1、非常纯粹的继承关系，实例是子类的实例，也是父类的实例
- 2、父类新增原型方法/属性，子类都能访问到
- 3、简单，易于实现 缺点：
- 1、要想为子类新增属性和方法，必须要在 `new Animal()` 这样的语句之后执行，不能放构造器中
- 2、来自原型对象的所有属性被所有实例共享
- 3、创建子实例时，无法向父类构造函数传参
- 4、不支持多继承

2、构造继承

核心：使用父类的构造器来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）

```
function Cat(name) {
  Animal.call(this);
  this.name = name || 'Tom';
}

var cat = new Cat();
console.log(cat.name); // Tom
cat.sleep() // Tom正在睡觉!

console.log(cat instanceof Animal); // false
console.log(cat instanceof Cat); // true
```

优点：

- 1、解决了 `原型链继承` 中，子类实例共享父类引用属性的问题
- 2、创建子类实例时，可以向父类传递参数
- 3、可以实现多继承(call多个父类对象) 缺点：
- 1、实例并不是父类的实例，知识子类的实例
- 2、是能继承父类的实例属性和方法，不能继承原型属性/方法
- 3、无法实现函数复用，每个子类都有父类实例函数的副本，影响性能

3、实例继承

核心：为父类实例添加新特性，作为子类实例返回

```
function Cat(name){
  var instance = new Animal();
  instance.name = name || 'Tom';
  return instance;
}

var cat = new Cat();
console.log(cat.name) // Tom
cat.sleep() // Tom正在睡觉!
```



```
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); // false
```

优点：

- 1、不限制调用方式，不管是 `new 子类()` 还是 `子类()`，返回的对象具有相同效果 缺点：
- 1、实例是父类的实例，不是子类的实例
- 2、不支持多继承

4、拷贝继承

核心：就一个一个拷贝

```
function Cat(name){
    var animal = new Animal();
    for(var p in animal){
        Cat.prototype[p] = animal[p];
    }
    this.name = name || 'Tom';
}

var cat = new Cat();
console.log(cat.name); // Tom
cat.sleep() // Tom正在睡觉!
console.log(cat instanceof Animal); // false
console.log(cat instanceof Cat); // true
```

优点：

- 1、支持多继承 缺点：
- 1、效率低，内存占用高（因为要拷贝父类的属性）
- 2、无法获取父类不可枚举方法（不可枚举方法，不能使用for in访问到）

5、组合继承

核心：通过父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用

```
function Cat(name){
    Animal.call(this);
    this.name = name || 'Tom';
}
Cat.prototype = new Animal();

Cat.prototype.constructor = Cat;

var cat = new Cat();
console.log(cat.name); // Tom
cat.sleep() // Tom正在睡觉!
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); // true
```

优点：

- 1、弥补了 `构造继承` 的缺陷，可以继承实例属性/方法，也可继承原型属性/方法
- 2、既是子类的实例，也是父类的实例
- 3、不存在引用属性共享问题
- 4、可传参
- 5、函数可复用 缺点：
- 1、调用了两次父类构造函数，生成了两份实例（子类实例将子类原型上的那份屏蔽了）

6、寄生组合继承

核心：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造时，就不会初始化两次实例方法/属性，避免 继承组合 的缺点

```
function Cat(name) {
  Animal.call(this);

  this.name = name || 'Tom';
}
// 创建一个没有实例方法的类
var Super = function () { };
Super.prototype = Animal.prototype;
//将实例作为子类的原型
Cat.prototype = new Super();

// Test Code
var cat = new Cat();
console.log(cat.name); // Tom
cat.sleep() // Tom正在睡觉!
console.log(cat instanceof Animal); // true
console.log(cat instanceof Cat); //true
```

优点：

- 1、堪称完美 缺点：
- 1、实现复杂

38、创建一个对象的方式有哪几种？

- new Object

```
const obj = new Object()
obj.name = 'Sunshine_Lin'
```

- 字面量

```
const obj = { name: 'Sunshin_Lin' }
```

- 工厂模式

```
function createObj(name) {
  const obj = new Object()
  obj.name = name
  return obj
}
const obj = createObj('Sunshine_Lin')
```

- 构造函数

```
function Person(name) {
  this.name = name
}
const person = new Person('Sunshine_Lin')
```

39、this指向的四种情况？

- new操作符

```
function Person(name) {
  this.name = name
  console.log(this)
```

```
}

// this指向当前person实例对象

const person = new Person('Sunshine_Lin')
```

- 指向window

```
function fn() {

    console.log(this)

}

fn() // 浏览器window, node里global
```

- 对象调用方法

```
const target = {

    fn: function () { console.log(this) }

}

target.fn() // target

// 这种就是改变了this了

const fn = target.fn

fn() // 浏览器window, node里global
```

- 箭头函数

```
const obj = {

    name: '林三心',

    fn: () => {

        console.log(this.name)

    }

}

console.log(obj.fn()) // undefined
```

- call、apply、bind改变this

```
const obj1 = {

    name: '林三心',

    sayName: function() {

        console.log(this.name)

    }

}

const obj2 = {

    name: 'Sunshin_Lin'

}

// 改变sayName的this指向obj2

obj1.sayName.call(obj2) // Sunshin_Lin

// 改变sayName的this指向obj2

obj1.sayName.apply(obj2) // Sunshin_Lin

// 改变sayName的this指向obj2

const fn = obj1.sayName.bind(obj2)

fn() // Sunshin_Lin
```

40、数组的常用方法有哪些？

方法	作用	是否影响原数组
push	在数组后添加元素，返回长度	✓
pop	删除数组最后一项，返回被删项	✓
shift	删除数组第一项，返回被删项	✓
unshift	数组开头添加元素，返回长度	✓

方法	作用	是否影响原数组
reverse	反转数组，返回数组	✓
sort	排序数组，返回数组	✓
splice	截取数组，返回被截取部分	✓
join	将数组变字符串，返回字符串	✗
concat	连接数组	✗
map	相同规则处理数组项，返回新数组	✗
forEach	遍历数组	✗
filter	过滤数组项，返回符合条件的数组	✗
every	每一项符合规则才返回true	✗
some	只要有一项符合规则就返回true	✗
reduce	接受上一个return和数组下一项	✗
flat	数组扁平化	✗
slice	截取数组，返回被截取区间	✗

41、Math的常用方法有哪些？

方法	作用
Math.max(...arr)	取arr中的最大值
Math.min(...arr)	取arr中的最小值
Math.ceil(小数)	小数向上取整
Math.floor(小数)	小数向下取整
Math.round(小数)	小数四舍五入
Math.sqrt(num)	对num进行开方
Math.pow(num, m)	对num取m次幂
Math.random() * num	取0-num的随机数

42、哪些因素导致内存泄漏？如何解决？

后面出一篇文章专门讲

43、讲讲JavaScript的垃圾回收机制

看这篇文章

赠你13张图，助你20分钟打败了「V8垃圾回收机制」！！！！

44、JS中有哪些不同类型的弹出框？

在JS中有三种类型的弹出框可用，分别是：

- Alert
- Confirm
- Prompt

45. 如何将 JS 日期转换为ISO标准

```
var date = new Date();
var n = date.toISOString();
console.log(n);

// YYYY-MM-DDTHH:mm:ss.sssZ
```

46、如何在JS中编码和解码 URL

- 编码：encodeURIComponent
- 解码：decodeURIComponent

47、什么是BOM？有哪些api？

BOM就是 `browser object model`，浏览器对象模型

api	作用	代表方法或属性
window.history	操纵浏览器的记录	history.back() history.go(-1)
window.innerHeight	获取浏览器窗口的高度	
window.innerWidth	获取浏览器窗口的宽度	
window.location	操作刷新按钮和地址栏	location.host: 获取域名和端口 location.hostname: 获取主机名 location.port: 获取端口号 location.pathname: 获取url的路径 location.search: 获取?开始的部分 location.href: 获取整个url location.hash: 获取#开始的部分 location.origin: 获取当前域名 location.navigator: 获取当前浏览器信息

48、BOM 和 DOM 的关系

BOM全称Browser Object Model，即浏览器对象模型，主要处理浏览器窗口和框架。

DOM全称Document Object Model，即文档对象模型，是 HTML 和XML 的应用程序接口（API），遵循W3C 的标准，所有浏览器公共遵守的标准。

JS是通过访问**BOM**（Browser Object Model）对象来访问、控制、修改客户端(浏览器)，由于**BOM**的window包含了document，window对象的属性和方法是直接可以使用而且被感知的，因此可以直接使用window对象的document属性，通过document属性就可以访问、检索、修改XHTML文档内容与结构。因为document对象又是DOM的根节点。

可以说，BOM包含了DOM(对象)，浏览器提供出来给予访问的是BOM对象，从BOM对象再访问到DOM对象，从而js可以操作浏览器以及浏览器读取到的文档。

49、JS中的substr()和substring()函数有什么区别

substr() 函数的形式为substr(startIndex,length)。它从startIndex返回子字符串并返回'length'个字符数。

```
var s = "hello";  
( s.substr(1,4) == "ello" ) // true
```

substring() 函数的形式为substring(startIndex,endIndex)。它返回从startIndex到endIndex - 1的子字符串。

```
var s = "hello";  
( s.substring(1,4) == "ell" ) // true
```

50、解释一下 "use strict" ？

“use strict”是Es5中引入的js指令。使用“use strict”指令的目的是强制执行严格模式下的代码。在严格模式下，咱们不能在不声明变量的情况下使用变量。早期版本的js忽略了“use strict”。

结语

点个【赞】和【在看】是对林三心最大的鼓励，林三心会非常开心的~~~

关注公众号【前端之神】， 回复【加群】， 即可获得加入【千人前端学习大群】的方式



前端之神

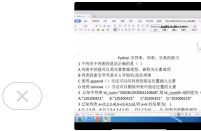
现已有100+篇原创文章，全网粉丝高达1w+，面试过超过100+个前端程序员，全网获赞2w+，全网阅读量播放量超过60w，B站「面试进...>
59篇原创内容

公众号

喜欢此内容的人还喜欢

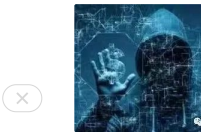
Python字符串，列表，字典的练习

开心开心开心



python代码画出“冰墩墩”和“时钟”（附源码）

Hacking黑白红



系统学习 TypeScript （三）——基础类型

编程三昧

