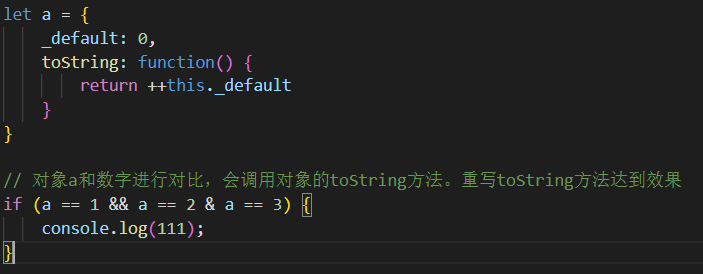
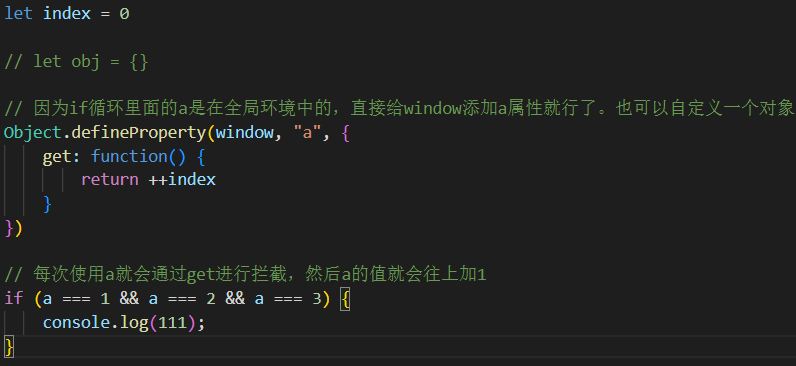
### 第一题 Object.defineProperty()

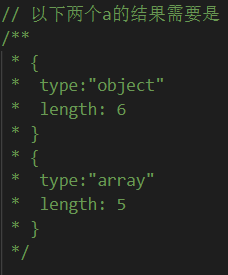
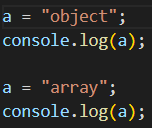
==



===



#### 思维扩散1



#### 思维扩散2



#### 思维扩散3



### 第二题



结果：30

解析：{}进行运算，会隐式转换为”[object Obejct]”



结果：0

解析：数组重写了toString方法，[1，2，3].toString() 🡺 “1,2,3”

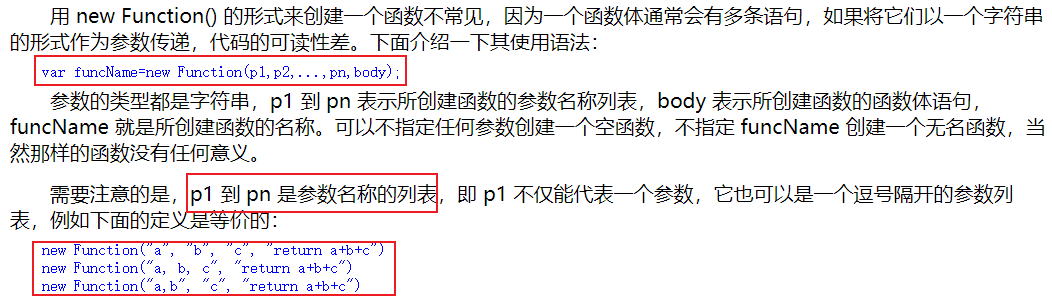
[] = “”



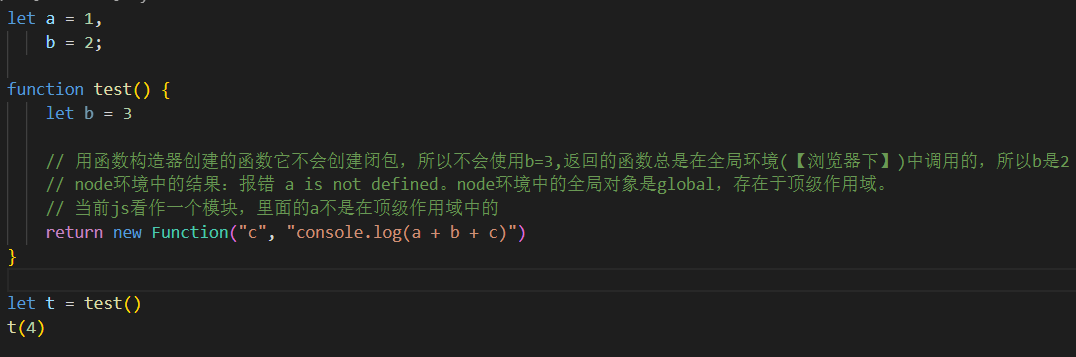
结果：0

解析：代码相当于

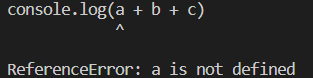
### 第三题：Function()相关



**通过new Function()声明的函数只能访问自己的作用域和全局的作用域**



node环境下的结果：



浏览器下的结果：

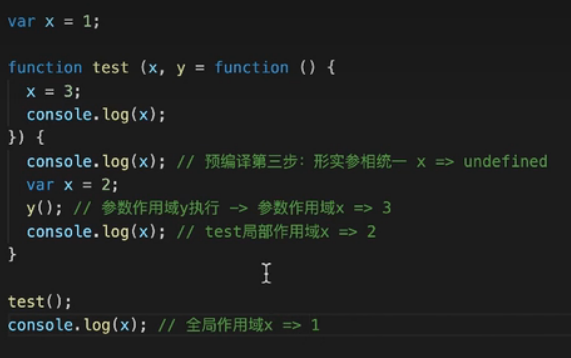


下面这两种方式调用Function是一样的



### 第四题(重点)：

[【全网首发:已完结】一道引发思考的『作用域间的纠葛』笔试题【知识储备】\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1Wq4y167UZ?spm_id_from=333.999.0.0&vd_source=a67e813436aadc4608992466bdaea6bc)

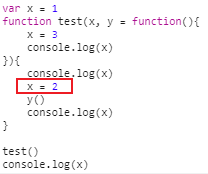


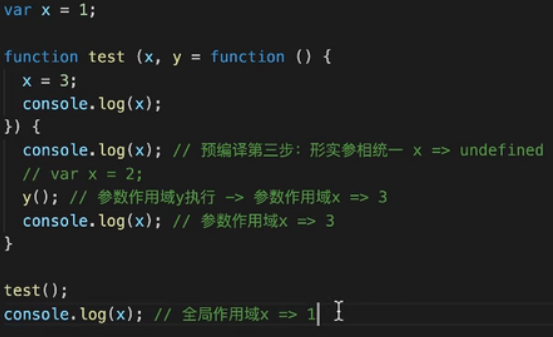
形参位置是有独立空间保存变量的.和函数体内部声明的变量，所在的作用域不同

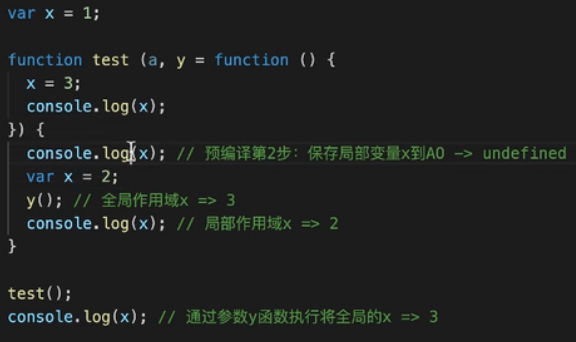
y的函数里面的x就是指向形参x

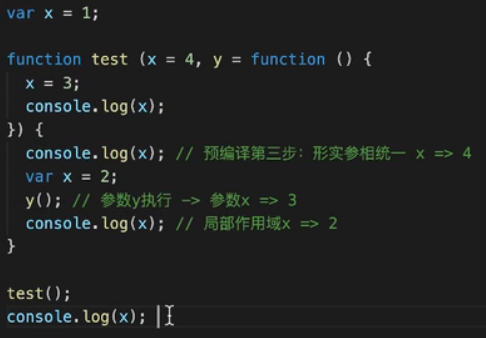
如果函数体里面重新let 了和形参名x一样的变量，那么这个变量x是当前函数体内的局部变量，和形参没有关系。函数体内部获取x变量，是从当前作用域里面获取。

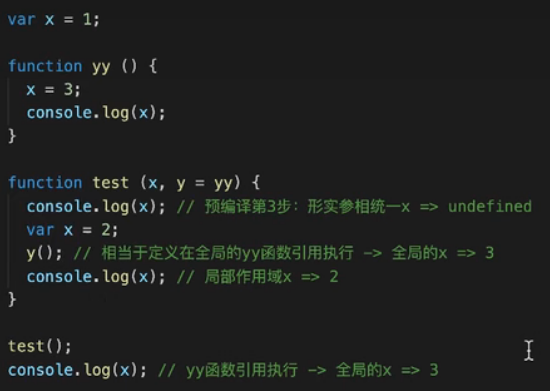
如果没有重新let 那么x就是形参x

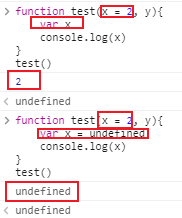
 





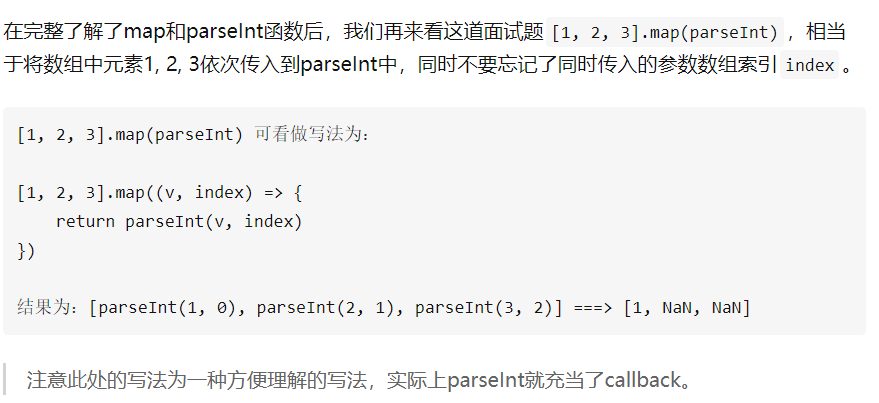




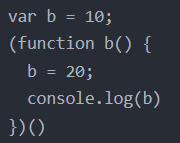


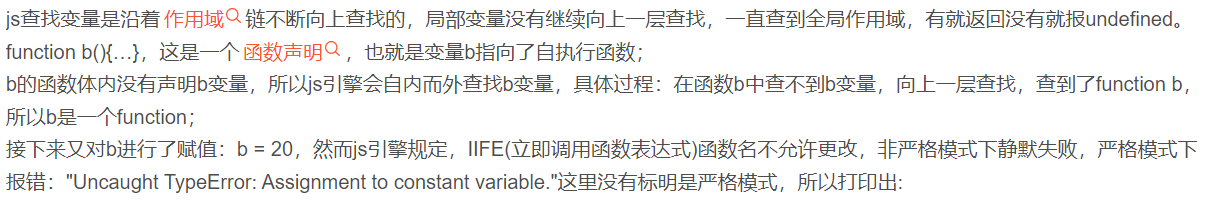
### 第五题

[1, 2, 3].map(parseInt)结果为？

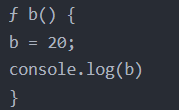


### 第六题

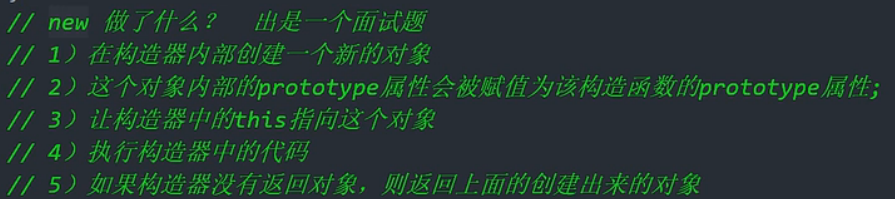


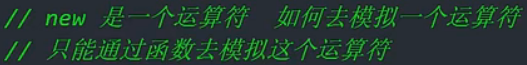


结果是立即执行函数



### new





闭包的概念

一个定义在函数内部的函数，可以读取到其他函数内部变量的函数，本质上，闭包就是一个把函数内部和外部连接起来的桥梁。

简单来说就是指有权访问另外一个函数作用域中的变量的函数（闭包就是能够读取其他函数内部变量的函数）

闭包的作用

函数内部定义的变量属于局部变量，当局部变量所在的函数被调用的时候，就开始执行，当调用执行一旦结束，局部变量就会被释放，当我们需要函数内部变量时，他已经被释放了，读取不到了，这个时候我们就可以用闭包，延长局部变量的执行时间，当函数执行完毕以后，局部变量不可以被内存释放，然后让外部可以访问到这个变量。

闭包的用途

闭包可以用在许多地方。它的最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。

闭包主要特点：

1、函数套函数，闭包一定有嵌套函数

2、外层函数一定有局部变量，且内层函数一定操作了外层函数的这个变量

3、内层函数会使用return返回外部（如果不返回这个内层函数，你就没办法使用这个闭包，返回内层函数的最终的目的就是让外部可以访问到这个闭包）

注意

外层函数被多次调用都会创建新的作用域，内层函数操作的外层函数的局部变量之间是不会影响的

闭包的好处

可以重复使用变量，并且不会造成变量污染

全局变量可以重复使用，但是容易造成变量污染。局部变量仅在局部作用域内有效，不可以重复使用，不会造成变量污染。

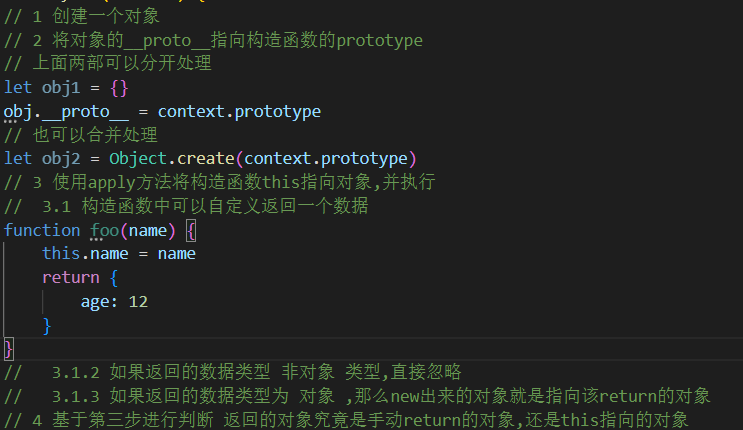
闭包的坏处

1.闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，可能会导致内存泄露

解决这个问题的办法就是在不使用这些变量时，及时把不需要的局部变量全部删除

2.闭包会在父函数外部，改变父函数内部变量的值。

如果你把父函数当作对象使用，把闭包当作它的公用方法（Public），把内部变量当作它的私有属性（private）时，要小心不要随便改变父函数内部变量的值。



#### 实现代码

function myNew(context) {

    // 1 创建一个对象

    // 2 将对象的\_\_proto\_\_指向构造函数的prototype

    // 3 使用apply方法将构造函数this指向对象,并执行

    //  3.1 构造函数中可以自定义返回一个数据

    //   3.1.2 如果返回的数据类型 非对象 类型,直接忽略

    //   3.1.3 如果返回的数据类型为 对象 ,那么new出来的对象就是指向该return的对象

    // 4 基于第三步进行判断 返回的对象究竟是手动return的对象,还是this指向的对象

    const obj = new Object

    obj.\_\_proto\_\_ = context.prototype

    let res = context.apply(obj, [...arguments].slice(1))

    return typeof res === "object" ? res : obj

}

function Foo(name, age) {

    this.name = name

    this.age = age

    this.fn = function() {

        console.log(this.name);

    }

    // return {

    //     gender: "王五"

    // }

}

let s = myNew(Foo, "zs", 12)

console.log(myNew(Foo, "zs", 12));

s.fn()

### 额外附加题

#### JSON.stringify()深拷贝

##### 弊端

如果 obj 的属性有时间对象，则 JSON.stringify 后再 JSON.parse 的结果，时间将转为字符串的形式，而不是对象的形式

如果 obj 的属性有 RegExp (正则表达式的缩写)、Error 对象，则序列化的结果将只得到空对象；

如果 obj 的属性有函数（function），undefined，Symbol， 则序列化的结果会把 function 或 undefined 丢失；

如果obj 的属性有 NaN、Infinity 和 -Infinity，则序列化的结果会变成 null

JSON.stringify() 只能序列化对象的可枚举的自有属性，例如如果 obj 中的对象是有构造函数生成的， 则使用 JSON.parse(JSON.stringify(obj)) 深拷贝后，会丢弃对象的 constructor；

如果对象中存在循环引用的情况也无法正确实现深拷贝，并且会导致死循环，最后抛出异常

-----------------------------------

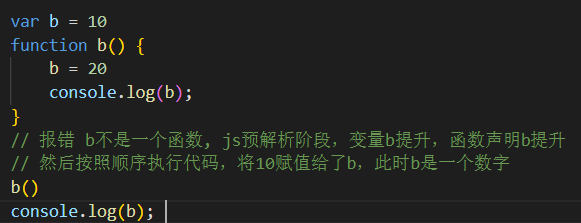
##### 其他常用场景

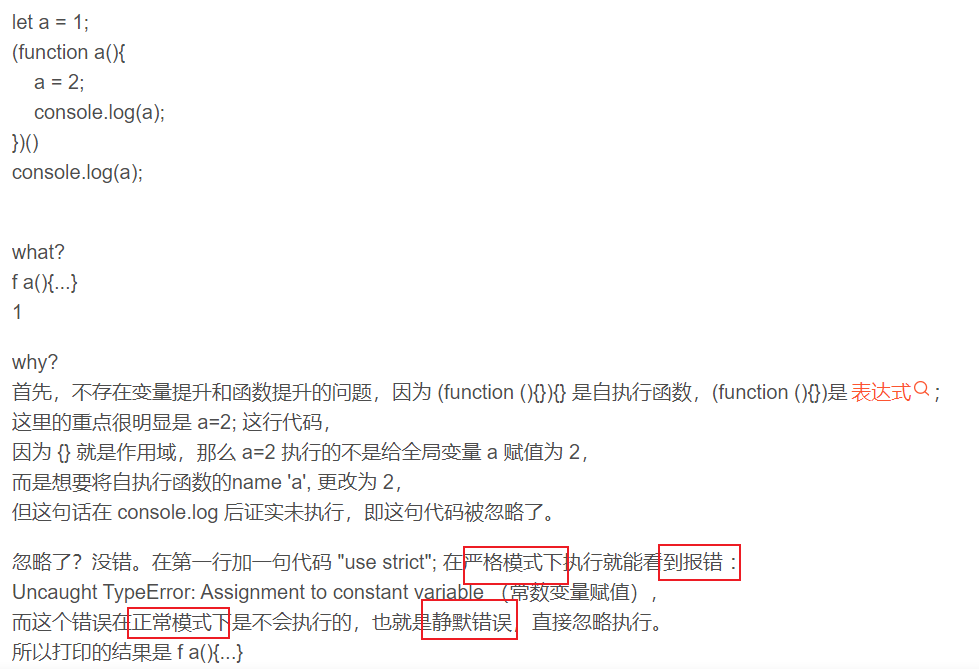
判断数组是否包含某对象： 将数组和要比对的对象转换成字符串后，比对字符串包含关系。（对象顺序很关键）

判断对象是否相等（对象顺序很关键）

让 localStorage / sessionStorage 可以存储对象

### 其他题目1





## 自定义的forEach解决异步问题

let arr = [

  () => console.log("start"),

  () => sleep(1000),

  () => console.log(1),

  () => sleep(2000),

  () => console.log("end"),

];

function sleep(delay) {

  return new Promise((resolve, reject) => {

    setTimeout(() => {

      resolve();

    }, delay);

  });

}

Array.prototype.myForEach = async function (fn) {

  let arr = this,

    arrLen = arr.length,

    \_self = arguments[1];

  for (let i = 0; i < arrLen; i++) {

    await fn.apply(\_self, [arr[i], i, arr]);

  }

};

async function fn(arr) {

  // for (let i = 0; i < arr.length; i++) {

  //   await arr[i]();

  // }

  arr.myForEach(async (item) => {

    await item();

  });

}

fn(arr);

# js中继承的情况有哪几种

## 第一种：类继承

class Father {

  constructor(name) {

    this.name = name;

  }

}

class Child extends Father {

  constructor(age) {

    super("张三");

    this.age = age;

  }

}

const c = new Child(12);

console.log(c);



## 第二种：原型链继承

特点：1、实例可继承的属性有：

实例的构造函数的属性，父类构造函数属性，父类原型的属性。（新实例不会继承父类实例的属性！

缺点：1、新实例无法向父类构造函数传参。

2、继承单一。

3、所有新实例都会共享父类实例的属性。（原型上的属性是共享的，一个实例修改了原型属 性，另一个实例的原 型属性也会被修改！）

function Father() {

  this.name = "zhangsan";

}

function Child() {

  this.age = 12;

}

Child.prototype = new Father();

Child.prototype.constructor = Child;

const c = new Child();

console.log(c.constructor);

## 第三种：借用构造函数继承

特点：1、只继承了父类构造函数的属性，没有继承父类原型的属性。

2、解决了原型链继承缺点1、2、3。

3、可以继承多个构造函数属性（call多个）。

4、在子实例中可向父实例传参。

缺点：1、只能继承父类构造函数的属性。

2、无法实现构造函数的复用。（每次用每次都要重新调用）

3、每个新实例都有父类构造函数的副本，臃肿。

function Father() {

  this.name = "zhangsan";

}

function Child() {

  Father.call(this);

  this.age = 12;

}

const c = new Child();

console.log(c.name);

## 第四种：组合式继承(原型链继承+借用构造函数继承)

缺点：父构造函数new了两次，性能不是很好

优点：1、可以继承父类原型上的属性，可以传参，可复用。

2、每个新实例引入的构造函数属性是私有的

function Father() {

  this.name = "zhangsan";

  this.eat = function () {

    console.log("吃东西");

  };

}

// 父构造函数的原型方法

Father.prototype.run = function () {

  console.log("跑步");

};

function Child() {

  // 通过借用构造函数继承的方式，继承父构造函数的实例属性

  Father.call(this);

  this.age = 12;

}

Child.prototype = new Father();

Child.prototype.constructor = Child;

const c = new Child();

console.log(c.name); // 父构造函数的name值

c.eat(); // 父构造函数实例方法

c.run(); // 父构造函数原型上的方法

## 第五种：寄生组合继承

重点：

就是给原型式继承外面套了个壳子。

优点：

没有创建自定义类型，因为只是套了个壳子返回对象（这个），这个函数顺理成章就成 了创建的新对象。

缺点：

没用到原型，无法复用。

function Father() {

  this.name = "zhangsan";

  this.eat = function () {

    console.log("吃东西");

  };

}

// 父构造函数的原型方法

Father.prototype.run = function () {

  console.log("跑步");

};

function Child() {

  // 通过借用构造函数继承的方式，继承父构造函数的实例属性

  Father.call(this);

  this.age = 12;

}

// 改动点，就是在组合式继承基础上进行了一些改动,看下方红字部分

// Child.prototype = new Father();

Child.prototype = Object.create(Father.prototype);

Child.prototype.constructor = Child;

const c = new Child();

console.log(c.name); // 父构造函数的name值

c.eat(); // 父构造函数实例方法

c.run(); // 父构造函数原型上的方法

console.log(c.constructor);

# ES6新特性

[(1条消息) ES6新特性有哪些？\_芮芮白的博客-CSDN博客\_es6新特性](https://blog.csdn.net/m0_48298027/article/details/111242626)

[面试官：ES6中对象新增了哪些扩展? · Issue #36 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/36)

## **新的原始类型和变量声明**

**symbol**

**let和const**

**解构赋值**

**Map和Set**

**新的对象和方法**

**属性简写**

拓展运算符(...)

Object,assign()

Object.is()

**字符串新方法**

includes()

startsWith()/endsWith()，判断字符串是否以参数字符串开头或结尾

padStart()/padEnd()，用参数字符串按给定长度从前面或后面补全字符串，返回新字符串

**数组的新方法**

from()

of()

find()和findIndex()

includes()

**函数**

**参数默认值**

**箭头函数**

**class（类）**

**模块导入和导出**

**Promise**

<https://www.liaoxuefeng.com/wiki/1022910821149312/1023024413276544>

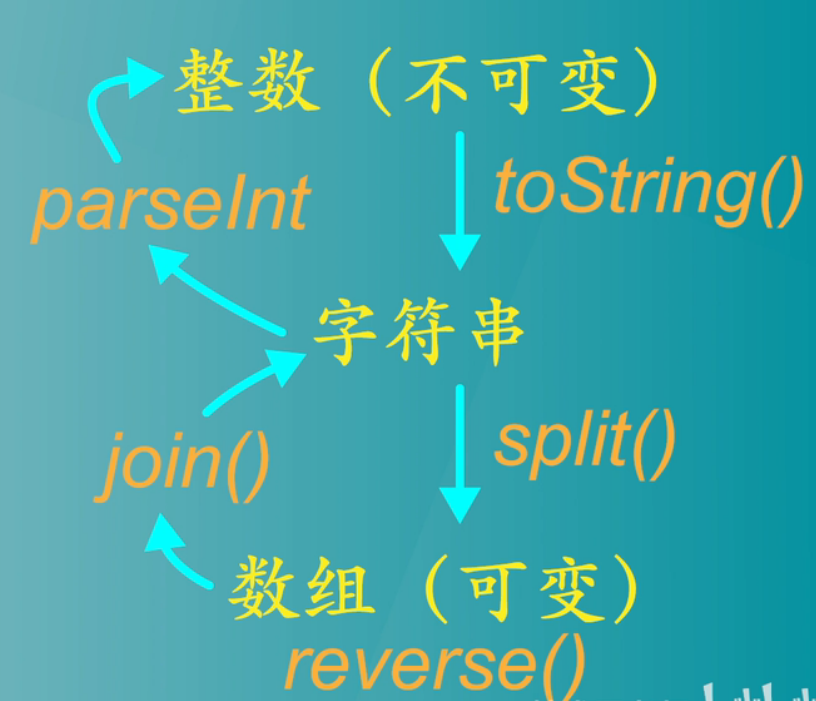
**Generator**

<https://www.liaoxuefeng.com/wiki/1022910821149312/1023024381818112>





# 整数反转



# js获取数组最后一位元素的几种方法



# get和post请求的区别

1.get请求一般是去取获取数据（其实也可以提交，但常见的是获取数据）；

post请求一般是去提交数据。

2.get因为参数会放在url中，所以隐私性，安全性较差，请求的数据长度是有限制的，

不同的浏览器和服务器不同，一般限制在 2~8K 之间，更加常见的是 1k 以内；

post请求是没有的长度限制，请求数据是放在body中；

3.get请求刷新服务器或者回退没有影响，post请求回退时会重新提交数据请求。

4.get请求可以被缓存，post请求不会被缓存。

5.get请求会被保存在浏览器历史记录当中，post不会。get请求可以被收藏为书签，因为参数就是url中，但post不能。它的参数不在url中。

6.get请求只能进行url编码（appliacation-x-www-form-urlencoded）,post请求支持多种（multipart/form-data等）。

## 深入理解

1…GET 和 POST都是http请求方式， 底层都是 TCP/IP协议；通常GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包（但firefox是发送一个数据包），

2.对于 GET 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200

（返回数据）表示成功；

而对于 POST，浏览器先发送 header，服务器响应 100， 浏览器再继续发送 data，服

务器响应 200 （返回数据）。

# 在地址栏里输入一个地址回车会发生哪些事情

1、***解析URL***：首先会对 URL 进行解析，分析所需要使用的传输协议和请求的资源的路径。如果输入的 URL 中的协议或者主机名不合法，将会把地址栏中输入的内容传递给搜索引擎。如果没有问题，浏览器会检查 URL 中是否出现了非法字符，如果存在非法字符，则对非法字符进行转义后再进行下一过程。

2、***缓存判断：***浏览器会判断所请求的资源是否在缓存里，如果请求的资源在缓存里并且没有失效，那么就直接使用，否则向服务器发起新的请求。

3、***DNS解析***： 下一步首先需要获取的是输入的 URL 中的域名的 IP 地址，首先会判断本地是否有该域名的 IP 地址的缓存，如果有则使用，如果没有则向本地 DNS 服务器发起请求。本地 DNS 服务器也会先检查是否存在缓存，如果没有就会先向根域名服务器发起请求，获得负责的顶级域名服务器的地址后，再向顶级域名服务器请求，然后获得负责的权威域名服务器的地址后，再向权威域名服务器发起请求，最终获得域名的 IP 地址后，本地 DNS 服务器再将这个 IP 地址返回给请求的用户。用户向本地 DNS 服务器发起请求属于递归请求，本地 DNS 服务器向各级域名服务器发起请求属于迭代请求。

4、***获取MAC地址：*** 当浏览器得到 IP 地址后，数据传输还需要知道目的主机 MAC 地址，因为应用层下发数据给传输层，TCP 协议会指定源端口号和目的端口号，然后下发给网络层。网络层会将本机地址作为源地址，获取的 IP 地址作为目的地址。然后将下发给数据链路层，数据链路层的发送需要加入通信双方的 MAC 地址，本机的 MAC 地址作为源 MAC 地址，目的 MAC 地址需要分情况处理。通过将 IP 地址与本机的子网掩码相与，可以判断是否与请求主机在同一个子网里，如果在同一个子网里，可以使用 APR 协议获取到目的主机的 MAC 地址，如果不在一个子网里，那么请求应该转发给网关，由它代为转发，此时同样可以通过 ARP 协议来获取网关的 MAC 地址，此时目的主机的 MAC 地址应该为网关的地址。

5、***TCP三次握手***： 下面是 TCP 建立连接的三次握手的过程，首先客户端向服务器发送一个 SYN 连接请求报文段和一个随机序号，服务端接收到请求后向客户端发送一个 SYN ACK报文段，确认连接请求，并且也向客户端发送一个随机序号。客户端接收服务器的确认应答后，进入连接建立的状态，同时向服务器也发送一个ACK 确认报文段，服务器端接收到确认后，也进入连接建立状态，此时双方的连接就建立起来了。

6、***HTTPS握手***： 如果使用的是 HTTPS 协议，在通信前还存在 TLS 的一个四次握手的过程。首先由客户端向服务器端发送使用的协议的版本号、一个随机数和可以使用的加密方法。服务器端收到后，确认加密的方法，也向客户端发送一个随机数和自己的数字证书。客户端收到后，首先检查数字证书是否有效，如果有效，则再生成一个随机数，并使用证书中的公钥对随机数加密，然后发送给服务器端，并且还会提供一个前面所有内容的 hash 值供服务器端检验。服务器端接收后，使用自己的私钥对数据解密，同时向客户端发送一个前面所有内容的 hash 值供客户端检验。这个时候双方都有了三个随机数，按照之前所约定的加密方法，使用这三个随机数生成一把秘钥，以后双方通信前，就使用这个秘钥对数据进行加密后再传输。

7、***返回数据***： 当页面请求发送到服务器端后，服务器端会返回一个 html 文件作为响应，浏览器接收到响应后，开始对 html 文件进行解析，开始页面的渲染过程。

8、***页面渲染***： 浏览器首先会根据 html 文件构建 DOM 树，根据解析到的 css 文件构建 CSSOM 树，如果遇到 script 标签，则判端是否含有 defer 或者 async 属性，要不然 script 的加载和执行会造成页面的渲染的阻塞。当 DOM 树和 CSSOM 树建立好后，根据它们来构建渲染树。渲染树构建好后，会根据渲染树来进行布局。布局完成后，最后使用浏览器的 UI 接口对页面进行绘制。这个时候整个页面就显示出来了。

9、***TCP四次挥手***： 最后一步是 TCP 断开连接的四次挥手过程。若客户端认为数据发送完成，则它需要向服务端发送连接释放请求。服务端收到连接释放请求后，会告诉应用层要释放 TCP 链接。然后会发送 ACK 包，并进入 CLOSE\_WAIT 状态，此时表明客户端到服务端的连接已经释放，不再接收客户端发的数据了。但是因为 TCP 连接是双向的，所以服务端仍旧可以发送数据给客户端。服务端如果此时还有没发完的数据会继续发送，完毕后会向客户端发送连接释放请求，然后服务端便进入 LAST-ACK 状态。客户端收到释放请求后，向服务端发送确认应答，此时客户端进入 TIME-WAIT 状态。该状态会持续 2MSL（最大报文生存时间，指报文段在网络中生存的时间，超时会被抛弃） 时间，若该时间段内没有服务端的重发请求的话，就进入 CLOSED 状态。当服务端收到确认应答后，也便进入 CLOSED 状态。

# 说一下HTTP和HTTPS协议的区别?

1、HTTPS协议需要CA证书,费用较高;而HTTP协议不需要

2、HTTP协议是超文本传输协议,信息是明文传输的,HTTPS则是具有安全性的SSL加密传输协议;

3、使用不同的连接方式,端口也不同,HTTP协议端口是80,HTTPS协议端口是443; 4、HTTP协议连接很简单,是无状态的;HTTPS协议是具有SSL和HTTP协议构建的可进行加密传输、身份认证的网络协议,比HTTP更加安全

# 为什么token要存在vuex中还要存本地一份

用户登录后，访问其他页面需要携带token，vuex是储存在内存里面的，而内存的特点就是快，将token存在vuex中可以提高获取token速度。因为localStorage的读取是一次磁盘读取，读取速度远低于vuex的内存读取，为了避免重复读取localStorage影响性能，需要将localStorage的数据放到vuex里维护。

由于vuex是储存在内存里面的，所以刷新页面就会消失，所以要存本地一份，刷新后token从本地获取。

每次更新token也是修改vuex中的token，然后再覆盖到localstorage中

* 本地存储
  + 获取麻烦
  + 数据不是响应式
* Vuex 容器（推荐）
  + 获取方便
  + 响应式的
* 登录成功，将 Token 存储到 Vuex 容器中
  + 获取方便
  + 响应式
* 为了持久化，还需要把 Token 放到本地存储
  + 持久化

# undefined和null的区别

**相同点**

第一：Undefined和Null两种数据类型都只有一个字面值，分别是undefined和null.

第二：Undefined和Null类型在转换为Boolean类型的值时，都会转换为false.

第三：在需要将两者转换成对象的时候，都会抛出一个TypeError的异常。

第四：Undefined类型派生自Null类型，所以在非严格相等的比较下，两者是相等的。

**不同点**

第一：null是JavaScript的关键字，而undefined是JavaScript的一个全局变量，也就是挂载在window对象上的一个变量，并不是关键字。

第二：在使用typeof运算符进行检测时，Undefined类型的值会返回undefined.而Null类型的值返回为object

第三：在进行数值类型的转换时，undefined会转换为NaN,无法参与计算，而null会转换为0,可以参与计算。

第四：建议：无论在什么情况下都没有必要将一个变量显示的赋值为undefined。如果需要定义某个变量来保存将来要使用的对象，应该将其初始化为null.

# 判断整数的不同方法

/\* 1. 添加一个是数字的判断 \*/  
function isInteger(obj) {  
 return typeof obj === 'number' && obj%1 === 0  
}  
  
/\* 2. 使用Math.round、Math.ceil、Math.floor判断 整数取整后还是等于自己。利用这个特性来判断是否是整数\*/  
function isInteger(obj) {  
 return Math.floor(obj) === obj  
}  
  
/\* 3. 通过parseInt判断 某些场景不准确 \*/  
function isInteger(obj) {  
 return parseInt(obj, 10) === obj  
}  
  
/\* 4. 通过位运算符\*/  
function isInteger(obj) {  
 return (obj | 0) === obj  
}  
  
/\* 5.ES6提供了Number.isInteger \*/

# dom节点平滑滚动到可视区域，顶部，底部

 document.body.scrollIntoView({ behavior: "smooth", block: "start" }) // 顶部  
 document.body.scrollIntoView({ behavior: "smooth", block: "end" }) // 底部  
 document.body.scrollIntoView({ behavior: "smooth"}) // 可视区域

# 各种数组克隆方法]

const clone = (arr) => arr.slice(0);  
const clone = (arr) => [...arr];  
const clone = (arr) => Array.from(arr);  
const clone = (arr) => arr.map((x) => x);  
const clone = (arr) => JSON.parse(JSON.stringify(arr));  
const clone = (arr) => arr.concat([]);  
const clone = (arr) => structuredClone(arr);

# 怎么在组件中监听路由参数的变化？（必会）

有两种方法可以监听路由参数的变化，但是只能用在包含<router-view />的组

件内。

**第一种**

watch: {

'$route'(to, from) {

// 在此处监听

},

},

**第二种**

beforeRouteUpdate (to, from, next) {

//这里监听

},

# commonjs和ES6模块的差异

[Module 的加载实现 - ES6 教程 - 网道 (wangdoc.com)](https://wangdoc.com/es6/module-loader.html#es6-%E6%A8%A1%E5%9D%97%E4%B8%8E-commonjs-%E6%A8%A1%E5%9D%97%E7%9A%84%E5%B7%AE%E5%BC%82)

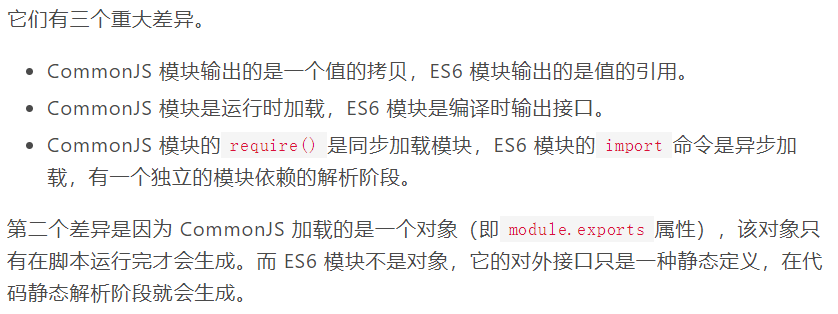
## **CommonJS**

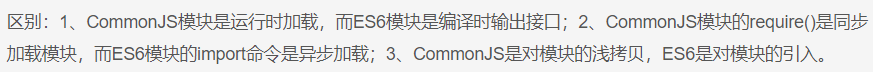
* 对于基本数据类型，属于复制。即会被模块缓存。同时，在另一个模块可以对该模块输出的变量重新赋值。
* 对于复杂数据类型，属于浅拷贝。由于两个模块引用的对象指向同一个内存空间，因此对该模块的值做修改时会影响另一个模块。
* 当使用require命令加载某个模块时，就会运行整个模块的代码。
* 当使用require命令加载同一个模块时，不会再执行该模块，而是取到缓存之中的值。也就是说，CommonJS模块无论加载多少次，都只会在第一次加载时运行一次，以后再加载，就返回第一次运行的结果，除非手动清除系统缓存。
* 循环加载时，属于加载时执行。即脚本代码在require的时候，就会全部执行。一旦出现某个模块被"循环加载"，就只输出已经执行的部分，还未执行的部分不会输出。

## **ES6模块**

* ES6模块中的值属于【动态只读引用】。
* 对于只读来说，即不允许修改引入变量的值，import的变量是只读的，不论是基本数据类型还是复杂数据类型。当模块遇到import命令时，就会生成一个只读引用。等到脚本真正执行时，再根据这个只读引用，到被加载的那个模块里面去取值。
* 对于动态来说，原始值发生变化，import加载的值也会发生变化。不论是基本数据类型还是复杂数据类型。
* 循环加载时，ES6模块是动态引用。只要两个模块之间存在某个引用，代码就能够执行。

## 区别





1.CommonJS 模块是运行时加载，ES6 模块是编译时输出接口。

2.CommonJS 模块的require()是同步加载模块，ES6 模块的import命令是异步加载，有一个独立的模块依赖的解析阶段。

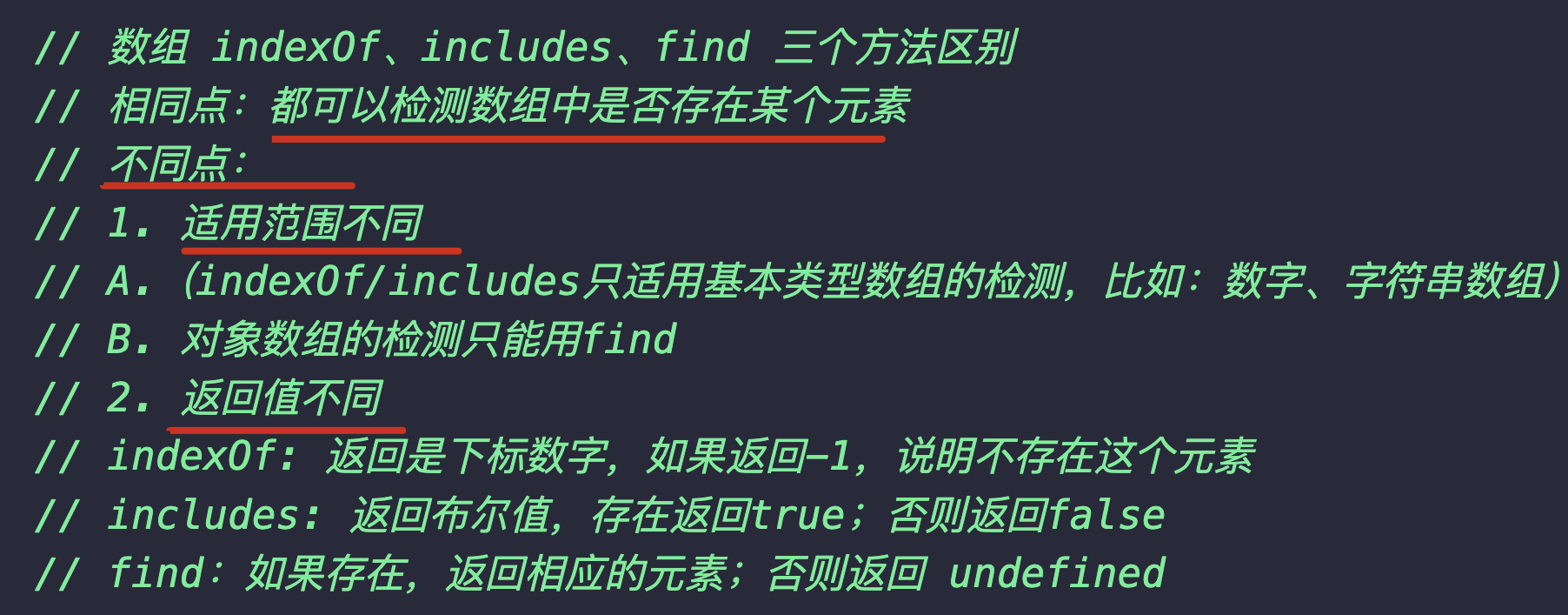
3.CommonJS是对模块的浅拷贝，ES6 Module是对模块的引入，即ES6 Module只存只读，不能改变其值，具体点就是指针指向不能变，类似const 。

4.import的接口是read-only（只读状态），不能修改其变量值。 即不能修改其变量的指针指向，但可以改变变量内部指针指向。可以对commonJS对重新赋值（改变指针指向），但是对ES6 Module赋值会编译报错。

## 共同点

1.CommonJS和ES6 Module都可以对引⼊的对象进⾏赋值，即对对象内部属性的值进行改变。

# 数组indexOf,includes,find方法的区别



# 为什么0.1+0.2!=0.3

* 进制转换 ：js 在做数字计算的时候，0.1 和 0.2 都会被转成二进制后无限循环 ，但是 js 采用的 IEEE 754 二进制浮点运算，最大可以存储 53 位有效数字，于是大于 53 位后面的会全部截掉，将导致精度丢失。

# 闭包

## 闭包的概念

一个定义在函数内部的函数，可以读取到其他函数内部变量的函数，本质上，闭包就是一个把函数内部和外部连接起来的桥梁。

简单来说就是指有权访问另外一个函数作用域中的变量的函数（闭包就是能够读取其他函数内部变量的函数）

## 闭包的作用

函数内部定义的变量属于局部变量，当局部变量所在的函数被调用的时候，就开始执行，当调用执行一旦结束，局部变量就会被释放，当我们需要函数内部变量时，他已经被释放了，读取不到了，这个时候我们就可以用闭包，延长局部变量的执行时间，当函数执行完毕以后，局部变量不可以被内存释放，然后让外部可以访问到这个变量。

## 闭包的用途

闭包可以用在许多地方。它的最大用处有两个，一个是可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。

## 闭包主要特点：

1、函数套函数，闭包一定有嵌套函数

2、外层函数一定有局部变量，且内层函数一定操作了外层函数的这个变量

3、内层函数会使用return返回外部（如果不返回这个内层函数，你就没办法使用这个闭包，返回内层函数的最终的目的就是让外部可以访问到这个闭包）

注意

外层函数被多次调用都会创建新的作用域，内层函数操作的外层函数的局部变量之间是不会影响的

## 闭包的好处

可以重复使用变量，并且不会造成变量污染

全局变量可以重复使用，但是容易造成变量污染。局部变量仅在局部作用域内有效，不可以重复使用，不会造成变量污染。

## 闭包的坏处

1.闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，可能会导致内存泄露

解决这个问题的办法就是在不使用这些变量时，及时把不需要的局部变量全部删除

2.闭包会在父函数外部，改变父函数内部变量的值。

如果你把父函数当作对象使用，把闭包当作它的公用方法（Public），把内部变量当作它的私有属性（private）时，要小心不要随便改变父函数内部变量的值。

# css3新增的特性

* [一、选择器](https://blog.csdn.net/weixin_44337386/article/details/124740203#_7)
* [二、新样式](https://blog.csdn.net/weixin_44337386/article/details/124740203#_10)
  + [1、边框](https://blog.csdn.net/weixin_44337386/article/details/124740203#1_11)
  + [2、背景](https://blog.csdn.net/weixin_44337386/article/details/124740203#2_16)
  + [3、文字](https://blog.csdn.net/weixin_44337386/article/details/124740203#3_22)
  + [4、颜色](https://blog.csdn.net/weixin_44337386/article/details/124740203#4_27)
* [三、transition过渡（一般和:hover搭配使用）](https://blog.csdn.net/weixin_44337386/article/details/124740203#transitionhover_31)
* [四、transform转换](https://blog.csdn.net/weixin_44337386/article/details/124740203#transform_56)
* [五、animation动画](https://blog.csdn.net/weixin_44337386/article/details/124740203#animation_110)
* [六、颜色渐变](https://blog.csdn.net/weixin_44337386/article/details/124740203#_120)
* [七、弹性布局+网络布局](https://blog.csdn.net/weixin_44337386/article/details/124740203#_132)

1. 为什么需要用vuex和cookie同时管理token？

2. token过期如何处理的？

3. 路由守卫如何做token拦截的？

4. 左侧菜单如何根据路由表动态生成的？

5. 代理跨域如何操作，原理是什么？

6. 环境变量配置的作用是什么？

7. 路由为何拆模块？如何拆模块？拆完如何用？

因为复杂中台项目的页面众多，不可能把所有的业务都`集中在一个文件上`进行管理和维护， (\*\*可维护性角度\*\*)

\*\*并且还有最重要的，前端的页面中主要分为两部分\*\* (我们的项目很可能带权限设计的!)

1. 一部分是所有人都可以访问的，(`静态路由` - 一直都有的路由) 比如: 登录页, 首页, 404

2. 一部分是只有有权限的人才可以访问的 (`动态路由` - 动态新增或移除的路由) 比如: 薪资管理, 人员管理

\*\*拆分多个模块便于更好的权限的控制 和 维护\*\*

# 构造函数和类的区别

Class在语法上更贴合面向对象的写法

Class实现继承更加易读、易理解

 class本质还是function.

 类的所有方法都定义在类的prototype属性上

 类创建的实例,里面也有\_\_proto\_\_ 指向类的prototype原型对象

 所以ES6的类它的绝大部分功能，ES5都可以做到，新的class写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已。

使用的时候，直接对类使用new命令，跟构造函数的用法完全一致，并且类必须用new调用，否则会报错

# 虚拟dom

我们先根据真实DOM生成一颗virtual DOM，当virtual DOM某个节点的数据改变后会生成一个新的Vnode，然后Vnode和oldVnode作对比，发现有不一样的地方就直接修改在真实的DOM上，然后使oldVnode的值为Vnode。

diff的过程就是调用名为patch的函数，比较新旧节点，一边比较一边给**真实的DOM**打补丁。

**Diff算法是一种对比算法**。对比两者是旧虚拟DOM和新虚拟DOM，对比出是哪个虚拟节点更改了，找出这个虚拟节点，并只更新这个虚拟节点所对应的真实节点，而不用更新其他数据没发生改变的节点，实现精准地更新真实DOM，进而提高效率。

新旧虚拟DOM对比的时候，Diff算法比较只会在同层级进行, 不会跨层级比较。 所以Diff算法是:深度优先算法。 时间复杂度:O(n)

当数据改变时，会触发setter，并且通过Dep.notify去通知所有订阅者Watcher，订阅者们就会调用patch方法，给真实DOM打补丁，更新相应的视图。

# key的作用

1、key的作用主要是为了更高效的对比虚拟DOM中的某个节点是否是相同节点。

2、Vue在patch过程中判断两个节点是否是相同节点key是一个必要条件，渲染一组列表时，key往往是唯一标识，所以如果不定义key的话，Vue只能认为比较的两个节点是同一个，哪怕它们实际上不是，这导致了频繁更新元素，使得整个patch过程比较低效，影响性能。

3、实际使用中在渲染一组列表时key必须设置，而且必须是唯一标识，应该避免使用数组索引作为key，这可能导致一些隐蔽的bug；Vue中在使用相同标签元素过渡切换时，也会使用key属性，其目的也是为了让Vue可以区分它们，否则Vue只会替换其内部属性而不会触发过渡效果。

4、从源码中可以知道，Vue判断两个节点是否相同时主要判断两者的key和元素类型等，因此如果不设置，它的值就是是undefined，则可能永远认为这是两个相同节点，只能去做更新操作，这造成了大量的DOM更新操作，明显是不可取的。

# vue.use()的原理

Vue.use方法的第一个参数只能接收对象与函数形式的参数，对象参数需要有install方法，当传入参数中是对象，且有install方法时，会自动执行其中的install方法，**并且install方法中的this依然指向该对象。**

传入该对象的install方法和函数中的第一个参数，必然是Vue对象本身

Vue.use会自动阻止多次注册相同的插件，届时即使多次调用也只会注册一次该插件

## 插件的作用：

插件通常用来为Vue添加全局功能

# MVVM

Model–View–ViewModel （MVVM） 是一个软件架构设计模式，由微软 WPF 和 Silverlight 的架构师 Ken Cooper 和 Ted Peters 开发，是一种简化用户界面的事件驱动编程方式。由 John Gossman（同样也是 WPF 和 Silverlight 的架构师）于2005年在他的博客上发表

MVVM 源自于经典的 Model–View–Controller（MVC）模式 ，MVVM 的出现促进了前端开发与后端业务逻辑的分离，极大地提高了前端开发效率，MVVM 的核心是 ViewModel 层，它就像是一个中转站（value converter），负责转换 Model 中的数据对象来让数据变得更容易管理和使用，该层向上与视图层进行双向数据绑定，向下与 Model 层通过接口请求进行数据交互，起呈上启下作用。如下图所示：

# 什么时候使用插槽

当子组件被复用时，需要在特定的区域展示不同的定制化的内容

# **什么是插槽？**

插槽（Slot）是Vue提出来的一个概念，正如名字一样，插槽用于决定将所携带的内容，插入到指定的某个位置，从而使模板分块，具有模块化的特质和更大的重用性。插槽显不显示、怎样显示是由父组件来控制的，而插槽在哪里显示就由子组件来进行控制

# 组件和插件的区别

两者的区别主要表现在以下几个方面：

* 编写形式
* 注册形式
* 使用场景



组件 (Component) 是用来构成你的 App 的业务模块，它的目标是 App.vue

插件 (Plugin) 是用来增强你的技术栈的功能模块，它的目标是 Vue 本身

简单来说，插件就是指对Vue的功能的增强或补充

# 状态码响应

## 400

访问网页显示400，其含义是**你访问的页面域名不存在或者请求错误。**

一般是因为我们输入的语法格式有错误，服务器无法理解用户的请求，不知道要表达的是什么。这个时候我们需要认真检查下语义、请求参数是否有误，不然再怎么刷新都没有用。

## 401

**表示服务器收到未经身份验证的请求。**由于您网站上特定网页或目标的凭据无效

## 403

访问网页显示403，就是今天小泽遇到的情况，这是一种常见的错误提示，**表示资源不可用。**服务器理解了客户的请求，但拒绝处理它，通常是由于服务器上文件或目录的权限设置导致的WEB访问错误。

## 405

出现405错误代码表示资源被禁止，对于请求所标识的资源，不允许使用请求行中所指定的方法。**有可能是文件目录权限不够导致的。**

# css工程化概念

让我们更好的去开发和维护，整理我们的css代码，让浏览器读取我们的css代码达到一个高效的解析速度，为我们未来的next语法去做准备，目前市面上实现的就是预处理器，打包工具

# drag拖拽的api

元素在拖放过程中触发的事件

先列下拖拽过程中可能触发的事件，如下：

dragstart：事件主体是被拖放元素，在开始拖放被拖放元素时触发。

darg：事件主体是被拖放元素，在正在拖放被拖放元素时触发。

dragenter：事件主体是目标元素，在被拖放元素进入某元素时触发。

dragover：事件主体是目标元素，在被拖放在某元素内移动时触发。

dragleave：事件主体是目标元素，在被拖放元素移出目标元素是触发。

drop：事件主体是目标元素，在目标元素完全接受被拖放元素时触发。

dragend：事件主体是被拖放元素，在整个拖放操作结束时触发。

其中事件主体是拖放元素的是，dragstart（开始拖动） 、darg（正在拖放） 、dragend（拖放结束），其他4个事件主体都是目标元素，进入、移动、离开、完全进入四个状态。

# transition属性

是一个简写属性，可用于设置四个过渡属性：

transition-property过渡效果的CSS属性的名称（height、width、opacity等）；

transition-duration完成过渡效果需要时间；

transition-timing-function规定速度效果的速度曲线（linear：线性过渡。ease：平滑过渡。ease-in：逐渐加速。ease-out：逐渐减速。ease-in-out：先加速后减速）；

transition-delay过渡效果何时开始（延迟时间）。

# 为什么不使用Object.defineProperty对数组做响应式

能监听到数组下标的变化，监听不到长度的变化，然后重写数组的7个方法，能监听到长度的变化，导致监听不到数组的下标的变化

defineProperty的劫持，是需要便利所有的属性；其实defineProperty是有能力检测数组下标变化的；只是vue2是放弃了这个方案，最大的问题就是考虑到数组的长度可能会特别大；性能问题太大；

# **insertAdjacentHTML()**

将文本解析为 **element** 元素，并将结果节点插入到DOM树中的指定位置。它不会重新解析它正在使用的元素，因此它不会破坏元素内的现有元素。这避免了额外的[序列化](https://so.csdn.net/so/search?q=%E5%BA%8F%E5%88%97%E5%8C%96&spm=1001.2101.3001.7020)步骤，使其比直接使用innerHTML操作更快。

element.insertAdjacentHTML(position,text)

position顾名思义，就是想要插入的位置，一共有4个固定的值

'beforebegin'：元素element自己的前面。

'afterbegin'：插入到元素element里面的第一个子节点之前（也就是总是会插入到最前面，例如我插入5个节点，顺序是1、2、3、4、5，那么我就需要以5、4、3、2、1的顺序插入，有一种栈结构先进后出的感觉）。

'beforeend'：插入元素element里面的最后一个子节点之后（这个比较容易理解，就是插入到最后一个节点后，例如我插入5个节点，顺序是1、2、3、4、5，那就正常的1、2、3、4、5就好啦，但是注意是在已有节点的后面哦）。

'afterend'：元素element自己的后面。

# babel的原理

babel的转移过程分为三个阶段，这三个步骤分别是：

1、解析parse:将代码解析生成抽象语法树（AST）,即词法分析和语法分析的过程。

2、转换Transform:对于AST进行变换的一些列的操作，babel接收得到的AST并通过babel-traverse对其进行遍历，在此过程中进行添加，更新以及移除等操作。

3、生成Generate:将变换后的AST再转换为JS代码，使用到的模块是babel-generator。