,map,set,let,const,class,symbol

节流防抖、bind底层、extend底层、$底层、cookie封装、扁平化、柯里化、promise原理、手写闭包、手写ajax、串行ajax请求处理、url处理、promise封装ajax。

**JS设计模式大全**

设计模式有六大原则：

1. 开闭原则。就是说模块应对扩展开放，而对修改关闭。
2. 里氏代换原则。如果调用的是父类的话，那么换成子类也完全可以运行。
3. 依赖倒转原则。把父类都替换成它的子类，程序的行为没有变化。
4. 接口隔离原则，每一个接口应该是一种角色，不多不少，不干不该干的事，该干的事都要干。
5. 单一职责原则。
6. 迪米特法则。 最少知识原则。

**发布订阅模式**

大大降低模块耦合度，便于扩展和维护

它定义了一种一对多的关系，让多个观察者对象同时监听某一个主题对象，这个主题对象的状态发生变化时就会通知所有的观察者对象，使得它们能够自动更新自己。

“订阅者/观察者”，指的是单个"event"

综合其过程，首先构造器函数初始化一个对象，为观察者列表，其键值对为

"event":[callback1,callback2.....]

订阅即添加上述键值对或向现存event添加callback，退订同理

发布事件则是在内部触发观察者列表的"event"对应所有callback

class Player {

constructor() {

// 初始化观察者列表

this.watchers = {}

} // es5 就直接写var watcher={},下面的都 function \_publish(event,data){ }这样

// 发布事件

\_publish(event, data) {

if (this.watchers[event] && this.watchers[event].length) {

this.watchers[event].forEach(callback => callback.bind(this)(data))

}

}

// 订阅事件

subscribe(event, callback) {

this.watchers[event] = this.watchers[event] || []

this.watchers[event].push(callback)

}

// 退订事件

unsubscribe(event = null, callback = null) {

// 如果传入指定事件函数，则仅退订此事件函数

if (callback&&event) {

if (this.watchers[event] && this.watchers[event].length) {

this.watchers[event].splice(this.watchers[event].findIndex(cb => Object.is(cb, callback)), 1) //Object.is(obj1,obj2) 判断两个是否相等

}

// 如果仅传入事件名称，则退订此事件对应的所有的事件函数

} else if (event) {

this.watchers[event] = []

// 如果未传入任何参数，则退订所有事件

} else {

this.watchers = {}

}

}

}

const player = new Player()

console.log(player)

const **onPlayerPlay1** = function(data) {

console.log('1: Player is play, the `this` context is current player', this, data)

}

const **onPlayerPlay2** = data => {

console.log('2: Player is play', data)

}

const **onPlayerPause** = data => {

console.log('Player is pause', data)

}

// 可订阅多个不同事件

player.subscribe('play', **onPlayerPlay1**)

player.subscribe('play', **onPlayerPlay2**)

player.subscribe('pause', **onPlayerPause**)

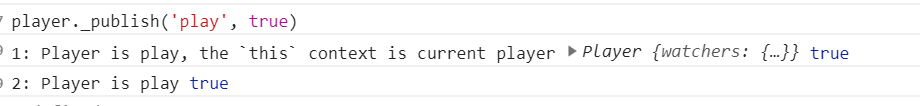
player.unsubscribe('play')

// 退订所有订阅事件

player.unsubscribe()

// 可以在外部手动发出事件（真实生产场景中，发布特性一般为类内部私有方法）

player.\_publish('loaded', true)



**单例设计模式**

保证一个类仅有一个实例,并提供一个访问它全局的点，也就是说调用一个类时，永远只返回同一个实例,一般通过闭包来实现，函数内创实例，返回则判断这个实例有没有，有则直接返回，没有则创建一个

var Single = (function(){

**var instance;**

function init() {

return { //define public methods and properties };

}

return {

// 获取实例,闭包刚好能找到函数内的instance

getInstance:function(){

**if(!instance){ instance = init(); }**

**return instance;**

} } })();

### 构造函数模式

function Animal(name, color){

this.name = name;

this.color = color;

this.getName = function(){ return this.name; }

}

### 工厂模式

工厂函数内部创造一个新实例，赋值并返回。

function Animal(opts){

**var obj = new Object();**

obj.name = opts.name;

obj.color = opts.color;

obj.getInfo = function(){

return '名称：'+obj.name +'， 颜色：'+ obj.color;

}

return obj; }  
**策略模式**  
一个对象里各种键包含函数，即为策略

var func={

text:function(){}，

default:function(){}

}

**桥接模式**

桥接模式的核心在于将抽象部分和它的实现部分分离，使它们都可以独立的变化。

比如要在点击事件时添加一个遮罩层。

首先，添加的遮罩层只能有一个，多次点击不会变多，则创建过程需要单例模式。

单例模式可能会放到全局变量里，要规避，则

抽象部分，是单例话创造的部分

实现部分，是创建node的过程，如此分离

//单例化过程

var singleton = function(fn){

var result;

return function() {

**return result || ( result = fn.apply(this,arguments));**

}

}

//创建节点过程

var origin = function(){

return document.body.appendChild(document.createElement('div'));

};

**var createMask = singleton(origin);**

**观察者模式**

**代理模式**

**装饰者模式**

策略模式

* 快排及其优化。
* 缓存算法实现及其优化（缓存算法简单模型：假设可以缓存三个数据，请求前三个数据时，直接进缓存列表，当请求第四个数据时，若命中缓存，将被缓存的数据放入缓存列表头部，否则把新加入的数据放入缓存列表头部，淘汰最后一个数据）。
* 怎么快速定位哪个组件出现性能问题。

**防抖与节流**

防抖（Debounce）和节流（throttle）都是用来控制某个函数在一定时间内执行多少次的技巧，两者相似而又不同。

函数去抖（debounce）：让一个函数在一定间隔内没有被调用时，才开始执行被调用方法。  
**只有几秒不操作才会触发，乘客上满了再开车。**

函数节流（throttle）：是让一个函数无法在很短的时间间隔内连续调用，当上一次函数执行后过了规定的时间间隔，才能进行下一次该函数的调用。**规定时间内只执行一次**

去抖Debounce

window.debounce = function(func,dely){

var timer = null;

return function(){

var context = this;

//执行回调函数的域

var args = arguments;

if(timer) { clearTimeout(timer) }; // 判断是否有定时器

**timer = setTimeout(function(){**

**timer = null ;**

**},dely)**

**if(!timer){ func.apply(context, args); } } }**

如果调用

foo = function(){

console.log('scroll work')

}

dom.addEventListener('scroll', debounce(foo, 2000));

节流Throttle

节流函数是处理类似场景但抖动不适合的另一种解决方案，比如大型电商网站当用户滚动到页面底部的时候再发AJAX请求获取图片，实现图片懒加载，如果使用去抖动，不管方案一还是二，都会用种奇怪的体验，假设设置500ms的delay时间，使用方案一，效果则是，用户滚动了，500ms后发AJAX获取图片，再显

示图片。期间500ms用户是只能看到图片缺失的。如果使用方案二，似乎是能实现需求，但是仔细想想，如果用户不是500ms滚动一次，而是玩命的在连续滚动，则AJAX只会触发一次，用户只能看到第一次滚动触发AJAX返回的图片，后面的则是图片缺失状态。

function throttle(func, wait) {

Let lastTime = null

Let timeout

return function () {

Let context = this;

Let now = newDate();

Let arg = arguments;

// 如果时间大于一个执行周期，则执行

if(now - lastTime - wait > 0) {

// 如果之前有了定时任务则清除

if(timeout) { clearTimeout(timeout) timeout = null}

func.apply(context, arg)

lastTime = now } elseif(!timeout) {

timeout = setTimeout(() => {

func.apply(context, arg) }, wait) } } }

4. 如何实现链式调用

对象深拷贝

var obj= JSON.parse(JSON.stringify(obj1))

spa相关（首屏加载，白屏问题，路由转换，seo）

.websocket原理和服务器端推送机制，长短轮询

## JavaScript 预解析

在预解析阶段，会将以关键字var和function开头的语句块提前进行处理。

当变量和函数的声明处在作用域比较靠后的位置的时候，变量和函数的声明会被提升到作用域的开头。

# 算法

斐波拉契数列

 function recurFib(n) {  
  if (n < 2) {  
    return n;  
  } else {  
    return recurFib(n-1) + recurFib(n-2);  
  } }

  return n<2?1:getNthFibonacci(n-1)+getNthFibonacci(n-2);

**找数组中（第一个）出现最多的元素，返回其和出现次数**

var str='fffddddfaaaa';

 var obj={};

 for(var i=0;i<str.length;i++){

    var cha=str.charAt(i);

if(!obj.hasOwnProperty(cha)){

  obj[cha]=1;

}else{ obj[cha]++; }

}

 var resultChar=0;

 var resultNum=0;

 for(var p in obj){

     if(obj[p]>resultNum){

    resultNum=obj[p];

resultChar=p;

}

 }

 console.log(resultChar+":"+resultNum);

**数组nums中去掉与val相同的元素，要求不能用新数组，显然要用splice**

var removeElement = function(nums, val) {

for(i=0;i<num.length;i++){

if(nums[i] == val){

nums.splice(i,1);

i--;

}

return nums.length;

}

public class Solution {

public int arrayPairSum(int[] nums) {

Arrays.sort(nums);

int result = 0;

for (int i = 0; i < nums.length; i += 2) {

result += nums[i];

}

return result;

}

}

清雨又在吃自助餐了。

排在清雨面前的有N种食物，排成一排，清雨可以选择其中的若干种食物，但是不能连续选择相邻的食物。因为清雨很挑食，当所有食物都不合口味时，他可以一种都不选，即一个都不选也算为一种方法。

请问他有多少种选择食物的方法呢？

Bob有N个空盒子，第i个盒子是边长为a[i]的立方体，如果一个盒子的边长严格小于另一个盒子，并且大盒子里面没有其他小盒子，小盒子也没有放入其他大盒子中，那么这个小盒子可以放入大盒子。Bob可以根据条件任意的放盒子，因为他想看见最少数量的盒子。没有放入其他盒子的盒子是可以被看到的，请你帮Bob确定能看见盒子的最少数量。