1. JavaScript

1.1. document. write 和 document.innerhtml 区别

write 是直接将内容写入页面的内容流，可能会导致页面全部重绘
innerHTML 将内容写入某个 dom 节点，不会导致页面全部重绘

1.2. 基础数据类型和引用数据类型的区别

基础数据类型，存储的是实际数据（String, Number, boolean, Null, Undefined）
引用数据类型，存储的是数据的引用地址（数组，对象）

基础数据类型，都是保存在栈内存中，内存会自动释放
引用数据类型，都是保存在堆内存中。需要手动清除。

1.3. 赋值与赋址有什么区别

赋值：
给一个变量赋予基本数据类型的值，叫做赋值。
多个变量赋予相同的值，变量之间的操作互不影响。
赋址：
给一个变量赋予引用数据类型的值，叫做赋址。
多个变量赋予相同的地址，变量之间的操作会相互影响。

1.4. 常用的基本数据类型与引用数据类型

基本数据类型：Undefined，Null，Boolean，Number 和 String
引用数据类型：Object，Array

1.5. js 判断数据类型常用的方法

typeof

typeof 是一个操作符，其右侧跟一个一元表达式，并返回这个表达式的数据类型。返回的结果用该类型的字符串(全小写字母)形式表示，包括 number,string,boolean,undefined,object,function,symbol 等。

instanceof

instanceof 用来判断 A 是否为 B 的实例，表达式为：A instanceof B，如果 A 是 B 的实例，则返回 true，否则返回 false。instanceof 检测的是原型，内部机制是通过判断对象的原型链中是否有类型的原型。

```
person instanceof Person        true
doctor instanceof Doctor        true
person instanceof Doctor        false
```

constructor

当一个函数 F 被定义时，JS 引擎会为 F 添加 prototype 原型，然后在 prototype 上添加一个 constructor 属性，并让其指向 F 的引用，F 利用原型对象的 constructor 属性引用了自身，当 F 作为构造函数创建对象时，原型上的 constructor 属性被遗传到了新创建的对象上，从原型链角度讲，构造函数 F 就是新对象的类型。这样做的意义是，让对象诞生以后，就具有可追溯的数据类型。

```
person.constructor == Person    true
```

1.6. 原型,原型链

原型：类形的 prototype.每个类都有一个 prototype 属性指向生成这个对象的原型对象，简称原型； prototype 可以对实例对象进行扩展，既添加原型对象的属性和方法

原型链：每个对象都有一个__proto__属性， 指向生成改对象的原型对象，这样，我们就可以找到是哪个对象生成了该对象， 原型链一般用于继承

原型链的核心就是依赖对象的__proto__的指向，当自身不存在的属性时，就一层层的扒出创建对象的原型对象，直至到 Object 时，就没有__proto__指向了。 因为 proto 实质找的是 prototype，所以我们只要找这个链条上的原型对象的 prototype。其中 Object.prototype 是没有 proto 属性的，它==null。

1.7. 常用的数组方法

push	在数组的末尾放值
splice	在指定位置添加，删除，或者替换元素
forEach	遍历数组元素
map	遍历数组，但是有返回值
reduce	对数组的值进行计算
filter	过滤数组
sort	数组排序
pop	弹出数组的最后一个元素
includes	判断数组中是否包含某个元素
shift	弹出数组的第一个元素

1.8. 手动实现数组的排序

```
选择排序
let arr = [1,2,3,6,7,8,11,9,4,5];
//进行排序
// 第 0 位和第 1，2，3，4，5，6，7，8 进行比较，把最值放到第 0 位  - 选择排序
for(let x=0;x<arr.length-1;x++){
    for(let y=x+1;y<arr.length;y++){
        if(arr[x] > arr[y]){
            let tmp = arr[x];
            arr[x] = arr[y];
            arr[y] = tmp;
        }
    }
}
console.log(arr);  //  1，2，3，4，5，6，7，8，9
```

1.9. 数组去重

```
第一种：
    let arr = [1,2,3,1,2,3,1,1,2,3,5,6,4];
    var newArr = [];
    arr.forEach(item=>{
        if(! newArr.includes(item)){
            newArr.push(item);
        }
    })
    arr = newArr;


第二种：
    let arr = [1,2,3,1,2,3,1,1,2,3,5,6,4];
    var set = new Set();
    arr.forEach(item=>{
        set.add(item);
```

```
    })
    arr = set;
```

1.10. this 指向 ，箭头函数的 this 指向

普通函数 this 的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定 this 到底指向谁，实际上 this 的最终指向的是那个调用它的对象，
但是，在箭头函数中，是不会改变 this 的指向的，函数外部 this 指向谁，函数内部的 this 仍然指向谁。

1.11. 是谁，能改变 tihs 指向

call 方法可以改变 this 的指向
特点：第一个参数是 this 指向的内容，后面的参数直接写即可

```
function f1 (name,age){
        console.log(this);//Date
        console.log(name,age);
    }
    f1.call(new Date(),"李闯",89)
```

apply 方法可以改变 this 的指向
特点：第一个参数是 this 指向的内容，后面的参数需要放到一个数组里面

```
function f1(name, age) {
        console.log("this-->",this);//Date
        console.log(name, age);
    }
    f1.apply(new Date(), ["李闯", 89])
```

bind 方法改变 this 指向
特点：第一个参数是 this 指向的内容，后面有几个参数直接写即可
返回一个新的方法，需要重新手动调用

```
function f1(name, age) {
        console.log("this-->", this);//Date
        console.log(name, age);
    }
    let newFun = f1.bind(new Date(), "李闯", 89)
    newFun();
```

1.12. 什么叫对象的拷贝（克隆）？

其实就相当于复制粘贴。就是创建另外一个一模一样的对象。形成了两个对象。只是两个对象的值一样的。

1.13. 深拷贝与浅拷贝有什么区别

浅拷贝：

创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值，如果属性是引用类型，拷贝的就是内存地址 ，所以如果其中一个对象改变了这个地址，就会影响到另一个对象。

常见方法：

Object.assign()　　对象浅拷贝

arr.concat()　数组浅拷贝

Arr.slice()　数组浅拷贝

深拷贝

深拷贝会拷贝所有的属性，并拷贝属性指向的动态分配的内存。当对象和它所引用的对象一起拷贝时即发生深拷贝。深拷贝相比于浅拷贝速度较慢并且花销较大。拷贝前后两个对象互不影响。

常用方法：

JSON.parse(JSON.stringify(object))

1.14. js 中的 cloneNode 方法是做什么的

主要是负责对节点的拷贝，分为影子拷贝和深度拷贝。影子拷贝表示只拷贝节点本身。
深度拷贝表示既要拷贝节点本身，也要拷贝所有的子节点

1.15. Js 继承的方式

原型继承
构造继承
组合继承
Es6 继承

1.16. Js 中的闭包是指什么

闭包就是指能够读取其他函数内部变量的函数。

```
function f1(){
                        let num = 10;
                        console.log(num)
                        return ()=>{
                                return ++num;
                        }
        }
```

```
        //当 f1 函数执行完毕的时候，由于内部函数引用了 num 变量，所以 num 变量并不会随着 f1 的完毕而消失
        //它会一直保存在内部函数当中
        let f2 = f1();
        console.log(f2())
        console.log(f2())

        let f3 = f1();
        console.log(f3())
```

1.17. 闭包使用场景

1. 函数防抖

在事件被触发 n 秒后再执行回调，如果在这 n 秒内又被触发，则重新计时。 实现的关键就在于 setTimeOut 这个函数，由于还需要一个变量来保存计时，考虑维护全局纯净，可以借助闭包来实现。

```
/*
 * fn [function] 需要防抖的函数
 * delay [number] 毫秒，防抖期限值
 */
function debounce(fn,delay){
  let timer = null
  //借助闭包
  return function() {
    if(timer){
      clearTimeout(timer) //进入该分支语句，说明当前正在一个计时过程中，并且又触发了相同事件。所以要取消当前的计时，重新开始计时
      timer = setTimeOut(fn,delay)
    }else{
      timer = setTimeOut(fn,delay) // 进入该分支说明当前并没有在计时，那么就开始一个计时
    }
  }
}
```

2. 函数赋值

```
var fn2;
function fn(){
  var name="hello";
  //将函数赋值给 fn2
  fn2 = function(){
    return name;
  }
```

```
}
fn()//要先执行进行赋值，
console.log(fn2())//执行输出 fn2
```

在闭包里面给 fn2 函数设置值，闭包的形式把 name 属性记忆下来，执行会输出 hello

3. 做缓存

比如求和操作，如果没有缓存，每次调用都要重复计算，采用缓存的方式，将上一次执行的结果通过闭包的方式进行保存，可以在下一次查询的时候进行访问上一次查询的值，这样就可以减少查询的次数，提高性能。

1.18. 防抖和节流

防抖：

抖动的现象：

当触发一个事件，到这个事件执行完毕，是需要一定的时间的。但是，如果在这个时间段内，该事件又重新触发，我们就称之为事件的抖动。

防抖的原理：

当事件触发的时候，就创造一个定时器，在定时器存在期间，如果再次触发事件，我们就重置定时器，直到定时器定时结束，才执行事件

js 实现防抖：

```
let _debonce = (fun, time) => {
    let obj = '';
    return () => {
        obj && clearTimeout(obj);
        obj = setTimeout(() => fun(), time)
    }
}
```

节流：

流动现象：

在短时间内，某一个事件多次触发

节流的原理

短时间内，限制某一个事件的执行次数。当事件执行完后，马上启动一个定时器，在定时器存续期间，若事件重复触发，不予执行。知道定时器定时结束。事件再次触发，才会执行，并且又会启动一个新的定时器。。

js 实现节流：

```
let _throwtle = (fun, time) => {
    let obj = '';
    return () => {
        if (obj) return;
        fun();
        obj = setTimeout(() => obj = null, time)
    }
}
```

1.19. 闭包产生问题

容易产生内存泄漏

内部函数 被 外部函数 包含时，内部函数会将外部函数的局部活动对象添加到自己的作用域链中。

而由于内部匿名函数的作用域链 在引用 外部包含函数的活动对象 ，即使 outFun 执行完毕了，它的活动对象还是不会被销毁！ 即，outerFun 的执行环境作用域链都销毁了，它的活动对象还在内存中留着呢。 并且根据垃圾回收机制，被另一个作用域引用的变量不会被回收。所以，除非内部的匿名函数解除对活动变量的引用（解除对匿名函数的引用），才可以释放内存。

所以，闭包会造成内存泄漏，就是因为它把外部的包含它的函数的活动对象也包含在自己的作用域链中了，会比一般的函数占用更多内存。

怎样解决

在退出函数之前，将不使用的局部变量赋值为 null 就可以了

1.20. 什么是内存泄漏，什么情况容易产生内存泄漏

不再用到的内存，没有及时释放，就叫做内存泄漏

内存泄漏的情况：

1. 在 js 中对 bom 与 dom 的引用没有及时清空
2. Js 中使用闭包的时候容易产生泄漏

1.21. 什么是内存溢出，都什么情况下容易产生内存溢出

当程序运行时需要的内存超过了剩余的内存时，就是抛出内存溢出的错误
内存溢出的情况：

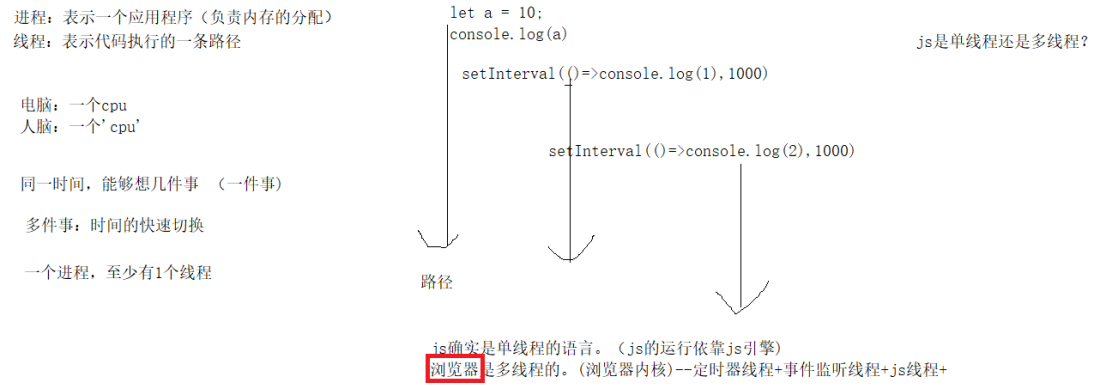
1. 死循环
2. 不合理的使用递归方法

1.22. 如何避免产生内存泄漏

1. 尽量避免使用闭包
2. 使用完的对象不用了尽量手动清除
3. 可以使用 es6 里面的 weakset 和 weakmap 来解决内存泄漏

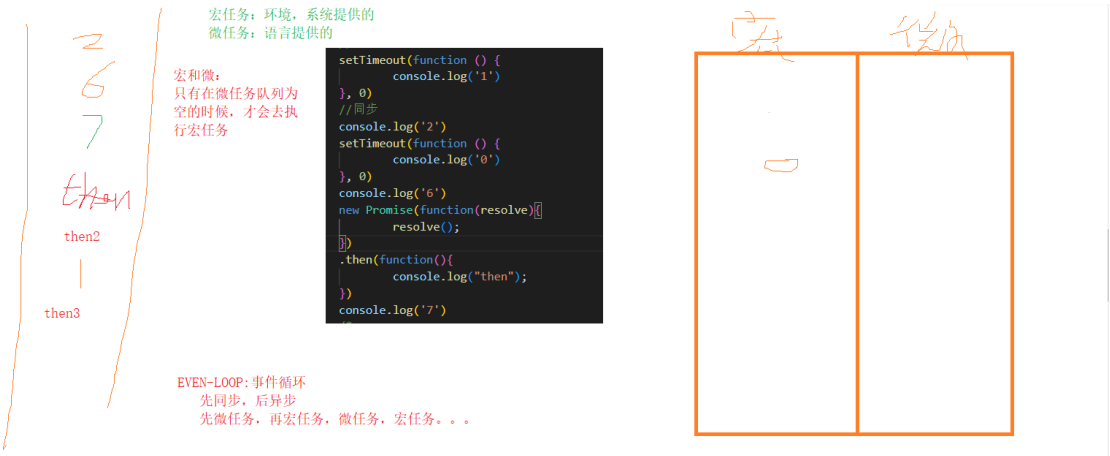
1.23. 进程和线程的区别是什么

进程：通常一个应用程序就是指一个进程。
线程：线程是表示进程中的一个执行路径。
多线程：一个进程中同时存在多个执行路径。
区别：线程是包含在进程里面的，一个进程至少包含一个线程。



1.24. 事件循环是什么意思？

事件循环 Event Loop
JavaScript 是一门单线程语言，异步操作都是放到事件循环队列里面，等待主执行栈来执行的，并没有专门的异步执行线程



1.25. 同步任务与异步任务

同步任务指的是，在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务；

异步任务指的是，不进入主线程、而进入"任务队列"（task queue）的任务，只有"任务队列"通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

1.26. 事件循环的执行过程

先执行全部的同步任务，然后执行全部的微任务，再执行 1 个宏任务，然后全部的微任务，再 1 个宏任务，如此循环执行任务，就是事件循环。

1.27. 宏任务与微任务的区别

在 JavaScript 中，异步任务被分为两种，一种宏任务（MacroTask），一种叫微任务（MicroTask）。

MacroTask（宏任务）

宿主环境提供的（浏览器和 node）是宏任务

`（window）setTimeout`、`setInterval`。

MicroTask（微任务）

语言标准提供的是微任务

`Promise`、`await`

1.28. 宏任务与微任务执行顺序：

- 执行栈在执行完**同步任务**后，查看**执行栈**是否为空，如果执行栈为空，就会去检查**微任务**队列是否为空，如果为空的话，就执行宏任务，否则就一次性执行完所有微任务。
- 每次单个**宏任务**执行完毕后，检查**微任务**队列是否为空，如果不为空的话，会按照**先入先**出的规则全部执行完**微任务**后，设置**微任务**队列为 `null`，然后再执行**宏任务**，如此循环。

总结：同步—>微任务—>宏任务

1.29. 常见异步任务类型

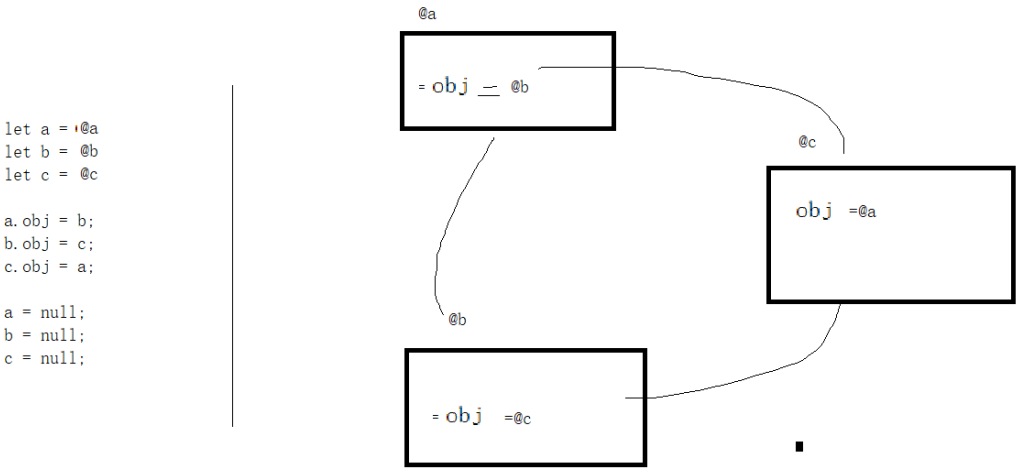
- settimeout 的回调函数放到宏任务队列里，等到执行栈清空以后执行
- promise 本身是**同步的立即执行函数**（同步）
- promise.then 里的回调函数会放到相应宏任务的**微任务**队列里，等宏任务里面的同步代码执行完再执行；

- async 函数表示函数里面可能会有异步方法，await 后面跟一个表达式，async 方法执行时，遇到 await 会立即执行表达式，然后把 await 表达式后面的代码放到微任务队列里，让出执行栈让同步代码先执行

1.30. JS 垃圾回收机制

问什么是垃圾

一般来说没有被引用的对象就是垃圾，就是要被清除，有个例外如果几个对象引用形成一个环，互相引用，但根访问不到它们，这几个对象也是垃圾，也要被清除。



垃圾回收算法

- 引用计数

早期的浏览器最常使用的垃圾回收方法叫做"引用计数"（reference counting）：语言引擎有一张"引用表"，保存了内存里面所有的资源（通常是各种值）的引用次数。如果一个值的引用次数是 0，就表示这个值不再用到了，因此可以将这块内存释放

优点：

引用计数为零时，发现垃圾立即回收；

最大限度减少程序暂停；

缺点：

无法回收循环引用的对象；

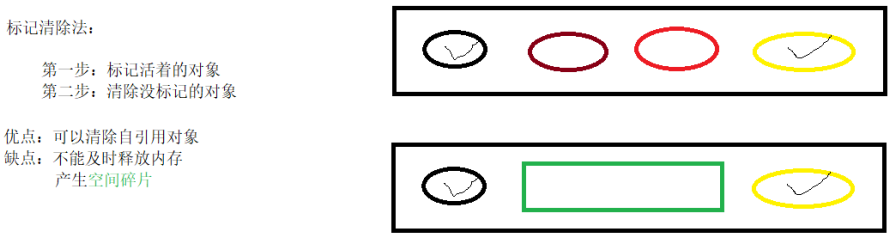
空间开销比较大；

- 标记清除

谷歌：v8 引擎使用最多的算法

1. 找到所有活动对象，做一个标记；
2. 遍历所有对象，找到没有标记对象；
3. 对没有标记的对象进行回收

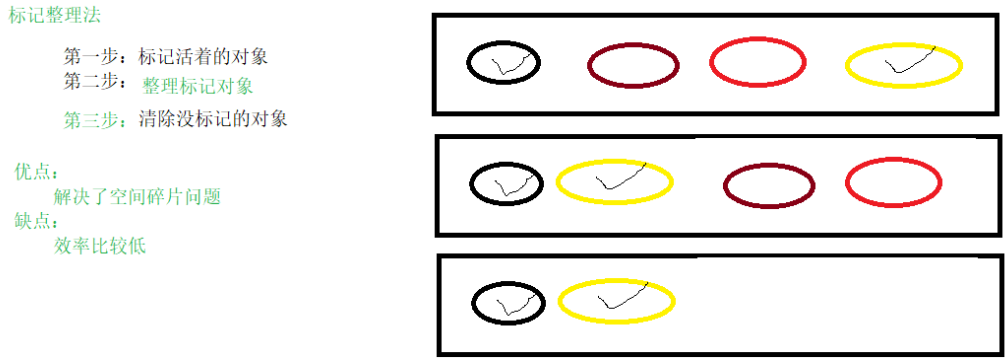
优点是可以回收循环引用的对象，缺点会产生空间碎片。



- 标记整理

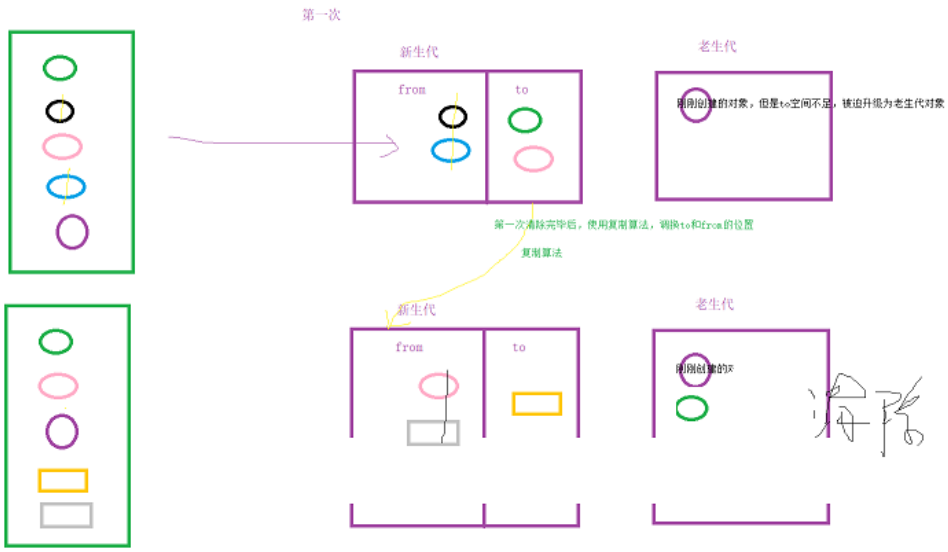
为了解决内存碎片化的问题，提高对内存的利用，引入了标记整理算法。标记整理可以看做是标记清除的增强。标记阶段的操作和标记清除一致。清除阶段会先执行整理，移动对象位置，将存活的对象移动到一边，然后再清理端边界外的内存。

标记整理的缺点是：移动对象位置，不会立即回收对象，回收的效率比较慢。



- 分代回收

针对不同对象采用不同算法：
（1）新生代：对象的存活时间较短。新生对象或只经过一次垃圾回收的对象。
（2）老生代：对象存活时间较长。经历过一次或多次垃圾回收的对象。



常见的浏览器分别用的是哪个算法

V8 擎垃圾回收策略

V8堆的空间等于新生代空间加上老生代空间。且针对不同的操作系统对空间做了内存的限制。

类型 \ 系统位数	64位	32位
老生代	1400MB	700MB
新生代	32MB	**700MB**

针对浏览器来说，这样的内存是足够使用的。限制内存的原因：

针对浏览器的GC机制，经过不断的测试，如果内存再设置大一点，GC回收的时间就会达到用户的感知，会造成感知上的卡顿。

对象晋升机制

一轮 GC 还存活的新生代需要晋升。

当对象从 From 空间复制到 To 空间时，若 To 空间使用超过 25%，则对象直接晋升到老生代中。设置为 25%的比例的原因是，当完成 Scavenge 回收后，To 空间将翻转成 From 空间，继续进行对象内存的分配。若占比过大，将影响后续内存分配。

回收新生代

回收新生代对象主要采用复制算法,将内存分为两个等大空间，使用空间为From，空闲空间为 To。检查 From 空间内的存活对象，若对象存活，检查对象是否符合晋升条件，若符合条件则晋升到老生代，否则将对象从 From 空间复制到 To 空间。若对象不存活，则**采用标记整理法**，释放不存活对象的空间。完成复制后，将 From 空间与 To 空间进行角色翻转。

回收老生代

回收老生代对象主要采用**标记清除**、**标记整理**、**增量标记**算法，主要使用**标记清除**算法，只有在内存分配不足时，采用**标记整理**算法。

1. 首先使用**标记清除**完成垃圾空间的**回收**；
2. 采用**标记整理**进行**空间优化**；
3. 采用**增量标记**进行**效率优化**；

增量标记

将本来可以一次标记完的过程分成多次标记，为了尽可能保证引用逻辑的顺利运行，减少卡顿。

2. 网络问题

2.1. api

同步
<a href="" get/>
<form action="#" method="get/post">
<script>open("http://www.baidu.com")
<script>location.href=""

异步
xmlHttpRequest
axios.get()
axios.post()

API（Application Programming Interface，应用程序接口）

2.2.接口管理（前后端协同）

普通文档：word （设置共享）
在线网页：swagger-ui （可以测试接口）
三方工具：apiPost（实时共享，测试接口，模拟接口 mock）

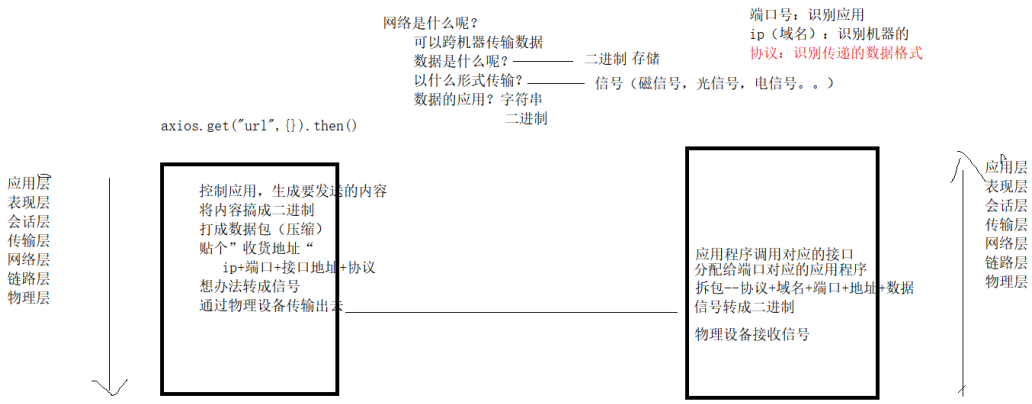
2.3.ui 交互

蓝湖

2.4.测试

bug 管理平台--
测试人员上传添加 bug
管理人员分配 bug
开发人员解决 bug
测试人员重新测试

2.5.概述：



2.6.网络七层模型(OSI)

物理层	负责信号的接收与发送
链路层	负责信号的转换，分包，校验等
网络层	进行逻辑地址寻址，实现不同网络之间的路径选择(判断 ip)
传输层	定义传输数据的协议端口号，以及流控和差错校验。
会话层	建立、管理、终止会话.对应主机进程，指本地主机与远程主机正在进行的会
表现层	数据的表示、安全、压缩。
应用层	网络服务与最终用户的一个接口

2.7.网络四层模型

应用层：	应用层，表现成，会话层
传输层：	
网络层：	
数据链路层：	数据链路层，物理层

2.8.参考网址，听说很详细

https://zhuanlan.zhihu.com/p/494612464

2.9.TCP/IP 是什么意思

是指能够在多个不同网络间实现信息传输的协议簇。TCP/IP 协议不仅仅指的是 TCP 和 IP

两个协议，而是指一个由 FTP、SMTP、TCP、UDP、IP 等协议构成的协议簇， 只是因为在 TCP/IP 协议中 TCP 协议和 IP 协议最具代表性，所以被称为 TCP/IP 协议

2.10. UDP 与 TCP

关于 udp

面向无连接，不可靠。有单播，多播，广播的功能
想发数据就可以开始发送了，并且也只是数据报文的搬运工，不会对数据报文进行任何拆分和拼接操作。

在发送端，应用层将数据传递给传输层的 UDP 协议，UDP 只会给数据增加一个 UDP 头标识下是 UDP 协议，然后就传递给网络层了

在接收端，网络层将数据传递给传输层，UDP 只去除 IP 报文头就传递给应用层，不会任何拼接操作

关于 tcp

面向连接的、可靠的、基于字节流的传输层通信协议。仅支持单播传输
建立连接需要经过三次握手

第一次握手：你在不在　　客户端进入准备状态，等待连接

第二次握手：我在呀　　服务端进入准备状态，等待连接

第三次握手：那我们链接呗　　客户端服务端建立连接

关闭连接需要经历四次挥手：

客户端进入断开状态

服务端进入断开装填，断开了客户端到服务端的连接通道

服务端进入断开装填

客户端进入断开装填，断开了服务端到客户端的连接通道

2.11. tcp 和 udp 的区别

	UDP	TCP
是否连接	无连接	面向连接
是否可靠	不可靠传输，不使用流量控制和拥塞控制	可靠传输，使用流量控制和拥塞控制
连接对象个数	支持一对一，一对多，多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
适用场景	适用于实时应用（IP电话、视频会议、直播等）	适用于要求可靠传输的应用，例如文件传输

TCP 与 UDP 区别总结：
1、TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接
2、TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付
3、TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流;UDP 是面向报文的 UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）
4、每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信
5、TCP 首部开销 20 字节;UDP 的首部开销小，只有 8 个字节 6、TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

2.12. TCP 与 HTTP 的关系是什么

TCP 协议在传输层，而 HTTP 协议对应于应用层。Http 协议是建立在 TCP 协议基础之上的。

2.13. http 请求的构成

2.14. 什么是长连接

在 HTTP/1.0 中默认使用短连接。也就是说，客户端和服务器每进行一次 HTTP 操作，就建立一次连接，任务结束就中断连接。 当客户端浏览器访问的某个 HTML 或其他类型的 Web 页中包含有其他的 Web 资源（如 JavaScript 文件、图像文件、CSS 文件等）， 每遇到这样一个 Web 资源，浏览器就会重新建立一个 HTTP 会话。 而从 HTTP/1.1 起，默认使用长连接，用以保持连接特性。使用长连接的 HTTP 协议，会在响应头加入这行代码：Connection:keep-alive 在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不会关闭，客户端再次访问这个服务器时 ，会继续使用这一条已经建立的连接。Keep-Alive 不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如 Apache）中设定这个 时间。实现长连接需要客户端和服务端都支持长连接。

2.15. http 的版本

1.0

属于瞬时连接，tcp 连接默认关闭

1.1

引入了持久连接，即 TCP 连接默认不关闭，可以被多个请求复用，不用声明 Connection: keep-alive。解决了 1.0 版本的 keepalive 问题，1.1 版本加入了持久连接，一个 TCP 连接可以允许多个 HTTP 请求

2.0

为了解决 1.1 版本利用率不高的问题，提出了 HTTP/2.0 版本。增加双工模式，即不仅客户端能够同时发送多个请求，服务端也能同时处理多个请求，解决了队头堵塞的问题

HTTP/3.0

HTTP 3.0 其实相较于 HTTP 2.0 要比 HTTP 2.0 相较于 HTTP 1.1 的变化来说小很多，最大的提升就在于效率，替换 TCP 协议为 UDP 协议，HTTP 3.0 具有更低的延迟，它的效率甚至要比 HTTP 1.1 快 3 倍以上

2.16. http 的常见的响应状态码

1 消息
- 100 Continue
- 101 Switching Protocols
- 102 Processing

2 成功
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 206 Partial Content
- 207 Multi-Status

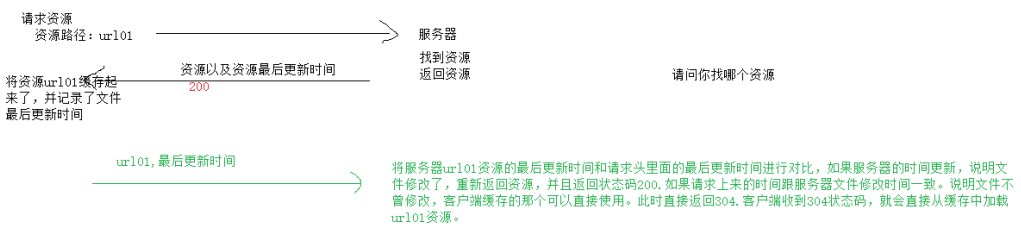
3 重定向
- 300 Multiple Choices
- 301 Moved Permanently
- 302 Move Temporarily
- 303 See Other
- 304 Not Modified
- 305 Use Proxy
- 306 Switch Proxy
- 307 Temporary Redirect

4 请求错误
- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout
- 409 Conflict
- 410 Gone
- 411 Length Required
- 412 Precondition Failed
- 413 Request Entity Too Large
- 414 Request-URI Too Long
- 415 Unsupported Media Type
- 416 Requested Range Not Satisfiable
- 417 Expectation Failed
- 418 I'm a teapot
- 421Misdirected Request
- 422 Unprocessable Entity
- 423 Locked
- 424 Failed Dependency
- 425 Too Early
- 426 Upgrade Required
- 449 Retry With
- 451 Unavailable For Legal Reasons

5 服务器错误
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported
- 506 Variant Also Negotiates
- 507 Insufficient Storage
- 509 Bandwidth Limit Exceeded
- 510 Not Extended
- 600 Unparseable Response Headers

200 表示请求成功
404 表示路径不存在
400 请求格式有误
500 表示服务器错误
302 表示重定向

304 表示数据未修改，使用缓存即可



2.17. http 请求的方式

常见的有 get，post，option。不常见的还有 put，delete，patch 等
get 和 post 主要是通过程序来发起的

option 是在满足一定条件的情况下，由浏览器自动发起的。主要用于跨域检查

Get 请求参数在地址栏可见，大小有限制
Post 请求参数在地址栏不可见，理论上大小不限制。

2.17.1. user-agent 是什么，有什么用?

他表示的是一个请求头，内容是当前浏览器信息。我们通过这些信息，识别用户用的是什么浏览器，甚至是什么设备，比如 pc，平板，android，ios。我们可以根据不同的设备，响应的不同的界面。（navigator.userAgent）

2.18. http 的请求的过程（浏览器输入地址敲回车）

域名解析 --> 发起 TCP 的 3 次握手 --> 建立 TCP 连接后发起 http 请求 --> 服务器响应 http 请求，浏览器得到 html 代码 --> 浏览器解析 html 代码，并请求 html 代码中的资源（如 js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

2.19. ip 协议

标志计算机的唯一地址
分为 ipv4 和 ipv6
v4 是由 4 个字节组成，编写的时候，分为 4 段，每段的长度都是 0 到 255
v6 是由 16 个字节组成，没用过

2.20. 域名是什么

一般情况下，我们的网络连接都是通过 ip 地址的，由于 ip 地址比较难记忆，所以便有了域名。域名其实本质上是 ip 地址的一种表现形式。相对来说更容易被用户识别，记忆。

2.21. 域名有哪些分类

常见的有：一级域名，二级域名，三级域名
Jd.com 一级域名
m.jd.com 二级域名
It.item.jd.com 三级域名
Ip.it.item.jd.com 四级域名

以上四个域名解析出来的 ip 地址跟 jd.com 是一样的。

2.22. Dns 是什么

它是域名解析服务器，记录了域名和 ip 地址的对应关系。在用户发起请求的时候，需要我们的 dns 服务器将域名解析成 ip 地址，进行寻址网络连接。

2.23. 常见的加密方式

Base64 加密
特点：别人得到密文，能够轻易破解
原理：3*8=24=4*6（6 位不足 8 位，高位补 0）
作用：迷惑作用
应用场景：
cookie 中保存中文
当 key value 中包含=号时，参数解析就会错误

```
Js:
加密：  window.btoa();
解密：  window.atob();
```

MD5

MD5 的全称是 Message-Digest Algorithm 5（信息-摘要算法）。128 位长度。目前 MD5 是一种不可逆算法。具有很高的安全性。它对应任何字符串都可以加密成一段唯一的固定长度的代码。（小贴士：为啥 MD5 加密算法不可逆呢~ 按道理来说有加密方式，就会有解密方式呀？因为 MD5 加密是有种有损的加密方式，比如一段数据为'123'，我在加密的时候，遇到 1 和 3 都直接当做是 a，加密后变成了'a2a'，所以解密的时候就出现了 4 种组合'323''121''123''321'，数据一多，自然找不到原始的数据了

```
引入一个 md5.js 文件，调用 Md5Utils.hex_md5("内容")
```

SHA1

SHA1 的全称是 Secure Hash Algorithm(安全哈希算法）。SHA1 基于 MD5，加密后的数据长度更长。它对长度小于 264 的输入，产生长度为 160bit 的散列值。比 MD5 多 32 位。因此，比 MD5 更加安全，但 SHA1 的运算速度就比 MD5 要慢了。使用方法和 MD5 其实是一样的

RSA

公钥加密，私钥解密

它是非对称加密,目前最重要的加密算法！计算机通信安全的基石，保证了加密数据不会被破解。你可以想象一下，信用卡交易被破解的后果。甲乙双方通讯，乙方生成公钥和私钥，甲方获取公钥，并对信息加密（公钥是公开的，任何人都可以获取），甲方用公钥对信息进行加密，此时加密后的信息只有私钥才可以破解，所以只要私钥不泄漏，就能保证信息的安全性。

AES

AES 加密为对称密钥加密，加密和解密都是用同一个解密规则，AES 加密过程是在一个 4×4 的字节矩阵上运作，这个矩阵又称为"状态(state)"，因为密钥和加密块要在矩阵上多次的迭代，置换，组合，所以对加密快和密钥的字节数都有一定的要求，AES 密钥长度的最少支持为 128、192、256，加密块分组长度 128 位。这种加密模式有一个最大弱点：甲方必须把加密规则告诉乙方，否则无法解密。保存和传递密钥，就成了最头疼的问题。

2.24. http 协议和 https 协议的区别

1、https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用。

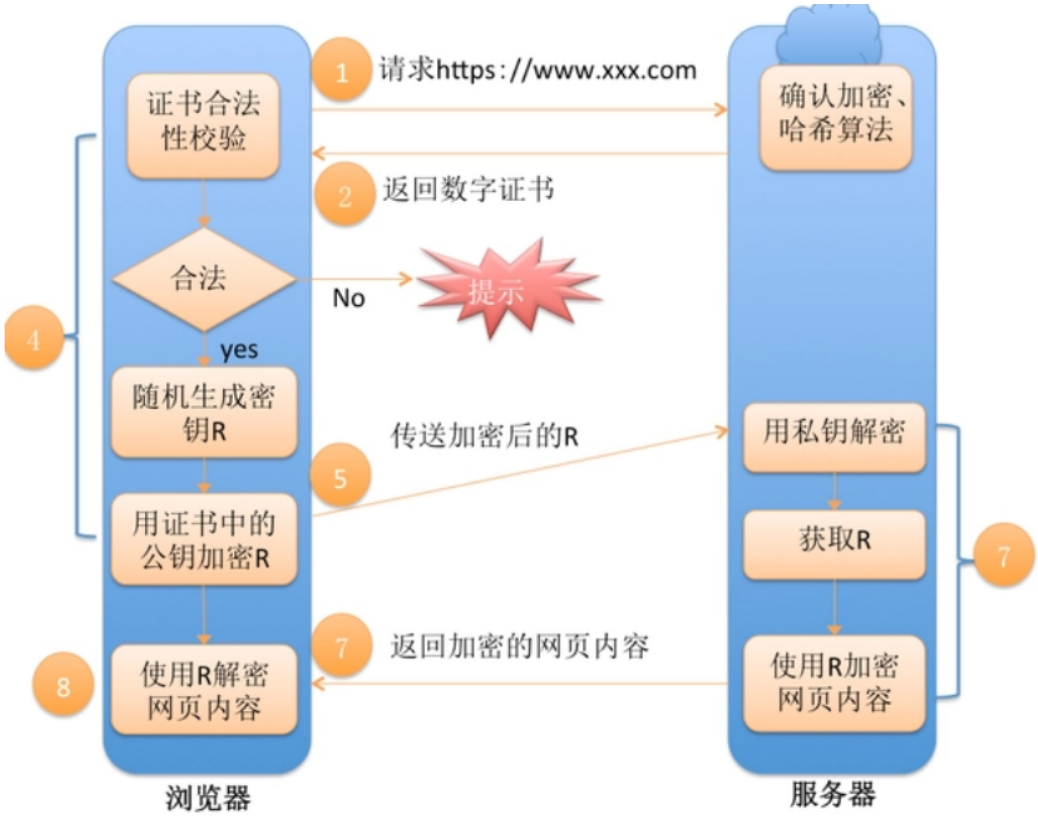
2、http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

3、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

4、http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

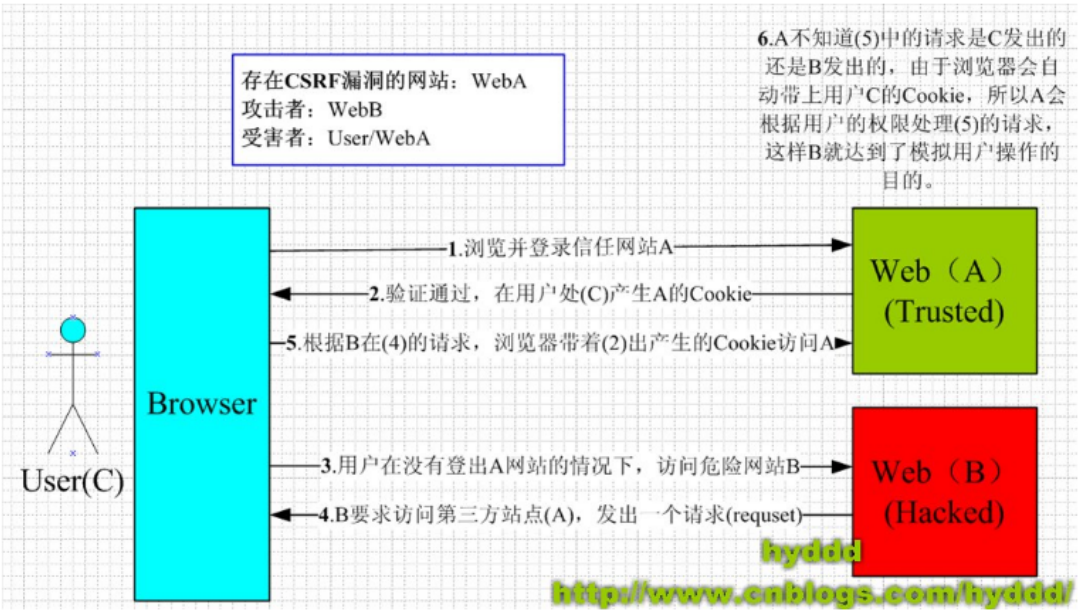
2.25. https 的请求过程

http 是明文传输，https 是密文传输



2.26. 网络攻击及解决办法

csrf:跨站请求伪造



CSRF 攻击防护

上文简单的叙述了 CSRF 攻击的原理，接下来将要介绍几种 CSRF 攻击的防护方法。

1. 只使用 JSON API

使用 JavaScript 发起 AJAX 请求是限制跨域的，并不能通过简单的 表单来发送 JSON，所以，通过只接收 JSON 可以很大可能避免 CSRF 攻击。

2. 验证 HTTP Referer 字段

根据 HTTP 协议，在 HTTP 头中有一个字段叫 Referer，它记录了该 HTTP 请求的来源地址。在通常情况下,访问一个安全受限页面的请求来自于同一个网站,比如上文中用户 User 想要在网站 WebA 中进行转账操作，那么用户 User

必须先登录 WabA
然后再通过点击页面上的按钮出发转账事件

这时该转帐请求的 Referer 值就会是转账按钮所在的页面的 URL，而如果黑客要对银行网站实施 CSRF 攻击，他只能在他自己的网站构造请求，当用户 User 通过黑客的网站发送请求到 WebA 时，该请求的 Referer 是指向黑客自己的网站。
因此，要防御 CSRF 攻击，网站 WebA 只需要对于每一个转账请求验证其 Referer 值，如果是以网站 WebA 的网址开头的域名，则说明该请求是来自 WebA 自己的请求，是合法的。如果 Referer 是其他网站的话，则有可能是黑客的 CSRF 攻击，拒绝该请求。

3. 在请求地址中添加 token 验证

CSRF 攻击之所以能够成功，是因为黑客可以完全伪造用户的请求，该请求中所有的用户验证信息都是存在于 cookie 中，因此黑客可以在不知道这些验证信息的情况下直接利用用户自己的 cookie 来通过安全验证。要抵御 CSRF，关键在于在请求中放入黑客所不能伪造的信息，并且该信息不存在于 cookie 之中。可以在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token，如果请求中没有 token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求。
这种方法要比检查 Referer 要安全一些，token 可以在用户登陆后产生并放于 session 之中，然后在每次请求时把 token 从 session 中拿出，与请求中的 token 进行比对。

xss:跨站脚本攻击
　　通过技术手段，在网页上添加额外的代码

sql 注入：
　　通过 sql 拼接的方式改变原有的 sql 结构，达到不可告人的目的。

2.27. 地狱回调

什么是地狱回调
　　函数作为参数层层嵌套

```
let m1 = (m2)=>{
    m2();
}
let m2 = (m3)=>{
    m3();
}
m1(m2((f1,f2)=>{
    f1(f2)
}))
```

地域回调
　　代码的层次很乱
　　维护难度大
　　逻辑混乱

解决地狱回调
　　保持你的代码简短(给函数取有意义的名字,见名知意,而非匿名函数,写成一大坨)

模块化(函数封装,打包，每个功能独立,可以单独的定义一个 js 文件 Vue,react 中通过 import 导入就是一种体现)

使用 promise 来进行解决

2.28. promise

promise 是一个状态对象，默认是等待状态，当转为成功状态，或者失败状态，就回调 then 里面的对应方法。从而达到控制异步程序执行顺序的目的。

2.28.1. Promise 有几种状态

3 中状态，分别是等待状态 pedding,成功状态 resovle，失败状态 reject

2.28.2. promise.all 作用

Promise.all 可以将多个 Promise 实例包装成一个新的 Promise 实例。同时，成功和失败的返回值是不同的，成功的时候返回的是一个结果数组，而失败的时候则返回最先被 reject 失败状态的值

2.28.3. async await 和 promise 和 generator 有什么区别

这几种都可以解决异步操作的地狱回调问题

Async 是建立在 promise 之上的。Async 修饰方法，返回的就是 promise 对象。

await 只能在 async 中使用，await 是阻塞的意思，就是暂停，你一起调用 2 个接口，第一个执行完，不输出结果，要等最第二个接口执行完，才返回这两个的结果。 是属于将异步操作转化为同步操作的做法

async await 和 generator： async 是 Generator 的语法糖 ，也就是说，async 是对 generator 的实现，而 generator 代码更复杂。 generator 除了能解决回调地域问题，还可以作为迭代器使用

2.28.4. 捕获 promise 的错误

```
new Promise(function(resolve,reject){
    try{
                aa();
        resolve(3)
    }catch(e){
        reject(e)
    }
})
.then(
(data)=>console.log("success",data.reduce((total,c)=>total+c,0))
// (data)=>console.log("fail->",data)
 )
 .catch(err=>{
    console.log("程序报错，plan b",err)
```

```
})
```

2.29. 图片/文件夹上传到后台是什么类型?

图片上传后台有三种格式:

file 格式 （创建 formData 来完成 file 上传）<input type="file" id="imgfile" accept="image/jpeg, image/png, image/jpg" >

base64 格式

```
<input type="file" id="imgfile">
var base64Pic = ‘ ‘
document.getElementById(‘imgfile‘).onchange = function(){
      var fileReader = new FileReader();
      fileReader.readAsDataURL(this.files[0]);
      fileReader.onload = function(){
        base64Pic = fileReader.result;
        console.log(base64Pic )//base64 可以直接放在 src 上 预览
      }
```

2.30. 跨域

2.30.1. 什么是跨域

概念

跨域是指在 Web 应用中,一个网站或应用访问另一个网站或应用所在的域之外的资源。由于浏览器的同源策略，一般情况下同一个域中的网站或应用可以互相访问资源，但跨域访问会被浏览器拒绝。

什么情况下产生跨域

跨协议，	(http://www.baidu.com==>https://www.baidu.com)
跨 ip(域名)	(http://www.baidu.com==>http://www.jd.com)
跨端口	(http://localhsot:8080 ==> http://localhost:9090)

跨域出现的问题

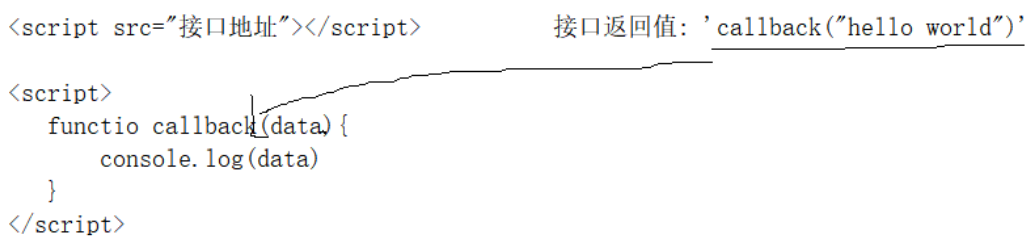
无法读取非同源网页的 Cookie、LocalStorage 和 IndexedDB

无法接触非同源网页的 DOM
无法向非同源地址发送 AJAX 请求

怎么解决跨域

jsonp:

网页通过添加一个<script>元素，向服务器请求 JSON 数据，服务器收到请求后，将数据放在一个指定名字的回调函数的参数位置传回来。



CORS:后台设置允许跨域

如果要带 cookie，前端也要设置一下

代理服务器

proxy 设置代理服务器即可

小程序解决跨域

在小程序的开发设置里面，有一个域名设置

2.30.2. Jsonp 的原理

1. 提前定义好回调的函数，做好数据的处理逻辑
2. 通过创建 script 标签，发起请求，加载 js 代码
3. 回调定义好的函数

3. es6

3.1. es6 新特性有哪些

let，var，const

解构赋值

set：不重复

map：双列结构

新的数组方法 filter,find,map,reduce

模块化--import export

类 class

promise

模板字符串 `sf`

箭头函数

延展操作符 ...

3.2.暂时性死区

概念

当程序的控制流程在新的作用域（module function 或 block 作用域）进行实例化时，在此作用域中用 let/const 声明的变量会先在作用域中被创建出来，但因此时还未进行词法绑定，所以是不能被访问的，如果访问就会抛出错误。因此，在这运行流程进入作用域创建变量，到变量可以被访问之间的这一段时间，就称之为暂时死区。

容易产生暂时性死区的情况

let/const 关键字未出现之前，typeof 运算符是百分之百安全的，现在也会引发暂时性死区的发生，像 import 关键字引入公共模块、使用 new class 创建类的方式，也会引发暂时性死区，究其原因还是变量的声明先与使用

解决暂时性死区

养成良好的变成习惯，变量的使用一定要在声明时候使用，否则就会引发‘暂时性死区’

3.3.es7,8,9,10,11,12 的新特性

https://blog.csdn.net/qq_20173195/article/details/127117264

比如说：

es7 里面的 includes 方法，可以查询数组中是否包含指定的值

比如说：

es8 里面的 async,await 控制异步顺序，Object.keys,Object.values 一次性获取对象的属性或值

比如说：

es10 里面的 flat 方法，可以实现数组的扁平化

3.4.map 和 foreach 的区别是什么

map 拥有返回值，在 react 中用来循环，非常常用。

foreach 没有返回值，主要用于遍历数组。

4.cs3

4.1. CSS3 的新特性

RGBA 和透明度

background-image

background-origin(content-box/padding-box/border-box)

background-size background-repeat

word-wrap (对长的不可分割单词换行) word-wrap：break-word
文字阴影：text-shadow： 5px 5px 5px #FF0000;（水平阴影，垂直阴影，模糊距离，阴影颜色）
font-face 属性：定义自己的字体
圆角（边框半径）：border-radius 属性用于创建圆角
盒阴影：box-shadow: 10px 10px 5px #888888
媒体查询：定义两套 css，当浏览器的尺寸变化时会采用不同的属性
弹性布局：display:flex
动画：@keyframes
过渡：transition
2d 效果：transform(translate/scale/sknew)

4.2. CS3 新增哪些伪元素

:first-of-type 选择属于其父元素的首个元素
:last-of-type 选择属于其父元素的最后元素
:only-of-type 选择属于其父元素唯一的元素
:only-child 选择属于其父元素的唯一子元素
:nth-child(2) 选择属于其父元素的第二个子元素
:enabled :disabled 表单控件的禁用状态。
:checked 单选框或复选框被选中。

4.3.请解释一下 CSS3 的 flexbox（弹性盒布局模型）,以及适用场景?

传统的 block 布局和 inline 布局,比较有局限性。比如 block 默认是从上向下排列,inline 是从左往右排列。

flex 布局，相对更加灵活，没有方向限制，开发人员可以根据需求自由设置。能够更好的兼容移动端开发，在 android 和 ios 上也完美支持。

它可以通过 flex:1 来控制份数，可以通过 flex-direction 来控制方向，通过 flex-wrap 控制换行，通过 justify-content 控制主轴对齐方式等等。

它主要用在移动端开发以及响应式界面开发上面。能够更好的完成适配效果。

4.4. Flex 布局常用的属性有哪些

flex-direction 属性决定主轴的方向
flex-wrap 属性决定项目在一行排不下的情况下是否换行
flex-flow 是 flex-direction 属性和 flex-wrap 属性的简写形式，默认值为 row nowrap。
justify-content 定义了项目在主轴上的对齐方式
align-items 每行项目在侧轴方向上的对齐方式
align-content 定义了容器在侧轴方向上有额外空间时怎样排布子容器
order 定义项目的排列顺序。数值越小排列越靠前，默觉得 0，可能的值为随意整数。
flex-grow 定义项目的放大比例，默觉得 0，即假设存在剩余空间也不放大。
flex-shrink 属性定义了项目的缩小比例，默觉得 1。即假设空间不足该项目将缩小。
flex-basis 属性定义了在分配多余空间之前，项目占领的主轴空间（main-size）。

flex 属性 flex 属性是 flex-grow, flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto。
align-self 同意单个项目有与其它项目不一样的侧轴对齐方式，可覆盖 align-items 属性。

5.html5

5.1. 版本

html的版本有:

1. HTML1.0
2. HTML2.0
3. HTML3.2
4. HTML4.0
5. HTML4.01（微小改进）
6. HTML5：2008年正式发布,现在都在用第5版的html

5.2.有哪些新特性

新标签 section ,header ,footer,nav, aside

表单校验:

canvas 画布

video 视频

audio 音频

localStorage 本地存储，永久存储

sessionStorage 本地存储，当会话结束，存储内容失效

cookie 缓存，服务器可以操作，浏览器会自动带 cookie 数据，存储不了中文

session 服务器的缓存

5.3.localStorage 如何设置过期时间

在保存数据的时候，以对象的形式保存，除了原有数据，添加额外的保存时间。比如我们要保存 name:jack,实际保存{name:"jack",createTime:'保存数据的那个时间',maxAge:"30"},
当使用这个数据的时候，先获取创建时间与当前时间的事件差，再对比 maxAge，判断是否过期，如果过期，手动清楚数据，否则，正常调用即可。

5.4.cookie 和 localStorage 以及 sessionStorage 的区别

他们都是可以在浏览器上面用来缓存数据。都是 k-v 结构
Cookie 可以设置过期时间，可以由浏览器自动发给服务器
localStorage 和 sessionStorage 都是 h5 新增的 api
sessionStorage 保存的数据在浏览器关闭后，会自动清楚数据
localStorage 没办法设置过期时间，需要手动清除数据

Cookie 只能保存字符串，另外两个可以保存对象

6. TypeScript

6.1. Typescript 是什么? 怎么用?

ts 是开源和跨平台的编程语言。它是 js 的一个超集，为 js 添加了可选的静态类型和基于类的面向对象编程

在创建项目的时候，一般可以直接勾选 ts，然后就可以直接使用了。项目打包的时候，通过插件，会将它编译成 js 的代码去运行

6.2.主要特性

基于类的面相对象编程
静态类型检查

7. Sass

7.1. sass 是什么? 怎么用?

sass 是 css 的一个超集。它拓展了 css 的功能。在项目中，创建过程中，也是可以直接勾选。然后在组件上，vue 项目 style 标签上添加 lang=scss 即可。react 项目，css 文件的后缀改成 scss 即可。小程序中可以勾选 sass 的模板即可使用。

8.nodejs

8.1. node.js 怎么创建服务器

```
cnpm i express --save
//引入 express 模块
const express = require("express")

//通过这个模块来创建服务器
const app = express();

//启动监听端口
app.listen(8888)
```

8.2.nodejs 怎么连接数据库

```
cnpm i mysql
//引入 mysql 数据库驱动
var mysql    = require('mysql');
//设置连接的参数
let conn = mysql.createConnection({
            user:"sun",
```

```
        password:"sun",
        host:"localhost",
        database:"k4"
})

//进行连接
conn.connect();

//执行 sql 语句--更新
conn.query(sql,function(error,result,field){})

//关闭连接通道，释放内存
conn.end();
```

8.3.手动写出冒泡排序

```
function bSort(arr) {
   var len = arr.length;
   for (var i = 0; i < len-1; i++) {
      for (var j = 0; j < len - 1 - i; j++) {
            //  相邻元素两两对比，元素交换，大的元素交换到后面
          if (arr[j] > arr[j + 1]) {
              var temp = arr[j];
              arr[j] = arr[j+1];
              arr[j+1] = temp;
          }
      }
   }
   return arr;}
//举个数组
myArr = [20,18,27,19,35];//使用函数 bSort(myArr)
```

9.移动端

9.1. viewport 的理解

viewport 表示视口

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0, max-scale=1.0, min-scale=1.0">
```

```
因为考虑到移动设备的分辨率相对于桌面电脑来说都比较小，所以为了能在移动设备上正常显示那些传统的为桌面浏览器设计的网站，移动设备上的浏览器都会把自己默认的 viewport 设为 980px 或 1024px
        视口-肉眼能够看见的区域
```

移动端设计稿：常见的是 750px 因为我们移动端都是参照 375 来设计的

9.2. rpx、px、em、rem、%、vh、vw 的区别是什么

rpx 相当于把屏幕宽度分为 750 份，1 份就是 1rpx

px 绝对单位，页面按精确像素展示

em 相对单位，相对于它的本身节点字体大小进行计算

rem 相对单位，相对根节点 html 的字体大小来计算

% 一般来说就是相对于父元素

vh 视窗高度，1vh 等于视窗高度的 1%

vw 视窗宽度，1vw 等于视窗宽度的 1%

9.3.css 优化方式

css 单一样式
避免使用通配符选择器(*)
尽量避免标签选择，而是用 class
能够使用继承的，尽量避免重复定义
属性值为 0 时，不加单位
属性值为小数的时候，可以省略整数部分的 0
使用雪碧图，字体图标
抽离公共样式，灵活复用
样式和内容分离
尽量使用 link，避免使用@import

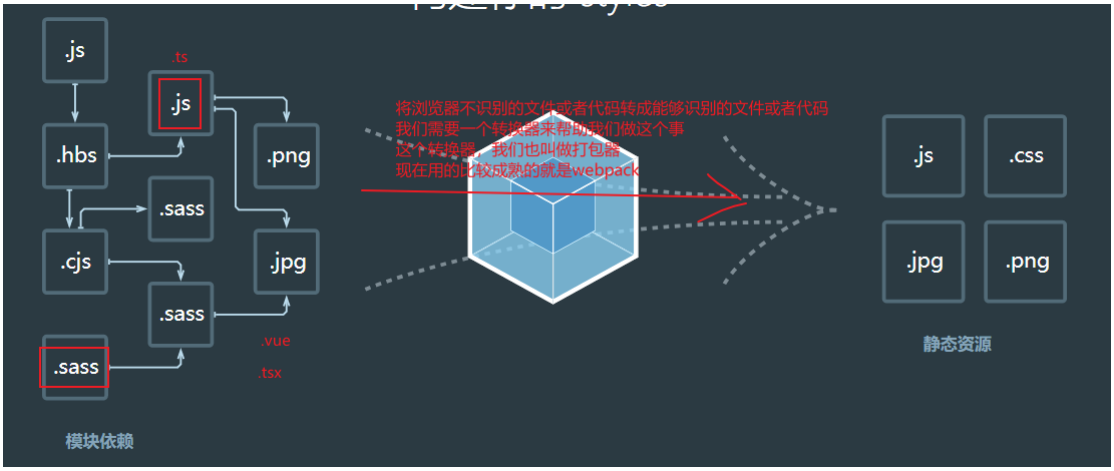
9.4. Js 优化方式

1.避免使用全局变量
2.减少判断层级
3.减少数据读取次数，对于频繁使用的数据，我们要对数据进行缓存。
4.减少循环体中的活动
5.事件绑定优化
6.避开闭包陷阱

10. webpack

10.1. 概念

webpakc 就是一个打包器，可以将我们项目中的各种资源打包成可运行的浏览器资源



基本的使用，通过命令打包
npx webpack ./index.js　　默认打包 index.js 文件到 dist/main.js 中

通过配置文件使用
在根目录创建 webpack.config.js 文件

在 package.json 中的 script 中配置脚本
“serve”:　“npx webpack”

启动　npm run serve

装一个实时更新的插件，我们选择 webpack-dev-server.提供了 web 服务以及实时更新

```
npm install --save-dev webpack-dev-server
```

配置　webpack.config.js

```
devServer: {
    static: './dist'
  },
```

在 script 中配置启动脚本

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "serve": "npx webpack",
  "start": "webpack serve --open"
},
```

10.2. 配置 html 打包插件

安装
npm install --save-dev html-webpack-plugin

配置

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
```

```
    plugins: [
     //打包一个 html 文件加载到服务器的内存中供调用,并且该内存 html 文件自动引用了
main.js 文件
      new HtmlWebpackPlugin({
        title: 'production',
        template: "./src/index.html"
      }),
    ],
```

10.3. 提供的 html

```
<h1>这里是 index.html</h1>
```

nginx--服务器软件

webpack 的打包原理

webpack 打包优化