1. 闭包问题

闭包是在某个作用域内定义的函数，它可以访问这个作用域内的所有变量。闭包作用域通常包括三个部分：1.函数本身作用域；2.闭包定义时的作用域；3.全局作用域

闭包的常见用途：1.创建特权方法用于访问控制；2.事件处理程序及回调

2. 请求方式（get，post）

表面上的区别：GET 在浏览器回退时是无害的，而 POST 会再次提交请求

GET 产生的 URL 地址可以被 Bookmark，而 POST 不可以

GET 请求会被浏览器主动 cache，而 POST 不会，除非手动设置

GET 请求只能进行 url 编码，而 POST 支持多种编码方式

GET 请求参数会被完整保留在浏览器历史记录里，而 POST 中的参数不会被保留

GET 请求在 URL 中传送的参数是有长度限制的，而 POST 么有

对参数的数据类型，GET 只接受 ASCII 字符，而 POST 没有限制

GET 比 POST 更不安全，因为参数直接暴露在 URL 上，所以不能用来传递敏感信息

GET 参数通过 URL 传递，POST 放在 Request body 中

实际上：本质都是 TCP 链接，并无差别，只是语义上的差别，由于 HTTP 的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。Get 请求产生一个 tcp 数据包（header 和 data 一次性发送出去，服务器响应 200），post 请求产生两个 tcp 数据包（先发送 header，服务器响应 100 continue，在发送 data，服务器响应 200）

3. 基本数据类型（typeof null）

null,underfined,number,boolean,string

null==undefined(true)

Typeof(null)==object

4. 数据存储 cookie，localStorage 和 sessionStorage

存储数据大小，声明周期，与服务器端通信。

5. 向后台提交数据的方式有哪几种 ?ajax 和 form 表单提交的区别

Ajax 异步进行，网页不需要刷新，form 表单提交的时候是新建一个页面，需要刷新

Ajax 提交时是在后台新建一个请求，form 鸟蛋是放弃本页面，而后再请求

Ajax 必须使用 js 来实现，form 表单提交是浏览器的功能

Ajax 在提交，请求以及接收的时候都需要对数据进行处理，form 表单提交是根据表单

结构自动完成

6. 跨域问题

同源：两个文档同源需满足协议相同、域名相同、端口相同

跨域通信方法：1.简单的单项通信，可以新建<img>,<script>,<link>,<iframe>元素，通过

src，href 属性设置为目标 url；2.如果请求 json 数据，使用<script>进行 jsonp 请求；3.内部服务器代理请求跨域 url，然后返回数据；4.服务器添加允许跨域请求操作

7. es6 语法常用

promise：简单来说就是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）。从语法上来说，promise 是一个对象，从它可以获取异步操作的消息。Promise 对象有以下两个特点：1.对象的状态不受外界影响；2.一旦状态改变，就不会再变，任何时候都可以得到这个结果。下面是一个 promise 实例：

```
    const promise = new Promise(function(resolve, reject) {
  // ... some code

  if (/*  异步操作成功  */){
    resolve(value);
  } else {
    reject(error);
  }
});
```

Promise.prototype.then()：定义在原型对象 Promise.prototype 上的。它的作用是为 promise 实例添加状态改变时的回调函数。then 方法的第一个参数是 resolved 状态的回调函数，第二个参数（可选）是 rejected 状态的回调函数；返回的是一个新的 promise 实例。

Promise.prototype.catch():方式是.then(null,rejection)或.then(undefined,rejection)的别名，用于指定发生错误时的回调函数。

Promise.prototype.finally():指定不管 promise 对象最后状态如何，都会执行的操作。

Promise.all():用于将多个 promise 实例，包装成一个新的 promise 实例

```
const p1 = new Promise((resolve, reject) => {
  resolve('hello');
})
.then(result => result);

const p2 = new Promise((resolve, reject) => {
  throw new Error('报错了');
})
.then(result => result);

Promise.all([p1, p2])
.then(result => console.log(result))
.catch(e => console.log(e));
// Error: 报错了
```

Promise.race():

```
const p = Promise.race([p1, p2, p3]);
```

上面代码中，只要 p1、p2、p3 之中有一个实例率先改变状态，p 的状态就跟着改变。那个率先改变的 Promise 实例的返回值，就传递给 p 的回调函数。

Promise.resolve():有时需要将现有对象转为 Promise 对象。该方法的参数分成四种情况：（1）参数是一个 promise 实例；（2）参数是一个具有 then 方法的对象；（3）参数不是具有 then 方法的对象，或根本就不是对象；（4）不带有任何参数

Promise.resolve():Promise.reject(reason)方法也会返回一个新的 Promise 实例，该实例的状态为 rejected

Promise.try():为所有的操作提供了统一的处理机制

async

8. 冒泡排序和快速排序以及时间复杂度

冒泡排序：比较相邻元素，第一个>第二个，则交换；对每一相邻元素做同样工作，直到最后一个

```
// 冒泡排序
```

```
bubblesortArr(arr){
for(var i=0;i<arr.length-1;i++){ //排序趟数  注意是小于
for(var j=0;j<arr.length-i-1;j++){
//一趟确认一个数，数组长度减当前趟数就是剩下未确认的数需要比较的次数
//因为 j 从 0 开始，所以还要再减 1，或者理解为 arr.length-(i+1)
if(arr[j]>arr[j+1]){
var temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
}
}
}
},
```

快速排序：在需要排序的数中随便找出一个，比它小的放左边，大的放右边，一样大的放一起。左右两边各种重复上述过程，直到两边只剩下一个或者另个数位置。快速排序的最坏运行情况是 O(n²)，比如说顺序数列的快排。但它的平摊期望时间是 O(n log n) ，且 O(n log n) 记号中隐含的常数因子很小，比复杂度稳定等于 O(n log n)的归并排序要小很多。所以，对绝大多数顺序性较弱的随机数列而言，快速排序总是优于归并排序。

```
quickSort(arr){
//如果数组<=1,则直接返回
if(arr.length<=1){return arr;}
var pivotIndex=Math.floor(arr.length/2);
//找基准，并把基准从原数组删除
var pivot=arr.splice(pivotIndex,1)[0];
//定义左右数组
var left=[];
var right=[];

//比基准小的放在 left，比基准大的放在 right
for(var i=0;i<arr.length;i++){
if(arr[i]<=pivot){
left.push(arr[i]);
}
else{
right.push(arr[i]);
}
}
//递归
return this.quickSort(left).concat([pivot],this.quickSort(right));
```

```
}
},
```

选择排序：在时间复杂度上表现最稳定的排序算法之一，因为无论什么数据进去都是 O(n²)的时间复杂度。。。所以用到它的时候，数据规模越小越好。唯一的好处可能就是不占用额外的内存空间了吧。

思想：不断寻找最小的数，按照最小数的索引保存

插入排序：扑克牌式，按照顺序临近的进行比较

希尔排序：插入排序的更高效率实现，它与插入排序的不同之处在于，它会优先比较距离较远的元素。希尔排序的核心在于间隔序列的设定。既可以提前设定好间隔序列，也可以动态的定义间隔序列。

等等其他排序...

9. 面向对象的三个方法以及实现的方式

1. 封装：将属性私有化，不允许外部数据直接访问，并设置相应的方法对属性进行设置和读取，从而实现对属性访问的限制
2. 继承：将多个类共用的属性和方法写在父类里，子类使用 extends 关键字继承父类，就可以使用父类非私有化的属性和方法。优点：提高代码复用性，维护性，将类与类关联起来
3. 多态：同一种事物，因为给定条件不同，展示不同的结果。同一个引用类型，使用不同的实例，产生不同的结果。实现条件：子类继承父类，子类重写父类方法，父类引用只想子类对象

10. git，svn 相关知识

git 是分布式的，svn 是集中式的，必须有一个服务器版本库作为中央服务器。

Git 把内容按元数据方式存储，而 svn 是按文件

Git 的内容完整性要优于 svn

分支不同，一个 svn 项目创建了几个分支，就相当于把源码复制拷贝了多少次。而 git 项目创建几个分支，仅仅是多了几个索引，占用很小的空间

Svn 的特点是简单，集中放置代码。Git 的特点是版本控制可以不依赖网络做任何事情，对分支和合并有更好的支持

11.http 协议相关，缓存机制等等

1. 从浏览器地址栏输入 url 到显示页面的步骤(以 HTTP 为例)

    a. 浏览器查看是否有缓存，如有缓存并未过期则直接转码

    b. 解析 url 获取协议，主机，端口，path，生成 http 请求报文

    c. 浏览器获取主机 ip 地址并且建立 TCP 链接（三次握手），成功后发送请求

    d. 服务器接收请求并解析，并准备响应。之后将响应报文通过 TCP 链接发送会浏览器。浏览器视情况决定关闭 tcp 链接(四次挥手)或者保留重用

    e. 浏览器相应状态码，可缓存的进行缓存，然后进行接码操作

    f. 解析 html 文档，构建 dom 树，下载资源，构建 css，执行 js 脚本（无严格先后顺序）

2. http 请求报文

    a. 首行：请求 url，请求方法，协议版本，CRLF（回车换行）

    b. 请求头：general-header（通用头部），request-header（响应头部）或者 entity-header（实体头部），每个一行以 CRLF 结束

c. 消息实体

3.HTTP response 报文结构

a. 首行：HTTP 版本,状态码，状态描述，CRLF（回车换行）

b. 响应头：通用头部，响应头部，实体头部，CRLF 结束

c. 响应实体

12. ajax 实现原理

ajax 是一种异步请求数据的 web 开发技术，在不需要重新刷新页面的情况下，通过异步请求加载后台数据。提高用户体验，较少的网络数据的传输量。核心是 XMLhttprequest 对象

使用：

1. 创建 ajax 核心对象 XMLhttprequest
2. 创建一个新的 HTTP 请求并指定该请求的方法、url 及验证信息
3. 设置响应 HTTP 请求状态变化的函数，并发送 HTTP 请求
4. 获取异步调用返回的数据
5. 使用 javascript 和 dom 实现局部刷新

```
var xhr =null;//创建对象

if(window.XMLHttpRequest){xhr = new XMLHttpRequest();}

else{xhr = new ActiveXObject("Microsoft.XMLHTTP");}
```

```
xhr.open(“方式”,”地址”,”标志位”);//初始化请求

xhr.setRequestHeader(“”,””);//设置 http 头信息

xhr.onreadystatechange =function(){}//指定回调函数

 xhr.send();//发送请求
```

缺点：不支持浏览器 back 按钮；暴露了与服务器交互的细节；对搜索引擎的支持比较弱；破坏了程序的异常机制

13.为什么 js 单线程可以实现异步多线程（js 异步机制）

Js 单线程：执行顺序从上到下依次执行，同一段时间内只有一段代码被执行。

虽然 js 是单线程的但是浏览器内部不是单线程的，一些 I/O 操作、定时器的计时和事件监听等都是由浏览器提供的其他线程来完成的

任务队列（消息队列）：将所有任务分为两种：同步任务和异步任务。同步任务是指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”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.

异步执行运行机制（event loop）：1.所有同步任务都在主线程上执行，形成一个执行栈；2.主线程之外，还存在一个“任务队列”。只要异步任务有了运行结果，就在“任务队列”之中放置一个事件；3.一旦“执行栈”中的所有同步任务执行完毕，系统就会读取“任务队列”，那些对应的异步任务于是就结束等待状态，进入执行栈，开始执行；4.主线程不断重复以上三步。

主线程从"任务队列"中读取事件，这个过程是循环不断的，所以整个的这种运行机制又称为 Event Loop（事件循环）。只要主线程空了，就会去读取"任务队列"，这就是

JavaScript 的运行机制。

那些语句会放入异步任务队列及放入时机：1.setTimeOut 和 setInterval；2.DOM 事件；3.ES6 中的 Promise；4.Ajax 异步请求。

异步循环里面设置定时器输出结果都是循环的最后一个值，如果需要循环输出的话可以怎么修改：1.将 var 变为 let；2.加个立即执行函数；3.通过闭包的方式

14. Js 的继承是怎么实现的，原型链的实现方式（待续）

1. 原型链继承
2. 借用构造函数继承
3. 组合继承（原型链+构造函数）
4. 原型式继承
5. 寄生式继承
6. 寄生组合式继承

原型：每个对象（除 null 外）都有另一个对象与之关联，这个另一个对象便称之为“原型”（prototype），每个对象都是从原型继承属性。原型链是指一系列链接的原型对象的链称为“原型链”

15. websocket 的机制

基于 HTTP 协议，服务器可以主动向客户端推送信息，客户端也可以主动向服务器发送信息，是真正的双向平等对话。

Ajax 轮询：让浏览器隔几秒就发送一次请求，询问服务器是否有新信息

long poll：轮询方式，阻塞模型（客户端发起连接后，如果没消息，就一直不返回 Response 给客户端。直到有消息才返回，返回完之后，客户端再次建立连接，周而复始）

16. promise 的常用方法

17.怎么让两个异步的事件哪个先完成先用哪个

promise.race()

18.vue-router 的实现机制（前端路由）

两种方式：1.利用 URL 中的 hash（#）;2.利用 History interface 在 HTML5 中新增的方法

根据 MDN 的介绍，调用 history.pushState()相比于直接修改 hash 主要有以下优势：

1.pushState 设置的新 URL 可以是与当前 URL 同源的任意 URL；而 hash 只可修改#后面的部分，故只可设置与当前同文档的 URL

2. pushState 设置的新 URL 可以与当前 URL 一模一样，这样也会把记录添加到栈中；而 hash 设置的新值必须与原来不一样才会触发记录添加到栈中

3. pushState 通过 stateObject 可以添加任意类型的数据到记录中；而 hash 只可添加短字符串

4. pushState 可额外设置 title 属性供后续使用

19.线程和进程的区别

进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径，线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉。所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。一个程序至少有一个进程，一个进程至少有一个线程。

线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源的管理和保护；而进程正相反。同时，线程适合于在 SMP 机器上运行，而进程则可以跨机器迁移。

20. 了解的 js 框架以及框架的特点

Vue.js：轻量级，双向数据绑定，指令，插件化

Angular：学习成本高，依赖对数据做脏检查

React：使用 jsx 语法。React 依赖 virtual DOM，而 vue.js 使用的是 DOM 模板，单向数据流，推崇函数式编程和单向数据流

21. 现代前端攻击技术的了解

1. XSS，跨站脚本攻击（往 web 网页里插入恶意 html 代码）

   反射性 xss：早期，通过 url 的输入，将恶意脚本附加到 URL 地址的参数中。能够弹出一个警告框，说明跨站脚本攻击漏洞存在。特点：单击时触发，执行一次。

解决办法：设计 xss Filter，分析用户提交的输入，并消除潜在的跨站脚本攻击、恶意的 HTML 等。在需要 html 输入的地方对 html 标签及一些特殊字符(” < > & 等等)做过滤，将其转化为不被浏览器解释执行的字符。

持久性 xss：攻击者事先将恶意 js 代码上传或存储到漏洞服务器中，只要受害者浏览包含此恶意 js 代码的页面就会执行恶意代码。不用单击触发。

绕过 xss Filter，字符编码，拆分跨站法

防御：输出编码、白名单、黑名单、URL 属性、自定义过滤规则、自定义 CSS 过滤器、去掉不在白名单上的标签及标签体

2. CSRF，跨站点伪造请求。攻击者通过各种方法伪造一个请求，模仿用户提交表单的行为，从而达到修改用户的数据，或者执行特定任务的目的。

解决思路：a.采用 POST 请求,增加攻击的难度.用户点击一个链接就可以发起 GET 类型的请求。而 POST 请求相对比较难，攻击者往往需要借助 javascript 才能实现。

B. 对请求进行认证，确保该请求确实是用户本人填写表单并提交的，而不是第三者伪造的.具体可以在会话中增加 token,确保看到信息和提交信息的是同一个人

3. Http Heads 攻击。响应头和内容之间有一个及空行，攻击者将任意字符“注入”到 headers 中。

   解决方法：就是过滤所有的 response headers，除去 header 中出现的非法字符，尤其是 CRLF

22.为什么使用多个域名访问

1. 减少主域名的连接数，优化页面响应速度；
2. 突破浏览器并发限制，浏览器能一次发送的 http 请求有限
3. CDN 缓存更方便
4. 节约 cookie 带宽
5. 防止不必要的安全问题

23.iframe 的优缺点

优点：原封不动把嵌入网页展示出来；如果有多个引用该 iframe 则只需要修改这个 iframe；遇到加载慢的第三方内容如图标和广告，可以用 iframe

缺点：产生很多页面，不易管理；用户体验差；代码复杂；不易于搜索引擎优化；增加服务器的 http 请求

27.盒子模型

content+padding+border+margin

W3C 盒模型：width=content

IE 盒模型：width=content+padding+border

切换盒模型：box-sizing：border-box/content-box(默认)

28.position 的几种值

static:默认；relative（不脱离文档流）；absolute；fixed（相对于 body 定位）；sticky：粘性定位

17. 清除浮动相关

清除浮动的原因：父元素因为子级元素浮动导致父级对象盒子不能被撑开

方法：1.在最后一个浮动标签后新增一个标签并设置 clear：both；（过多的无语义化标签）2.父级元素添加 overflow：hidden（会导致内容被隐藏，无法显示要溢出的元素）3.使用 after 伪元素清除浮动；

18. 数组方法(去重、排序等等)

concat（）链接两个或更多数组

join（）数组化为字符串，split（）字符串化为数组

unshift（）和 push（）从数组前后添加元素

shift（）和 pop（）从数组前后删除元素

sort（）排序　reverse（）数组反转

splice（）删除或添加新元素

slice（start，end）不改变之前数组，返回切除来的数组

indexOf（）和 lastIndexOf（）某个元素索引值

arr.fill（target，start，end）使用给定的值填充一个数组，改变原数组

target:待填充的元素，start：开始位置，end：结束位置

arr.copyWithin（）将制定位置的数组复制到其他位置，覆盖元素组项，返回当前数组，

参数：target（必选，当前位置开始替换），start（可选，当前位置读取），end（可选，当前位置停止读取）

includes（）数组中是否包含给定的值

keys（）遍历数组的键名　values（）遍历数组键值　entries()遍历数组键名和键值

Array.from（）伪数组变成数组

Array.of（）将一组值转换成数组，类似于声明数组

tips:callback 参数：value，index，array

arr.forEach（callback）遍历数组，无 return

arr.map(callback)映射数组，有 return 返回一个新数组

arr.filter（callback）过滤数组

arr.every（callback）依据判断条件，数组元素全满足则返回 true

arr.some（callback）若有一个满足则返回 true

arr.find（callback）找到第一个符合条件的数组成员

arr.findIndex（callback）找到第一个符合条件的数组成员的索引值

arr.reduce（callback，initialValue）迭代数组的所有项，累加器，数组的每个值从左到右合并，最终计算为一个值。

arr.reduceRight（callback，initialValue），与 reduce 功能一样，从末尾到前顺序

callback:previousValue(必选，上一次调用回调返回的值),currentValue(必选，当前被处理的数组项),index(可选，索引值),arr(可选，原数组);initialValue(可选，初始值)

19. date 类型转换

GMT：格林尼治平均时

UTC：通用协调时

Timestamp：时间戳。格林尼治时间 1970 年 01 月 01 日 00 时 00 分 00 秒起至现在的毫秒数

getFullYear ()，getMonth ()，getDate ()，getDay ()，getHours ()，getMinutes ()，getSeconds ()

20. Css3 动态效果的实现

animation 使用：

```
div {
        animation:mymove 5s infinite;
        -webkit-animation:mymove 5s infinite;
        /* Safari 和 Chrome */
}
@keyframes mymove
{
    from {left:0px;}
    to {left:200px;}
}
```

```
animation: name duration timing-function delay iteration-count direction fill-mode play-state;
```

过渡 transition：过渡 transition 是一个复合属性，包括 transition-property、transition-duration、transition-timing-function、transition-delay 这四个子属性

transition-property: 过渡属性(默认值为 all)，可以指定为 width，background 等

transition-duration: 过渡持续时间(默认值为 0s)

transiton-timing-function: 过渡函数(默认值为 ease 函数)。典型的诸如 step 函数，贝塞尔曲线

transition-delay: 过渡延迟时间(默认值为 0s)

一般地，过渡 transition 的触发有三种方式，分别是伪类触发、媒体查询触发和 javascript 触发。其中常用伪类触发包括:hover、:focus、:active 等

变形 transform：旋转(rotate),扭曲(skew)，缩放 (scale)、移动(translate)和矩阵变形 matrix

transform ： none | <transform-function> [<transform-function>]*

transform: rotate | scale | skew | translate |matrix;

none：

无转换

matrix(<number>,<number>,<number>,<number>,<number>,<number>)：

以一个含六值的(a,b,c,d,e,f)变换矩阵的形式指定一个 2D 变换，相当于直接应用一个[a,b,c,d,e,f]变换矩阵

translate(<length>[, <length>])：

指定对象的 2D translation（2D 平移）。第一个参数对应 X 轴，第二个参数对应 Y 轴。如果第二个参数未提供，则默认值为 0

translateX(<length>)：

指定对象 X 轴（水平方向）的平移

translateY(<length>)：

指定对象 Y 轴（垂直方向）的平移

rotate(<angle>)：

指定对象的 2D rotation（2D 旋转），需先有 transform-origin 属性的定义

scale(<number>[, <number>])：

指定对象的 2D scale（2D 缩放）。第一个参数对应 X 轴，第二个参数对应 Y 轴。如果第二个参数未提供，则默认取第一个参数的值

scaleX(<number>)：

指定对象 X 轴的（水平方向）缩放

scaleY(<number>)：

指定对象 Y 轴的（垂直方向）缩放

skew(<angle> [, <angle>])：

指定对象 skew transformation（斜切扭曲）。第一个参数对应 X 轴，第二个参数对应 Y 轴。如果第二个参数未提供，则默认值为 0

skewX(<angle>)：

指定对象 X 轴的（水平方向）扭曲

skewY(<angle>)：

指定对象 Y 轴的（垂直方向）扭曲

21. 事件机制（冒泡，捕获）

DOM 事件流包括三个阶段：1.事件捕获阶段（从最上一级标签开始往下查找，直到捕获到事件目标）；2.目标阶段；3.事件冒泡阶段（事件从事件目标（target）开始，往上冒泡直到页面的最上一级标签。

阻止冒泡：stopPropagation（）；cancelBubble=true（IE 下）

22. 样式导入方式

1. 行内式，写在 style 属性中实现
2. 嵌入式，写在 style 标签中
3. 引入式，外部链接
4. 导入式，@import。（基本不使用，先加载 html 再加载 css，会导致页面样式的延迟）

23. 前端性能优化方案

SEO：1.合理的 title、description、keywords ；2.语义化的 HTML 代码 ；3.重要内容的 HTML 代码放在最前 ；4.重要内容不要用 js 输出 ；5.少用 iframe ；6.非装饰性图片必须加 alt ；7.提高网站速度

性能：

24. flex 布局

弹性布局

flex-direction:项目排列方式。row | row-reverse | column | column-reverse

flex-wrap:项目换行方式。nowrap | wrap | wrap-reverse

flex-flow:排列方式和换行方式的简写，用||连接

justify-content:项目在主轴上的对其方式。flex-start | flex-end | center | space-between | space-around

align-items：项目在交叉轴上如何对齐。flex-start | flex-end | center | baseline | stretch;

align-content：多根轴线的对齐方式。flex-start | flex-end | center | baseline | stretch;

项目的属性：order，flex-grow，flex-shrink，flex-basis，flex，align-self

25. grid 布局

网格布局

grid-template-columns：设置列宽（可以是具体数值或者百分比、rem、fr）

grid-template-rows：设置行高

grid-gap：网格项间隙

grid-column-start/ grid-column-end/ grid-row-start/ grid-row-end：垂直方向/水平方向的开始结束位置网格线

可简写为：grid-column，grid-row。用/符号分隔。亦可 grid-area

26. 水平垂直居中一个元素

flex 布局

grid 布局

position 定位

margin

text-align，vertical-align

translate

27. 实现浏览器多个标签页之间的通信

Webstorage

Cookie+setInterval

28. H5 和 CSS3 的新特性

Html5：语义化的标签，增强型表单，视频和音频，canvas 绘图，地理定位，拖放 API，webStorage，websocket

Css3：animation，transition，transform，选择器，@font-face 特性，gradient 渐变，阴影 shadow，圆角，背景效果

29. css 预处理工具（less 和 sass）

基于 css 的另一种语言，通过工具编译成 css，添加了很多 css 不具备的特性，能提升 css 文件的组织

提供功能：1.嵌套，反映层级和约束；2.变量和计算；3.Extend 和 Mixin 代码片段；4.循环 适用于复杂有规律的样式；5.import CSS 文件模块化

30. mvvm 模式和 mvc 模式以及 mvp 模式

双向绑定和单向绑定。

MVC：View 持有了 Controller，把事件传递给 Controller，Controller 由此去触发 Model 层的事件，Model 更新完数据（网络或者本地数据）之后触发 View 的更新事件。

MVP：MVP 其实是 MVC 的封装和演化，Controller 被拆分，只用它处理 View 的点击事件，数据绑定，等处理，而 View 被拆分，更加专注于视图的更新，只做跟视图相关的操作，而 Presenter 被独立出来,用于沟通 View 和 Model 之间的联系,Model 不能直接作用于 View 的更新,只能通过 Presenter 来通知 View 进行视图的刷新,比如 showLoading（),showEmpty()，showToast （）等等，这样 View 就完全被独立出来了，只是被动接受 Presenter 的命令，这样避免了 View 有过多的逻辑处理，更加简单。Presenter 持有了 Model。Model 只用于处理跟数据获取相关的逻辑。

MVVM：又称状态机制，View 和 ViewModel 是进行绑定的，改变 ViewModel 就会直接作用到 View 视图上，而 View 会把事件传递给 ViewModel,ViewModel 去对 Model 进行操作并接受更新。

双向绑定原理： Object.defineProperty() 重新定义了对象获取属性值（get）和设置属性值（set）

31. 正则的了解

str.match(regexp)

match 的用法主要区分就是,正则表达式是否有全局标示 g

1)如果有 g 全局标志,那么返回的数组保存的是,所有匹配的内容，不包过子匹配。

2)如果没有 g 全局标志,那么返回的数组 arr.arr[0]保存的是完整的匹配.arr[1]保存的是第一个括号里捕获的字串,依此类推 arr[n]保存的是第 n 个括号捕获的内容.也就是当包含有全局的标志时则返回的结果第一个是正确匹配的结果，后面依次是子匹配的结果。

var result1 = *regexp*.exec(*str*)　　等价于不含 g 全局标志的 match。

32. 对象的 toString 和 valueOf 方法的应用场景

toString （）函数作用是返回 object 的字符串表示，valueOf （）函数作用是返回该 object

本身。valueOf 的意思是返回最适合该对象类型的原始值，而 toString 则是将在该对象类型的原始值以字符串形式返回

for example：toString():bbb;String(bbb)

valueOf():+bbb；''+bbb；Number（bbb）；bbb=='10''

33. 浅克隆和深度克隆

基本类型：underfined，null，boolean，number 和 string，变量是按值存放的，存放在栈内存中的简单数据段，可以直接访问。

引用类型：存放在堆内存中的对象，对象保存的是一个指针，这个指针指向另一个位置。当需要访问引用类型(如对象、数组等)的值时，需要先从栈中获得该对象的地址指针，然后再从堆内存中取得所需的数据。

JavaScript 存储对象都是存地址的，所以浅拷贝会导致 obj1 和 obj2 指向同一块内存地址。改变了其中一方的内容，都是在原来的内存上做修改会导致拷贝对象和源对象都发生改变，而深拷贝是开辟一块新的内存地址，将原对象的各个属性逐个复制进去。对拷贝对象和源对象各自的操作互不影响。

浅拷贝：1.简单的引用复制；2.Object.assign()（可以把任意多个源对象的可枚举属性拷贝给目标对象，然后返回目标对象）

深拷贝：

数组：slice（）和 concat（）返回一个新数组，不会改变原数组

对象：定义一个新对象，遍历源对象的属性并赋给新对象的属性

通过 json.parse（json.stringfy（））实现深拷贝应该注意的坑：

1. 如果 obj 里面有时间对象，则时间将只是字符串的形式，而不是时间对象
2. 如果 obj 里有 RegExp、Error 对象，则序列化的结果将只得到空对象
3. 如果 obj 里有函数，underfined，则会丢失
4. 如果 obj 里面有 NaN、infinity，则序列化的结果会变成 null
5. JSON.stringfy 只能序列化对象的可枚举的自有属性。如果对象是有构造函数生成的，深

拷贝之后会丢弃对象的 constructor

6. 如果对象中存在循环引用的情况下也无法正确实现深拷贝

   如果不涉及以上情况可以直接用 json.parse（json.stringfy（）），否则可以使用如下方法实现深拷贝：判断对象里面各项的数据类型，如果是对象类型或者是数组类型则递归循环遍历拷贝，直到不再具有下一层次为止

```
function  deepClone(data) {
    const type = this.judgeType(data);
    let obj;

    if (type === 'array') {
      obj = [];
    } else if (type === 'object') {
      obj = {};
    } else {
  //  不再具有下一层次
      return data;
    }
    if (type === 'array') {
      // eslint-disable-next-line
      for (let i = 0, len = data.length; i < len; i++) {
        obj.push(this.deepClone(data[i]));
      }
    } else if (type === 'object') {
```

```
            for (const key in data) {
                obj[key] = this.deepClone(data[key]);
            }
        }
        return obj;
    },
function   judgeType(obj) {
  // tostring 会返回对应不同的标签的构造函数
        const toString = Object.prototype.toString;
        const map = {
            '[object Boolean]': 'boolean',
            '[object Number]': 'number',
            '[object String]': 'string',
            '[object Function]': 'function',
            '[object Array]': 'array',
            '[object Date]': 'date',
            '[object RegExp]': 'regExp',
            '[object Undefined]': 'undefined',
            '[object Null]': 'null',
            '[object Object]': 'object',
        };
        if (obj instanceof Element) {
            return 'element';
```

```
        }

        return map[toString.call(obj)];

    },
```

34. 防抖函数的封装和使用

适用性：解决了多次触发事件时的性能问题

```
var _debounce = function (fn, t) {
    let delay = t || 500;
    let timer;
    return function () {
        let args = arguments;
        if (timer) {
            clearTimeout(timer);
        }
        timer = setTimeout(() => {
            timer = null;
            fn.apply(this, args);
        }, delay);
    }
}
```

35. js 设计模式（常用）

构造函数模式

工厂模式

发布订阅模式

原型链

…

36. webpack 相关知识（面试题）

Webpack 是一个打包模块化 javascript 的工具，在 webpack 里一切文件皆模块，通过 loader 转换文件，通过 plugin 注入钩子，最后输出由多个模块组合成的文件。

WebPack 可以看做是模块打包机：它做的事情是，分析你的项目结构，找到 JavaScript 模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript 等），并将其打包为合适的格式以供浏览器使用。

Webpack 的优缺点：

专注于处理模块化的项目，能做到开箱即用，一步到位
可通过 plugin 扩展，完整好用又不失灵活
使用场景不局限于 web 开发
社区庞大活跃，经常引入紧跟时代发展的新特性，能为大多数场景找到已有的开源扩展
良好的开发体验
缺点：只能用于采用模块化开发的项目

几个常见的 loader：

File-loader:把文件输出到一个文件夹中，在代码中通过相对 URL 去引用输出的文件
url-loader: 和 file-loader 类似，但是能在文件很小的情况下以 base64 的方式把文件内容注入到代码中去
source-map-loader：加载额外的 Source Map 文件，以方便断点调试
image-loader：加载并且压缩图片文件
babel-loader：把 ES6 转换成 ES5
css-loader：加载 CSS，支持模块化、压缩、文件导入等特性
style-loader：把 CSS 代码注入到 JavaScript 中，通过 DOM 操作去加载 CSS。
eslint-loader：通过 ESLint 检查 JavaScript 代码

几个常见的 plugin：

define-plugin：定义环境变量
terser-webpack-plugin：通过 TerserPlugin 压缩 ES6 代码
html-webpack-plugin：为 html 文件中引入的外部资源，可以生成创建 html 入口文件
mini-css-extract-plugin：分离 css 文件
clean-webpack-plugin：删除打包文件
happypack：实现多线程加速编译

Webpack 和 grunt 和 gulp 有啥不同：

Webpack 是一个模块打包器，grunt 和 gulp 是执行任务的，webpack 可以递归的打包项目中的所有模块（递归：指定一个入口，分析模块的依赖，它会递归的查找所有相关的依赖），最终生成几个打包后的文件，他和其他的工具的最大的不同在于它支持 code-splitting（代码分割），模块化（AMD，ESM，CommonJS）开发，全局的分析工具（分析整个项目引入的模块）

什么是 bundle，什么是 chunk，什么是 module：

bundle 是由 webpack 打包出来的文件，chunk 是指 webpack 在进行模块依赖分析的时候，代码分割出来的代码块，module 是开发中的单个模块

什么是 loader，什么是 plugin：

loader 是用来告诉 webpack 如何转化处理某一类型的文件，并且引入到打包出的文件中。plugin 是用来自定义 webpack 打包过程中的方式，一个插件是含有 apply 方法的一个对象，通过这个方法可以参与到整个 webpack 打包的各个流程（生命周期）

webpack-dev-server 和 http 服务器如 nginx 有什么不同：

webpack-dev-server 使用内存来存储 webpack 开发环境下的打包文件，并且可以使用模块热更新，他比传统的 http 服务器对开发更加简单高效

什么模块热更新：

模块热更新是 webpack 的一个功能,他可以使得代码修改过后不用刷新浏览器就可以更新，是高级版的自动刷新浏览器（将代码重新执行而不整体刷新浏览器）。优势：保持应用的数据状态，节省调试时间，样式调试更快

什么是长缓存？在 webpack 中如何做到长缓存优化？

浏览器在用户访问页面的时候为了加快加载速度，会对用户访问的静态资源进行存储，但是每一次代码升级或者更新都需要浏览器去下载新的代码，最方便和简单的更新方式就是引入新的文件名称。在 webpack 中可以在 output 给输出的文件制定 chunkhash，并且分离经常更新的代码和框架代码。通过 namedModulesPlugin 或是 HashdModuleIdsPlugin 使再次打包文件名不变

37. spa 单页面应用程序的优缺点

通过改变 URL，在不重新请求页面的情况下，更新页面视图

优点：体验好，前后端分离，减轻服务端压力

缺点：seo 不利于搜索引擎抓取，首屏加载较慢

38. 网络七层协议

网络七层协议的通俗理解

OSI中的层	功能	TCP/IP协议族
应用层	文件传输，电子邮件，文件服务，虚拟终端	TFTP，HTTP，SNMP，FTP，SMTP，DNS，RIP，Telnet
表示层	数据格式化，代码转换，数据加密	没有协议
会话层	解除或建立与别的接点的联系	没有协议
传输层	提供端对端的接口	TCP，UDP
网络层	为数据包选择路由	IP，ICMP，OSPF，BGP，IGMP，ARP，RARP
数据链路层	传输有地址的帧以及错误检测功能	SLIP，CSLIP，PPP，MTU，ARP，RARP
物理层	以二进制数据形式在物理媒体上传输数据	ISO2110，IEEE802，IEEE802.2

39. Javascript 有哪几种方法定义函数

1.函数声明表达式；2.function 操作符；3.function 构造函数；4.ES6 箭头函数

40. apply，call

修改函数的执行上下文（this）,改变 this 的指向

1. 每个函数都包含两个非继承而来的方法：apply（）和 call（）
2. 他们的用途相同，都是在特定的作用域中调用函数
3. 接收参数方面不同，apply（）接收两个参数，一个是函数运行的作用域（this），另一个是参数数组
4. call（）方法第一个参数与 apply 方法相同，但传递的参数必须列举出来

bind()最简单的用法是创建一个函数，使这个函数不论怎么调用都有同样的 this 值。

不用 call 和 apply 模拟实现 bind()方法：https://www.jianshu.com/p/6a1bc149b598

41. BFC 块级格式化上下文

创建规则：1.根元素；2.浮动元素；3.绝对定位元素；3.display 取值为 inline-block、table-cell、table-caption、flex、inline-flex 之一的元素；4.overflow 不是 visible 的元素

作用：1.可以包含浮动元素；2.不被浮动元素覆盖；3.阻止父子元素的 margin 折叠

42.项目中遇到的印象深刻的点以及解决方式

1.关于分页数据绑定的问题

问题描述：表单搜索的时候数据是双向绑定的，会出现我在输入框中填入数据没有点

击搜索直接下一页的时候，输入框中的数据会被筛选上。应该只有搜索的时候才应该按照新的筛选条件筛选。

解决方式：克隆表单对象，在下一页或者改变每页显示条数的时候，将克隆的对象和绑定的表单对象合并，克隆替换绑定的值，点击搜索的时候再克隆覆盖一次。

2.客群

问题描述：对象数组的必填校验

解决方式：prop 写成动态的。循环数组。For example ：v-for="(item,index) in conditionForm.customerGroupDtos" 之后每项的 prop 写成 :prop="'customerGroupDtos.'+index+'.cgName'"

3.批次报表里面的 Echarts 图表

问题描述：根据批次编号获取不确定次数呼叫的堆叠柱状图，鼠标移入只显示当前纵坐标并且独立的一条数据的参数信息。以及对获取的数据进行逻辑处理操作。

解决方式：formatter 函数里面处理，获取到鼠标移入地方的数据。

```
axisPointer:{
    type:'cross',
    label:{
        formatter:function(params){
            // param会打印两次，主要取鼠标在x轴的位置
            if(params.seriesData.length==0){
                window.mouseCurValue = params.value;
            }
        },
        show:false,
    }
},
// 鼠标移到某一块显示那一块的数据
formatter:function(params){
    let res = "", sum = 0;
    for(let i=0;i<params.length;i++){
        let series = params[i];
        if(series.data==undefined){
            series.data=0;
        }
        sum += Number(series.data);
        if (sum >= window.mouseCurValue) {
            res = series.axisValue + "<br/>" + series.marker + series.seriesName + ":" + series.data + "<br/>";
            break;
        }
    }
    return res;
}
```

需要进行数据处理的数据格式：

```
{data: [{未通知，确认本人：1，未呼通：4}，{未通知，确认本人：1，未呼通：4}，{开场白：1，未呼通：4}，{开场白：1，未呼通：4},…],
  code: 0
▼data: [{未通知，确认本人：1，未呼通：4}，{未通知，确认本人：1，未呼通：4}，{开场白：1，未呼通：4}，{开场白：1，未呼通：4},…]
  ▶0: {未通知，确认本人：1，未呼通：4}
  ▶1: {未通知，确认本人：1，未呼通：4}
  ▶2: {开场白：1，未呼通：4}
  ▶3: {开场白：1，未呼通：4}
  ▶4: {未呼通：4，已通知本人，未明确态度：1}
  ▶5: {开场白：1，未呼通：4}
  ▶6: {未呼通：5}
  ▶7: {未呼通：5}
  ▶8: {未呼通：5}
  ▶9: {未呼通：5}
  ▶10: {未呼通：5}
```

4.websocket 转人工和录音导出

问题描述：需要实时监听呼叫系统返回的呼叫信息，然后监听是否有通话内容推送，如果有推送，则根据推送的案件 id 来绑定该用户的案件并跳转到案件详情页，同时展示出通话内容以便记催记。

解决方式：使用 SockJs 和 stomp+websocket 实时刷新数据，先建立链接对象，然

后获取 STOMP 子协议的客户端对象，然后向服务器发起 websocket 连接并发送 connect 帧。链接成功后订阅消息。然后将数据存在 localstorage 里面在案件详情页使用监听到的数据。

```
//二、 监听呼叫系统返回信息
window.addEventListener('message', this.watchMessage, false);
//建立连接对象（还未发起连接）
var socket = new SockJS("/api/webSocketServer");
// 获取 STOMP 子协议的客户端对象
self.stompClient = Stomp.over(socket);
// 向服务器发起websocket连接并发送CONNECT帧
self.stompClient.connect({}, function connectCallback(frame) {
  // 连接成功时（服务器响应 CONNECTED 帧）的回调方法
  console.log("连接成功");
  // console.log(frame);
  // setConnected(true);
  //订阅消息 如果返回type为1是未接来电消息提醒，2是未读消息提醒
  self.stompClient.subscribe('/user/topic/remind', function (response) {
    if (response) {
      // console.log(response);
      // console.log(response.body);
      // console.log(typeof(response.body));
      //把通话记录存在localStorage里面
      localStorage.setItem("telNote",response.body);
```

问题描述（录音导出）：由于录音文件较大，普通的下载会让页面点击后长时间处于无反应状态（请求时间较长），所以改成 websocket 下载，下载中同时提示下载进度。同时用户在下载过程中阻止用户离开页面操作。用户离开当前页面进行提示，离开后断开原来的 websocket 链接。

解决方式（录音导出）：

```
let self=this;
// self.percentage=parseInt(300/122);
self.ws=new WebSocket('wss://'+window.location.host+'/api/download/');
// 链接成功时候
self.ws.addEventListener('open',function(event){
  let param=self.downloadForm;
  console.log("链接成功");
  console.log(event);
  console.log("发送数据");
  //  如果是已经下载了一部分 就不需要重新传参了
  self.ws.send(JSON.stringify(self.downloadForm));
  console.log("发送数据成功");
  localStorage.setItem("fresh",true)
})
self.ws.addEventListener('message',function(event){
  let msg=JSON.parse(event.data);
  console.log(msg);
  if(msg.code==101){
    self.showprocess=true;
    let complete=msg.data.complete,total=msg.data.total;
    //  等待压缩完成的时候不会有这两个参数
    if(complete&&total){
      self.percentage=parseInt(complete*100/total);
    }
```

```
  let that=this;
// 如果有正在导出的录音的话刷新给提示
  window.addEventListener("beforeunload",(e)=>{
   if(localStorage.getItem("fresh")){
    e=e||window.event;
    if(e){
     e.returnValue="确定离开当前页面吗";
     }
     return "确定离开当前页面吗"
     }
   })
    window.addEventListener("unload",(e)=>{
      if(localStorage.getItem("fresh")){
          that.ws.close();
          localStorage.removeItem("fresh");
      }
    })
      window.addEventListener("load",(e)=>{
      if(localStorage.getItem("fresh")){
        that.$message.warning("录音导出中断,如有需要请重新导出");
      }
    })

estroyed(){
 if(localStorage.getItem("fresh")){
   this.$message.warning("上个页面还有未处理完成的操作！");
```

5.tree 多个树形节点的绑定

问题描述：需要提交的对象数组中有多个 tree 需要遍历绑定半选中的值

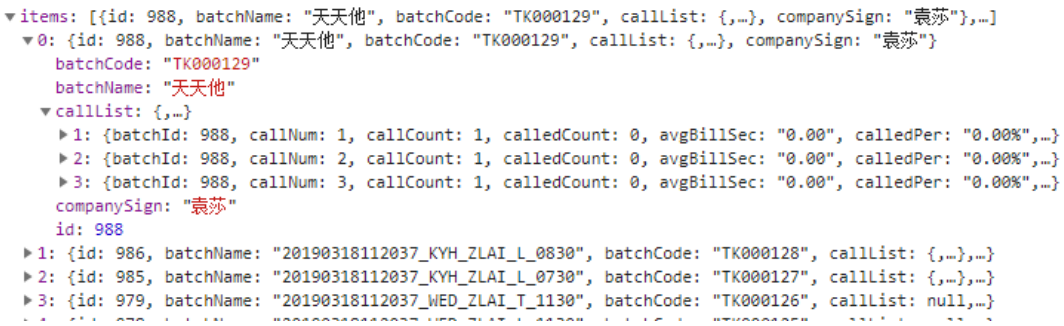
解决方式：this.$refs.tree[index].getHalfCheckKeys().forEach(element =>{

Chooseprovince.push(element);

})

6.自定义表格样式（单次通话详情）

问题描述：表头固定，前两列一项数据，后面分别有五项数据，且数据长度以及有无数据不固定。数据格式：



解决方式：表格原生写法，v-for 循环，样式用 position：sticky 固定表头。

7.异步回调多次的控制

问题描述：接口统一的错误处理，登录超时的时候提示并跳转到登录页面，然后会出现用户刚登录成功就超时又重新链接到登录页的情况。

解决方式：加公共字段，为 true 的时候执行重定向操作，执行该次操作一次之后将改值设置成 false，每次 login 的时候设置为 true。

8.双向绑定直接赋值会导致校验失效

问题描述:修改的时候，直接给表单赋值，赋值的对象里面包含需要切换的对象数组，当这个对象数组有表单切换的时候，校验会失效。

分析原因：直接粗暴赋值，则里面的对象数组是新的对象数组，不再是原先的数组，所以双向绑定失效，原先的校验也就失效了

解决方式：遍历赋值。

```
if (res.data.code == 0) {
  // this.data = res.data.data;
  let dataObj = res.data.data;
   for(let key in dataObj){
    if(key == "intervalTimeBean"&&!dataObj.intervalTimeBean){}
    else{
      this.data[key] = dataObj[key];
    }
  }
```