最近两周出去面试遇到的面试题（前端初级、长更）

1、vue实现双向数据绑定原理是什么?

html 复制代码

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <script src="https://cdn.bootcss.com/vue/2.5.16/vue.js"></script>
    <!-- 引入vue文件 -->
    <div id="box">
      <new-input v-bind:name.sync="name"></new-input>
      {{name}}
      <!-- 小胡子语法 -->
      <input type="text" v-model="name" />
    </div>
    <script>
      Vue.component("new-input", {
        props: ["name"],
        data: function () {
          return {
            newName: this.name,
          };
        },
        template: `<label><input type="text" @keyup="changgeName"
        v-model="newName" /> 你的名字：</label>`,
        // 模板字符串
        methods: {
          changgeName: function () {
            this.$emit("update:name", this.newName);
          },
        },
        watch: {
          name: function (v) {
            this.newName = v;
          },
        },
        //    监听
      });
```

```
      new Vue({
        el: "#box",
        //挂载实例
        data: {
          name: "nick",
        },
        //赋初始值
      });
    </script>
  </body>
</html>
```

html 复制代码

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <input type="text" v-mode="msg" />
    <p v-mode="msg"></p>
    <script>
      const data = {
        msg: "你好",
      };
      const input = document.querySelector("input");
      const p = document.querySelector("p");
      input.value = data.msg;
      p.innerHTML = data.msg;
      //视图变数据跟着变
      input.addEventListener("input", function () {
        data.msg = input.value;
      });
      //数据变视图变
      let temp = data.msg;
      Object.defineProperty(data, "msg", {
        get() {
          return temp;
        },
        set(value) {
          temp = value;
          //视图修改
          input.value = temp;
          p.innerHTML = temp;
        },
```

```
      });
      data.msg = "小李";
    </script>
  </body>
</html>
```

js 复制代码

八股文我不想写了自己百度去

2、v-model语法糖是怎么实现的

html 复制代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!-- v-model 只是语法糖而已 -->
    <!-- v-model 在内部为不同的输入元素使用不同的property并抛出不同的事件 -->
    <!-- text和textarea 元素使用value property 和 input事件 -->
    <!-- checkbox 和radio使用checked  property 和 change事件-->
    <!-- select 字段将value 作为prop 并将change 作为事件 -->
    <!-- 注意：对于需要使用输入法(如中文、日文、韩文等)的语言，你将会发现v-model不会再输入法
    组合文字过程中得到更新 -->
    <!-- 再普通标签上 -->
    <input v-model="sth" />  //这一行等于下一行
    <input v-bind:value="sth" v-on:input="sth = $event.target.value" />
    <!-- 再组件上 -->
    <currency-input v-model="price"></currentcy-input>
        <!--上行代码是下行的语法糖
         <currency-input :value="price" @input="price = arguments[0]"></currency-input>
        -->
        <!-- 子组件定义 -->
        Vue.component('currency-input', {
         template: `
          <span>
           <input
            ref="input"
            :value="value"
            @input="$emit('input', $event.target.value)"
           >
          </span>
```

```
        `,
        props: ['value'],
      })
</body>
</html>
```

3、Hash和history有什么区别

戳右面的链接：juejin.cn/post/699384...

4、什么是深拷贝和浅拷贝?以及怎么实现深拷贝和浅拷贝?

戳右面的链接： juejin.cn/post/684490...

5、什么是原型什么是原型链?

html 复制代码

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
<script>
    function Person () {

    }
    var person  = new Person();
    person.name = 'Kevin';
    console.log(person.name) // Kevin

    // prototype
    function Person () {

    }
    Person.prototype.name = 'Kevin';
```

```
var person1 = new Person();
var person2 = new Person();
console.log(person1.name)// Kevin
console.log(person2.name)// Kevin

// __proto__
function Person () {

}
var person = new Person();
console.log(person.__proto__ === Person.prototype) // true

//constructor
function Person() {

}
console.log(Person === Person.prototype.constructor) // true

//综上所述
function Person () {

}
var person = new Person()
console.log(person.__proto__ == Person.prototype) // true
console.log(Person.prototype.constructor == Person) // true
//顺便学习一下ES5得方法,可以获得对象得原型
console.log(Object.getPrototypeOf(person) === Person.prototype) // true

//实例与原型
function Person () {

}
Person.prototype.name = 'Kevin';
var person = new Person();
person.name = 'Daisy';
console.log(person.name) // Daisy
delete person.name;
console.log(person.name) // Kevin

//原型得原型
var obj = new Object();
obj.name = 'Kevin',
console.log(obj.name) //Kevin

 //原型链
 console.log(Object.prototype.__proto__ === null) //true
 // null 表示"没用对象" 即该处不应该有值

 // 补充
```

```
    function Person() {

    }
    var person = new Person()
    console.log(person.constructor === Person) // true
    //当获取person.constructor时，其实person中并没有constructor属性,当不能读取到constructor属性时,会从p
    //也就是Person.prototype中读取时,正好原型中有该属性,所以
    person.constructor === Person.prototype.constructor

    //__proto__
    //其次是__proto__，绝大部分浏览器都支持这个非标准的方法访问原型，然而它并不存在于Person.prototype中，
    // 是来自与Object.prototype,与其说是一个属性，不如说是一个getter/setter,当使用obj.__proto__时，可以
    // Object.getPrototypeOf(obj)
    总结：

    1、当一个对象查找属性和方法时会从自身查找,如果查找不到则会通过__proto__指向被实例化的构造函数的prototy

    2、隐式原型也是一个对象,是指向我们构造函数的原型

    3、除了最顶层的Object对象没有__proto_，其他所有的对象都有__proto__,这是隐式原型

    4、隐式原型__proto__的作用是让对象通过它来一直往上查找属性或方法，直到找到最顶层的Object的__proto__属

</script>
</html>
```

6、箭头函数和普通函数有什么区别?

```
js 复制代码
（1）箭头函数比普通函数更加简洁
    如果没有参数，就直接写一个空括号即可
    如果只有一个参数，可以省去参数括号
    如果有多个参数，用逗号分割
    如果函数体的返回值只有一句，可以省略大括号
    如果函数体不需要返回值，且只有一句话，可以给这个语句前面加一个void关键字。最常用的就是调用一个函数：
    let fn = () => void doesNotReturn()

(2) 箭头函数没有自己的this
箭头函数不会创建自己的this,所以它没有自己的this,它只会在自己作用域的上一层继承this。所以箭头函数中的this的

（3）箭头函数继承来的this指向永远不会改变

(4) call()、apply()、bind()等方法不能改变箭头函数中的this指向
```

(5) 箭头函数不能作为构造函数使用

(6) 箭头函数没有自己的arguments

(7) 箭头函数没有prototype

(8) 箭头函数不能用作Generator函数,不能使用yeild关键字

7、New操作符做了什么事情?

```
1、首先创建了一个新对象
2、设置原型，将对象的原型设置为函数的prototype对象
3、让函数的this指向这个对象，执行构造函数的代码（为这个新对象添加属性）
4、判断函数的返回值类型，如果是值类型，返回创建的对象。如果是引用类型，就返回这个引用类型的对象
```

8、说一下eventloop

戳右边链接：segmentfault.com/a/119000001...

9、什么是闭包，闭包的作用是什么

```
当一个内部函数被调用，就会形成闭包，闭包就是能够读取其他函数内部变量的函数。
闭包作用：
局部变量无法共享和长久的保存，而全局变量可能造成变量污染，所以我们希望有一种机制既可以长久的保存变量又不会造成
```

10、Promise是什么?

Promise 是异步编程的一种解决方案：从语法上讲，promise是一个对象，从它可以获取异步操作的消息；从本意上讲，它是承诺，承诺它过一段时间会给你一个结果。promise有三种状态： pending(等待态)，fulfiled(成功态)，rejected(失败态) ；状态一旦改变，就不会再变。创造promise实例后，它会立即执行。

```
const PENDING = "pending";
const RESOLVED = "resolved";
const REJECTED = "rejected";

function MyPromise(fn) {
  // 保存初始化状态
  var self = this;

  // 初始化状态
  this.state = PENDING;

  // 用于保存 resolve 或者 rejected 传入的值
  this.value = null;

  // 用于保存 resolve 的回调函数
  this.resolvedCallbacks = [];

  // 用于保存 reject 的回调函数
  this.rejectedCallbacks = [];

  // 状态转变为 resolved 方法
  function resolve(value) {
    // 判断传入元素是否为 Promise 值，如果是，则状态改变必须等待前一个状态改变后再进行改变
    if (value instanceof MyPromise) {
      return value.then(resolve, reject);
    }

    // 保证代码的执行顺序为本轮事件循环的末尾
    setTimeout(() => {
      // 只有状态为 pending 时才能转变，
      if (self.state === PENDING) {
        // 修改状态
        self.state = RESOLVED;

        // 设置传入的值
        self.value = value;

        // 执行回调函数
        self.resolvedCallbacks.forEach(callback => {
          callback(value);
        });
      }
    }, 0);
  }

  // 状态转变为 rejected 方法
  function reject(value) {
    // 保证代码的执行顺序为本轮事件循环的末尾
    setTimeout(() => {
```

```
      // 只有状态为 pending 时才能转变
      if (self.state === PENDING) {
        // 修改状态
        self.state = REJECTED;

        // 设置传入的值
        self.value = value;

        // 执行回调函数
        self.rejectedCallbacks.forEach(callback => {
          callback(value);
        });
      }
    }, 0);
  }

  // 将两个方法传入函数执行
  try {
    fn(resolve, reject);
  } catch (e) {
    // 遇到错误时，捕获错误，执行 reject 函数
    reject(e);
  }
}

MyPromise.prototype.then = function(onResolved, onRejected) {
  // 首先判断两个参数是否为函数类型，因为这两个参数是可选参数
  onResolved =
    typeof onResolved === "function"
      ? onResolved
      : function(value) {
          return value;
        };

  onRejected =
    typeof onRejected === "function"
      ? onRejected
      : function(error) {
          throw error;
        };

  // 如果是等待状态，则将函数加入对应列表中
  if (this.state === PENDING) {
    this.resolvedCallbacks.push(onResolved);
    this.rejectedCallbacks.push(onRejected);
  }

  // 如果状态已经凝固，则直接执行对应状态的函数
```

```
  if (this.state === RESOLVED) {
    onResolved(this.value);
  }

  if (this.state === REJECTED) {
    onRejected(this.value);
  }
};
```

11、Set 和 Map有什么区别?

js 复制代码

1、**Map**是键值对，**Set**是值得集合，当然键和值可以是任何得值
2、**Map**可以通过get方法获取值，而set不能因为它只有值
3、都能通过迭代器进行**for...of** 遍历
4、**Set**的值是唯一的可以做数组去重，而**Map**由于没有格式限制，可以做数据存储

12、map和foreach有什么区别

js 复制代码

foreach()方法会针对每一个元素执行提供得函数,该方法没有返回值,是否会改变原数组取决与数组元素的类型是基本类型还
map()方法不会改变原数组的值,返回一个新数组,新数组中的值为原数组调用函数处理之后的值:

13、localStorage sessionStorage cookies 有什么区别?

js 复制代码

localStorage:以键值对的方式存储 储存时间没有限制 永久生效 除非自己删除记录
sessionStorage: 当页面关闭后被清理与其他相比不能同源窗口共享 是会话级别的存储方式
cookies 数据不能超过4k 同时因为每次http请求都会携带cookie 所有cookie只适合保存很小的数据 如会话标识

14、Vuex有哪些基本属性?为什么 Vuex 的 mutation 中不能做异步操作?

js 复制代码

有五种，分别是 **State**、 **Getter**、**Mutation** 、**Action**、 **Module**
1、state => 基本数据(数据源存放地)
2、getters => 从基本数据派生出来的数据
3、mutations => 提交更改数据的方法，同步
4、actions => 像一个装饰器，包裹mutations，使之可以异步。

5、modules => 模块化**Vuex**

1、**Vuex**中所有的状态更新的唯一途径都是mutation，异步操作通过 **Action** 来提交 mutation实现，这样可以方便地跟踪
2、每个mutation执行完成后都会对应到一个新的状态变更，这样devtools就可以打个快照存下来，然后就可以实现 time-

15、Loader和Plugin 有什么区别

Loader：直译为"加载器"。Webpack将一切文件视为模块，但是webpack原生是只能解析js文件，如果想将其他文件也打包的话，就会用到`loader`。 所以Loader的作用是让webpack拥有了加载和解析非JavaScript文件的能力。 Plugin：直译为"插件"。Plugin可以扩展webpack的功能，让webpack具有更多的灵活性。 在 Webpack 运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过 Webpack 提供的 API 改变输出结果。

16、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
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
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
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，TC
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
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
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 ht
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
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

17、UDP和TCP有什么区别

	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（数据顺序和正确性），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，例如视频会议、直播	适用于要求可靠传输的应用，例如文件传输

18、项目中常用的性能优化方式有哪些?

js 复制代码

太多了，自己整理吧

19、怎么解决跨域问题的，你是怎么配置的

20、计算属性和watch有什么区别?以及它们的运用场景?

js 复制代码

```
// 区别
  computed 计算属性：依赖其它属性值，并且computed的值有缓存，只有它依赖的属性值发生改变，下一次获取computed
  watch 侦听器：更多的是观察的作用,无缓存性,类似与某些数据的监听回调,每当监听的数据变化时都会执行回调进行后续
//运用场景
  当需要进行数值计算,并且依赖与其它数据时,应该使用computed,因为可以利用computed的缓存属性,避免每次获取值时都
  当需要在数据变化时执行异步或开销较大的操作时,应该使用watch,使用watch选项允许执行异步操作（访问一个API),限制
```

21、Vue的生命周期是什么 每个钩子里面具体做了什么事情

js 复制代码

```
Vue 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模版、挂载Dom -> 渲染、更新 -> 渲染、卸载 等
1、beforeCreate（创建前） ：数据观测和初始化事件还未开始，此时 data 的响应式追踪、event/watcher 都还没有被
2、created（创建后） ：实例创建完成，实例上配置的 options 包括 data、computed、watch、methods 等都配置完成
3、beforeMount（挂载前） ：在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，
4、mounted（挂载后） ：在el被新创建的 vm.$el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译
5、beforeUpdate（更新前） ：响应式数据更新时调用，此时虽然响应式数据更新了，但是对应的真实 DOM 还没有被渲染。
6、updated（更新后）：在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。此时 DOM 已经根据响应式数据的变化
7、beforeDestroy（销毁前） ：实例销毁之前调用。这一步，实例仍然完全可用，`this` 仍能获取到实例。
8、destroyed（销毁后） ：实例销毁后调用，调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，
另外还有 `keep-alive` 独有的生命周期，分别为 `activated` 和 `deactivated` 。用 `keep-alive` 包裹的组件在
```

22、组件之间的传值有几种方式

js 复制代码

```
1、父传子
2、子传父
3、eventbus
4、ref/$refs
```

5、$parent/$children
6、$attrs/$listeners
7、依赖注入(provide/inject)

23、Eventbus具体是怎么实现的

24、父组件到子组件更新的方式是什么样的

25、$nexttick 是干嘛的，你一般拿它做什么

26、Keepalive 是什么，里面有哪些钩子

27、插槽是什么 怎么使用的

28、Es6常见的语法你知道哪一些

29、自定义指令你是怎么用的

30、重绘和重排

31、常见的水平垂直方式有几种?

html 复制代码

```
//利用绝对定位，先将元素的左上角通过 top:50%和 left:50%定位到页面的中心，然后再通过 translate 来调整元素的中
.parent {
    position: relative;
}

.child {
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%,-50%);
}
//利用绝对定位，设置四个方向的值都为 0，并将 margin 设置为 auto，由于宽高固定，因此对应方向实现平分，可以实现
.parent {
    position: relative;
}

.child {
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
```

```
    margin: auto;
}
//利用绝对定位，先将元素的左上角通过 top:50%和 left:50%定位到页面的中心，然后再通过 margin 负值来调整元素的
.parent {
    position: relative;
}

.child {
    position: absolute;
    top: 50%;
    left: 50%;
    margin-top: -50px;      /* 自身 height 的一半 */
    margin-left: -50px;     /* 自身 width 的一半 */
}
//使用 flex 布局，通过 align-items:center 和 justify-content:center 设置容器的垂直和水平方向上为居中对齐
.parent {
    display: flex;
    justify-content:center;
    align-items:center;
}
//另外，如果父元素设置了flex布局，只需要给子元素加上`margin:auto;`就可以实现垂直居中布局
.parent{
    display:flex;
}
.child{
    margin: auto;
}
```

32、标准盒模型和怪异盒模型

33、Flex常见的属性 flex：1代表什么

34、Rem你是怎么做适配的

35、媒体查询是什么

36、首屏性能优化你是怎么做的

37、怎么解决白屏问题

```
1、加loading
2、骨架屏
```

38、浏览器的性能监控你是怎么做的

戳右边链接：blog.csdn.net/qq_29438877...

39、Diff算法是什么 ： key = index 为什么不常用数组的下标作为index 加了它有什么好处

40、虚拟列表你是怎么实现的

41、说一下防抖和节流

42、哪些情况会导致内存泄漏

```
1、意外的全局变量：由于使用未声明的变量,而意外的创建了一个全局变量,而使这个变量一直留在内存中无法被回收
2、被遗忘的计时器或回调函数：设置了 setInterval 定时器，而忘记取消它，如果循环函数有对外部变量的引用的话，那
3、脱离 DOM 的引用：获取一个 DOM 元素的引用，而后面这个元素被删除，由于一直保留了对这个元素的引用，所以它也无
4、闭包：不合理的使用闭包，从而导致某些变量一直被留在内存当中。
```

43、Vue的父子组件生命周期钩子函数执行顺序?

```
<!-- 加载渲染过程 -->
    <!-- 父beforeCreate -> 父created -> 父beforeMount -> 子beforeCreate -> 子created ->
    子beforeMount -> 子mounted -> 父mounted -->
    <!-- 子组件更新过程 -->
    <!-- 父beforeUpdate -> 子beforeUpdate -> 子updaed -> 父updated -->
    <!-- 父组件跟新过程 -->
    <!-- 父beforeUpdate -> 父updated -->
    <!-- 销毁过程 -->
    <!-- 父beforeDestroy -> 子beforeDestroy -> 子destroyed ->父destroyed -->
```

44、说一下常见的检测数据类型的几种方式?

typeof 其中数组、对象、null都会被判断为**Object**，其他判断都正确

instanceof 只能判断引用数据类型,不能判断基本数据类型

constructor 它有2个作用 一是判断数据的类型，二是对象实例通过constructor对象访问它的构造函数。需要注意的事情

```
Object.prototype.toString.call()
```

45、说一下data为什么是一个函数而不是一个对象?

JavaScript中的对象是引用类型的数据，当多个实例引用同一个对象时，只要一个实例对这个对象进行操作，其他实例中的数据也会发生变化。而在Vue中，我们更多的是想要复用组件，那就需要每个组件都有自己的数据，这样组件之间才不会相互干扰。所以组件的数据不能写成对象的形式，而是要写成函数的形式。数据以函数返回值的形式定义，这样当我们每次复用组件的时候，就会返回一个新的data，也就是说每个组件都有自己的私有数据空间，它们各自维护自己的数据，不会干扰其他组件的正常运行。

46、说一下slice splice split 的区别?

js 复制代码

```
// slice(start,[end])
// slice(start,[end])方法：该方法是对数组进行部分截取，该方法返回一个新数组
// 参数start是截取的开始数组索引，end参数等于你要取的最后一个字符的位置值加上1（可选）。
// 包含了源函数从start到 end 所指定的元素，但是不包括end元素，比如a.slice(0,3)；
// 如果出现负数就把负数与长度相加后再划分。
// slice中的负数的绝对值若大于数组长度就会显示所有数组
// 若参数只有一个，并且参数大于length，则为空。
// 如果结束位置小于起始位置，则返回空数组
// 返回的个数是end-start的个数
// 不会改变原数组
var arr = [1,2,3,4,5,6]
/*console.log(arr.slice(3))//[4,5,6] 从下标为0的到3，截取3之后的数
console.log(arr.slice(0,3))//[1,2,3] 从下标为0的地方截取到下标为3之前的数
console.log(arr.slice(0,-2))//[1,2,3,4]
console.log(arr.slice(-4,4))//[3,4]
console.log(arr.slice(-7))//[1,2,3,4,5,6]
console.log(arr.slice(-3,-3))// []
console.log(arr.slice(8))//[]*/
// 个人总结：slice的参数如果是正数就从左往右数，如果是负数的话就从右往左边数，
// 截取的数组与数的方向一致，如果是2个参数则截取的是数的交集，没有交集则返回空数组
// ps：slice也可以切割字符串，用法和数组一样，但要注意空格也算字符

// splice(start,deletecount,item)
// start：起始位置
// deletecount：删除位数
// item：替换的item
// 返回值为被删除的字符串
```

```
// 如果有额外的参数，那么item会插入到被移除元素的位置上。
// splice:移除，splice方法从array中移除一个或多个数组，并用新的item替换它们。
//举一个简单的例子
var a=['a','b','c'];
var b=a.splice(1,1,'e','f');
 console.log(a) //['a', 'e', 'f', 'c']
 console.log(b) //['b']

 var a = [1, 2, 3, 4, 5, 6];
//console.log("被删除的为：",a.splice(1, 1, 8, 9)); //被删除的为：2
// console.log("a数组元素：",a); //1,8,9,3,4,5,6

// console.log("被删除的为：", a.splice(0, 2)); //被删除的为：1,2
// console.log("a数组元素：", a) //3,4,5,6
console.log("被删除的为：", a.splice(1, 0, 2, 2)) //插入 第二个数为0，表示删除0个
console.log("a数组元素：", a) //1,2,2,2,3,4,5,6

// split(字符串)
// string.split(separator,limit)：split方法把这个string分割成片段来创建一个字符串数组。
// 可选参数limit可以限制被分割的片段数量。
// separator参数可以是一个字符串或一个正则表达式。
// 如果separator是一个空字符，会返回一个单字符的数组，不会改变原数组。
var a="0123456";
var b=a.split("",3);
console.log(b);//b=["0","1","2"]
// 注意：String.split() 执行的操作与 Array.join 执行的操作是相反的。
```

47、说一下怎么把类数组转换为数组?

js 复制代码

```
//通过call调用数组的slice方法来实现转换
Array.prototype.slice.call(arrayLike)

//通过call调用数组的splice方法来实现转换
Array.prototype.splice.call(arrayLike,0)

//通过apply调用数组的concat方法来实现转换
Array.prototype.concat.apply([],arrayLike)

//通过Array.from方法来实现转换
Array.from(arrayLike)
```

48、说一下数组如何去重,你有几种方法?

```js 复制代码
let arr = [1,1,"1","1",true,true,"true",{},{},"{}",null,null,undefined,undefined]

// 方法1
let uniqueOne = Array.from(new Set(arr)) console.log(uniqueOne)

// 方法2
let uniqueTwo = arr => {
    let map = new Map(); //或者用空对象 let obj = {} 利用对象属性不能重复得特性
    let brr = []
    arr.forEach( item => {
        if(!map.has(item)) { //如果是对象得话就判断 !obj[item]
            map.set(item,true) //如果是对象得话就obj[item] =true 其他一样
            brr.push(item)
        }
    })
    return brr
}
console.log(uniqueTwo(arr))

//方法3
let uniqueThree = arr => {
    let brr = []
    arr.forEach(item => {
        // 使用indexOf 返回数组是否包含某个值 没有就返回-1 有就返回下标
        if(brr.indexOf(item) === -1) brr.push(item)
        // 或者使用includes 返回数组是否包含某个值 没有就返回false 有就返回true
        if(!brr.includes(item)) brr.push(item)
    })
    return brr
}
console.log(uniqueThree(arr))

//方法4
let uniqueFour = arr => {
     // 使用 filter 返回符合条件的集合
    let brr = arr.filter((item,index) => {
        return arr.indexOf(item) === index
    })
    return brr
}
console.log(uniqueFour(arr))
```

49、说一下怎么取出数组最多的一项?

```js 复制代码
// 我这里只是一个示例
const d = {};
let ary = ['赵', '钱', '孙', '孙', '李', '周', '李', '周', '周', '李'];
ary.forEach(k => !d[k] ? d[k] = 1 : d[k]++);
const result = Object.keys(d).sort((a, b) => d[b] - d[a]).filter((k, i, l) => d[k] === d[l[0]]);
console.log(result)
```

50、说一下JSON.stringify有什么缺点?

```
1.如果obj里面有时间对象，则JSON.stringify后再JSON.parse的结果，时间将只是字符串的形式，而不是对象的形式
2.如果obj里有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；
3、如果obj里有函数，undefined，则序列化的结果会把函数或 undefined丢失；
4、如果obj里有NaN、Infinity和-Infinity，则序列化的结果会变成null
5、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.pa
6、如果对象中存在循环引用的情况也无法正确实现深拷贝；
```

51、说一下for...in 和 for...of的区别?

```
for...of遍历获取的是对象的键值，for...in获取的是对象的键名；
for...in会遍历对象的整个原型链，性能非常差不推荐使用,而for...of只遍历当前对象不会遍历原型链；
对于数组的遍历,for...in会返回数组中所有可枚举的属性(包括原型链上可枚举的属性),for...of只返回数组的下标对应的
总结：for...in循环主要是为了遍历对象而生,不适用遍历数组；for....of循环可以用来遍历数组、类数组对象、字符串、
```

52、说一下类组件和函数组件的区别?

```
1. 语法上的区别：

函数式组件是一个纯函数，它是需要接受props参数并且返回一个React元素就可以了。类组件是需要继承React.Component

2. 调用方式

函数式组件可以直接调用，返回一个新的React元素；类组件在调用时是需要创建一个实例的，然后通过调用实例里的render

3. 状态管理

函数式组件没有状态管理，类组件有状态管理。
```

4. 使用场景

类组件没有具体的要求。函数式组件一般是用在大型项目中来分割大组件（函数式组件不用创建实例，所有更高效），一般情

53、说一下react的更新机制

答案戳这里：zhuanlan.zhihu.com/p/35801438

54、说一下redux 里面有什么

55、说一下react和vue框架的区别

56、说一下proxy 它有什么优点

57、说一下vue3.0你了解多少?

```
<!-- 响应式原理的改变 Vue3.x 使用Proxy取代 Vue2.x 版本的Object.defineProperty -->
<!-- 组件选项声明方式Vue3.x 使用Composition API setup 是Vue3.x新增的一个选项，他
    是组件内使用Composition API 的入口 -->
<!-- 模板语法变化slot具名插槽语法 自定义指令 v-model 升级 -->
<!-- 其它方面的更改Suspense支持Fragment(多个根节点) 和Protal (在dom其他部分渲染组建内容)组件
     针对一些特殊的场景做了处理。基于treeshaking优化，提供了更多的内置功能。 -->
```

js 复制代码

58、说一下bfc bfc有什么优缺点

59、说一下你对盒模型的理解?

CSS3中的盒模型有以下两种:标准盒模型、IE盒模型
盒模型都是由四个部分组成的,分别是margin、border、padding和content
标准盒模型和IE盒模型的区别在于设置width和height时，所对应的范围不同
1、标准盒模型的width和height属性的范围只包含了content
2、IE盒模型的width和height属性的范围包含了border、padding和content
可以通过修改元素的box-sizing属性来改变元素的盒模型：
1、box-sizing：content-box表示标准盒模型（默认值）
2、box-sizing：border-box表示IE盒模型（怪异盒模型）

js 复制代码

60、说一下SPA单页面有什么优缺点?

```
js 复制代码
```

优点：

1.体验好，不刷新，减少 请求 数据ajax异步获取 页面流程；

2.前后端分离

3.减轻服务端压力

4.共用一套后端程序代码，适配多端

缺点：

1.首屏加载过慢；

2.SEO 不利于搜索引擎抓取

61、说一下前端登录的流程?

初次登录的时候，前端调后调的登录接口，发送用户名和密码，后端收到请求，验证用户名和密码，验证成功，就给前端返回一个token，和一个用户信息的值，前端拿到token，将token储存到Vuex中，然后从Vuex中把token的值存入浏览器Cookies中。把用户信息存到Vuex然后再存储到LocalStroage中,然后跳转到下一个页面，根据后端接口的要求，只要不登录就不能访问的页面需要在前端每次跳转页面师判断Cookies中是否有token，没有就跳转到登录页，有就跳转到相应的页面，我们应该再每次发送post/get请求的时候应该加入token，常用方法再项目utils/service.js中添加全局拦截器，将token的值放入请求头中 后端判断请求头中有无token，有token，就拿到token并验证token是否过期，在这里过期会返回无效的token然后有个跳回登录页面重新登录并且清除本地用户的信息

62、说一下前端权限管理怎么实现

答案戳这里：blog.csdn.net/weixin_4059...

63、说一下购物车的逻辑?

```
//vue中购物车逻辑的实现
1. 购物车信息用一个数组来存储，数组中保存对象，对象中有id和count属性

2. 在vuex中state中添加一个数据 cartList 用来保存这个数组

3. 由于商品详情页需要用到加入购物车功能，所以我们需要提供一个mutation，用来将购物车信息加入 cartList中

4. 加入购物车信息的时候，遵照如下规则： 如果购物车中已经有了该商品信息，则数量累加，如果没有该商品信息，则新增

5. 在商品详情页，点击加入购物车按钮的时候，调用vuex提供的addToCart这个mutation将当前的商品信息 （id count）

// js中购物车逻辑的实现
1.商品页点击“加入购物车”按钮，触发事件

2.事件调用购物车“增加商品”的Js程序（函数、对象方法）

3.向Js程序传递传递“商品id”、“商品数量”等数据

4.存储“商品id”、“商品数量”到浏览器的localStorage中

**展示购物车中的商品******

1.打开购物车页面

2.从localStorage中取出“商品Id”、“商品数量”等信息。

3.调用服务器端“获得商品详情”的接口得到购物车中的商品信息（参数为商品Id）

4.将获得的商品信息显示在购物车页面。

**完成购物车中商品的购买******

1.用户对购物车中的商品完成购买流程，产生购物订单

2.清除localStorage中存储的已经购买的商品信息

备注1：购物车中商品存储的数据除了“商品id”、“商品数量”之外，根据产品要求还可以有其他的信息，例如完整的商品详情

备注2：购物车商品除了存储在localStorage中，根据产品的需求不同，也可以存储在sessionStorage、cookie、session
```

64、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HTT

65、说一下常见的HTTP状态码?说一下状态码是302和304是什么意思？你在项目中出现过么？你是怎么解决的?

```
<!-- 状态码：由3位数字组成，第一个数字定义了响应的类别 -->
<!-- 1xx：指示消息,表示请求已接收，继续处理 -->
<!-- 2xx：成功,表示请求已被成功接收，处理 -->
<!-- 200 OK：客户端请求成功
     204 No Content：无内容。服务器成功处理，但未返回内容。一般用在只是客户端向服务器发送信息，而服务器
     206 Partial Content：服务器已经完成了部分GET请求（客户端进行了范围请求）。响应报文中包含Content-R
-->
<!-- 3xx 重定向 -->
<!-- 301 Moved Permanently：永久重定向，表示请求的资源已经永久的搬到了其他位置。
     302 Found：临时重定向，表示请求的资源临时搬到了其他位置
     303 See Other：临时重定向，应使用GET定向获取请求资源。303功能与302一样，区别只是303明确客户端应该
     307 Temporary Redirect：临时重定向，和302有着相同含义。POST不会变成GET
     304 Not Modified：表示客户端发送附带条件的请求（GET方法请求报文中的IF…）时，条件不满足。返回304时
-->
<!-- 4xx：客户端错误 -->
<!-- 400 Bad Request：客户端请求有语法错误，服务器无法理解。
     401 Unauthorized：请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。
     403 Forbidden：服务器收到请求，但是拒绝提供服务
     404 Not Found：请求资源不存在。比如，输入了错误的url
     415 Unsupported media type：不支持的媒体类型
-->
<!-- 5xx：服务器端错误，服务器未能实现合法的请求。 -->
<!-- 500 Internal Server Error：服务器发生不可预期的错误。
     503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常，
-->
```

66、说一下常见的git操作

```
git branch 查看本地所有分支
git status 查看当前状态
git commit 提交
git branch -a 查看所有的分支
git branch -r 查看远程所有分支
git commit -am "init" 提交并且加注释
git remote add origin git@192.168.1.119:ndshow
git push origin master 将文件给推到服务器上
```

```
git remote show origin 显示远程库origin里的资源
git push origin master:develop
git push origin master:hb-dev 将本地库与服务器上的库进行关联
git checkout --track origin/dev 切换到远程dev分支
git branch -D master develop 删除本地库develop
git checkout -b dev 建立一个新的本地分支dev
git merge origin/dev 将分支dev与当前分支进行合并
git checkout dev 切换到本地dev分支
git remote show 查看远程库
git add .
git rm 文件名(包括路径) 从git中删除指定文件
git clone git://github.com/schacon/grit.git 从服务器上将代码给拉下来
git config --list 看所有用户
git ls-files 看已经被提交的
git rm [file name] 删除一个文件
git commit -a 提交当前repos的所有的改变
git add [file name] 添加一个文件到git index
git commit -v 当你用－v参数的时候可以看commit的差异
git commit -m "This is the message describing the commit" 添加commit信息
git commit -a -a是代表add，把所有的change加到git index里然后再commit
git commit -a -v 一般提交命令
git log 看你commit的日志
git diff 查看尚未暂存的更新
git rm a.a 移除文件(从暂存区和工作区中删除)
git rm --cached a.a 移除文件(只从暂存区中删除)
git commit -m "remove" 移除文件(从Git中删除)
git rm -f a.a 强行移除修改后文件(从暂存区和工作区中删除)
git diff --cached 或 $ git diff --staged 查看尚未提交的更新
git stash push 将文件给push到一个临时空间中
git stash pop 将文件从临时空间pop下来

//  我是cv的自取吧
```