笔试题

1.==的问题

- 两边如果都是字符串，则比较字符串的内容
- 两边 如果都是数值，则比较数值大小
- 两边如果是引用类型，则比较引用地址，故 `[]==[]//false`

比较规则

1.若其中一个数据类型为数值，那么另一个数据类型会先转换成数值然后再与之比较

- 字符串与数值

```
console.log(""==0);   // true
console.log("hello"==1);   // false 'hello'转换为数值为NaN
console.log("000"==0);   // true
console.log("666"==666);   // true
```

- 布尔类型和数值

true为1，false为0

- **数组与数值**

> 数组与数值进行比较，数组会先通过调用`toString()`转换成**字符串**，再转成数值，再进行比较

```
console.log([]==0);   // true []先转为字符串为"",然后""==0
console.log([1]==1);   // true
console.log(["1"]==1);   // true
console.log([1,2]==1);   // false [1,2]==>"1,2"==NaN
console.log([true]==1);   // false [true]==>'true'==>NaN
```

- **null、undefined和NaN**

1. null==undefined为true；
2. null和undefined不会进行数据转换；
3. NaN不与任何值相等，甚至包括它自己！

```
onsole.log(null==undefined);   // true
console.log(null==0);   // false
console.log(null==false);   // false
console.log(null=="");   // false
console.log(undefined==0);   // false
console.log(undefined==false);   // false
console.log(undefined=="");   // false
console.log(undefined==NaN);   // false
console.log(null==NaN);   // false
console.log(NaN==NaN);   // false
```

2. 若其中一个数据类型为布尔值，则会先将其转换成数值再进行比较

（总之就是两边都转换成数值来比较，true==1，false==0，然后再带入其中）

```
console.log(true=="001");   // true
console.log(true==[]);   // false
console.log(true==["true"]);   // false
console.log(true==[true]);   // false
console.log(true==[1]);   // true
```

3. “!” 为逻辑非，在操作非布尔值类型的数据时，会将该数据类型先转换为布尔值后再取反

关于其他数据类型，以下均返回true：

> ① 非空字符串
> ② 非零数值
> ③ 数组
> ④ 对象
> ⑤ 函数

总结

① 相等操作符，不同数据类型会根据一定规则转换为同一数据类型再比较
② 数组与数组比较，比较的是其引用
③ 不同类型比较，遇数值则转数值，布尔值自身转数值
④ 非空非零引用型，转为布尔均为真

2.语义化标签的特点

- 代码结构清晰，方便阅读，有利于团队合作开发
- 方便其他设备解析
- 有利于SEO

3.列举除px外的CSS度量单位并解释其含义

1. rem 相对于根字体大小的单位，比如可以设置 1rem=50px
2. em 相对于父级元素的font-size，比如font-size：16px（浏览器默认），则2em=32px
3. vm 即viewpoint width，视窗宽度，比如1vm的宽度为相对于视窗的宽度的百分之一
4. vh 即viewpoint height ，同上

4.new操作符做了什么?

```
var func = function(){}
var newO = new func()
```

new一共经历了四个阶段

```
var obj = new Object()//1.创建了空对象
obj.__proto__= func.prototype//2.设置原型链 obj的proto属性指向构造函数的prototype
var result = func.call(obj)//3.将func中的this指向obj
if (typeof(result) == "object"){
  func=result;
}
else{
    func=obj;
}// 4.判断Func的返回值类型,如果无返回值 或者 返回一个非对象值，则将 obj 作为新对象返回；否则会将 result 作为新对象返回
```

5. 简述cookie/session记住登录状态机制原理。

cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

6.CSS属性position有哪些值?

绝对，相对，固定，默认值，粘性sticky，inherit（继承父系）

粘性定位是相对定位和绝对定位的结合，可以实现当滚动条往下滑时，导航绝对定位的效果（新属性）

- 粘性定位条件：父元素不能是`overflow:hidden/auto`;必须指定`top`、`bottom`、`left`、`right`4个值之一;父元素的高度不能低于sticky元素的高度；sticky元素仅在其父元素内生效。

7.让文字在水平和垂直方向重叠的两个属性

水平letter-spacing，垂直line-height

8.可继承的样式属性

只有颜色，文字，字体间距行高对齐方式，和列表的样式可以继承。

> 所有元素可继承：visibility和cursor。
> 内联元素可继承：letter-spacing、word-spacing、white-space、line-height、color、font、font-family、font-size、font-style、font-variant、font-weight、text-decoration、text-transform、direction。
> 终端块状元素可继承：text-indent和text-align。
> 列表元素可继承：list-style、list-style-type、list-style-position、list-style-image。

9.多行文本溢出设置省略号

```
p{
 width:200px;
 height: 40px;
 overflow: hidden;//超出部分隐藏
 text-overflow: ellipsis;//溢出的内容为省略号
 display: -webkit-box; //将对象作为弹性盒子显示
 -webkit-line-clamp: 2;//最多显示的行数
 -webkit-box-orient: vertical;//设置对象子元素的排列方式
 font-size: 14px;
}
```

10.异步任务的宏任务和微任务

```
// 定时器任务属于宏任务，并且需要先在任务队列等待，等到同步任务执行完，执行栈清空，才会在任务队列中按顺序选任务进去
setTimeout(() => console.log('a'));//4. 打印a

//Promise 属于异步微任务，在本轮同步任务结束之前执行
Promise.resolve().then(
    // 1. 打印 b
   () => console.log('b');
 ).then(
    // 箭头函数的resolve传递的参数作为下一个then的参数
   () => Promise.resolve('c').then(
     // 执行立即执行函数
     (data) => {
        // 把定时器任务也放入任务队列中等待，在第一个定时器之后
       setTimeout(() => console.log('d')); //5. 打印d
       // 2.打印 f
       console.log('f');
       // 此时返回的 数据作为下一个then的参数
       return data;
     }
   )
 ).then(data => console.log(data)); // 3.打印 c
```

输出顺序是 b,f,c,a,d.

11.将其他数据类型转换为数值数据类型

一：Number()

1. 如果是Boolean值，true和false值将分别被转换为1和0。
2. 如果是数字值，只是简单的传入和返回。
3. **如果是null值，返回0。**
4. 如果是undefined，返回NaN。
5. 如果是字符串：
   a. 如果字符串中只包含数字时，将其转换为十进制数值，忽略前导0（01.1=》1.1）
   b. 如果字符串中包含有效浮点格式，如“1.1”，将其转换为对应的浮点数字，忽略前导0
   c. 如果字符串中包含有效的十六进制格式，如“0xf”，将其转换为相同大小的十进制数值
   d. **如果字符串为空，将其转换为0**
   e. 如果字符串中包含除上述格式之外的字符，则将其转换为NaN
6. 如果是对象，则调用对象的valueOf（）方法，然后依照前面的规则转换返回的值。如果转换的结果是NaN，则调用对象的toString（）方法，然后再依照前面的规则转换返回的字符串值。

二：ParseInt()

处理整数的时候parseInt()更常用。parseInt()函数在转换字符串时，会忽略字符串前面的空格，直到找到第一个非空格字符。

如果第一个字符不是数字或者负号，parseInt() 就会返回NaN，同样的，**用parseInt() 转换空字符串也会返回NaN。**

如果第一个字符是数字字符，parseInt() 会继续解析第二个字符，直到解析完所有后续字符串或者遇到了一个非数字字符。

parseInt()方法还有基模式，可以把二进制、八进制、十六进制或其他任何进制的字符串转换成整数。

```
var num3 = parseInt("10",2);//2(按照二进制解析)
```

基是由parseInt()方法的第二个参数指定的，所以要解析十六进制的值，当然，对二进制、八进制，甚至十进制（默认模式），都可以这样调用parseInt()方法。

三：ParseFloat()

用parseFloat() 转换空字符串也会返回NaN，与parseInt() 函数类似，parseFloat() 也是从第一个字符（位置0）开始解析每一个字符。也是一直解析到字符串末尾，或者解析到遇见一个无效的浮点数字字符为止。 也就是说，字符串中第一个小数点是有效的，而第二个小数点就是无效的了，它后面的字符串将被忽略。

parseFloat() 只解析十进制，因此它没有第二个参数指定基数的用法

如果字符串中包含的是一个可解析为正数的数（没有小数点，或者小数点后都是零），parseFloat() 会返回整数。

> 如果是一个只包含一个数字的数组，parseInt和parseFloat也可以解析出该数字。
>
> 三种方法在转换时，如果是含一个数字的数组或者类数字的字符串，都会忽略前导0

12.ES6有哪些新属性

1. let、const
2. 定义类的语法糖class
3. 新增一种基本数据类型 Symbol
4. 解构赋值
5. 箭头函数
6. 模块化import/export
7. Object类的一些方法（Obect.assign()）
8. Set和Map数据结构

13.GET和POST的区别

1. GET提交的数据放在URL中，POST则不会。这是最显而易见的差别。这点意味着GET更不安全（POST也不安全，因为HTTP是明文传输抓包就能获取数据内容，要想安全还得加密）
2. GET传送的数据量较小，这主要是因为受URL长度限制；Post传送的数据量较大，一般被默认为不受限制
3. GET执行效率比POST好，GET是form提交的默认方法。

14.HTTP和HTTPS的区别

HTTP是不安全的（明文传输），而HTTPS是安全的; HTTP标准端口是80,而HTTPS的标准端口是443;在网络模型中, HTTP工作于应用层，而HTTPS工作在传输层; HTTP无需证书,而HTTPS需要认证证书。

15.箭头函数和普通函数的区别

1. 外观不一样
2. 箭头函数全是匿名函数

3. 箭头函数不能用作构造函数
4. this指向问题，箭头函数本身是没有this的，它内部的this是在它声明时它所处作用域中的this，而普通函数的this就是指向调用此函数的对象，如果这个普通函数时个构造函数，那么this指向构造函数创建的对象实例
5. 箭头函数没有arguments

16.call、apply、bind方法使用

这三个方法可以改变普通函数中this的指向

```
let web = {webName:"蚂蚁部落"};
function fn() {
  console.log(this);
  console.log(this.webName);
}
fn.call(web);//蚂蚁部落
//构造函数形式
function lay(name,age) {
   this.name=name;
   this.age=age;
   this.sayAge=function()=>{
       console.log(this.age)
   }
}
var p1=new lay('zs',14)
var p2={name:'ls',age:'11'}
p1.sayAge.call(p2)//11
```

当函数需要传递多个变量时, apply 可以接受一个数组作为参数输入, call 则是接受一系列的单独变量。bind 和call很相似，第一个参数是this的指向，从第二个参数开始是接收的参数列表。区别在于bind方法返回值是函数以及bind接收的参数列表的使用。

```
var p2Way=p1.sayAge.bind(p2)
p2Way()//11
```

17.垂直/水平/垂直水平居中

- 水平居中

1.行内元素：父元素是块元素，如果不是就设置成块元素，然后给该元素text-align:center；

2.块级元素，margin：0 auto;

3.弹性盒子实现

- 垂直居中

1.行内元素：line-height设置跟父元素一样高

2.块级元素：设置父元素相对定位，子元素绝对定位，top：50%。margin-top：-元素高度的一半

3.弹性盒子实现

- 垂直水平居中

1.弹性盒子

2.绝对定位，四个方向设置为0，margin：auto

3.left：50%；top：50%；margin-left：-元素宽度的一半；margin-top：-元素宽度的一半

18. CSS3有哪些新特性

- border-radius
- box-shadow
- border-image
- @font-face
- 2d转换：translate() rotate() scale()
- 动画 @keyframes

19.减少网页加载的方法

服务器角度

- 增加服务器带宽

客户端角度

- 合理组织CSS、JavaScript代码位置
- 减少DOM操作、添加事件委托
- 对于图片可以懒加载
- 合并CSS图片（精灵图/雪碧图）

资源优化打包角度

- 使用打包工具将js文件、CSS文件和静态文件进行恰当打包处理。

20. http协议中与资源缓存相关的协议头有哪些?

expire

cache-control

if-modified-since/last-modified

etag/if-none-match

21. 在Javascript中什么情况下会进行装箱/拆箱转换?

装箱：把基本数据类型转换成对应的引用类型的操作。

拆箱：把引用类型转换成基本数据类型的操作。

在Javascript中出现 基本数据类型数据 和 引用数据类型数据要进行转换的情况下会进行装箱/拆箱操作。

22.性能优化

- 减少http请求
- CSS Sprites
- 懒加载图片
- 合理设计事件监听器
- 压缩js和css

23.函数声明提升和变量声明提升

函数声明提升的优先级高于变量声明提升。

```
console.log(a)//输出函数
var a=3
function a(){
    console.log(1);
}
```

但是，如果变量赋予了初值，那么变量优先级大于函数

```
var a=3
function a(){
    console.log(1);
}
console.log(a)//输出3
```

24.vue的scoped

vue中的scoped属性的效果主要通过PostCSS转译实现，如下是转译前的vue代码：

```
<style scoped>
.example {
  color: red;
}
</style>

<template>
  <div class="example">hi</div>
</template>
```

转译后： PostCSS给一个组件中的所有dom添加了一个独一无二的动态属性，然后，给CSS选择器额外添加一个对应的属性选择器来选择该组件中dom，这种做法使得样式只作用于含有该属性的dom——组件内部dom。

```
<style>
.example[data-v-5558831a] {
  color: red;
}
</style>

<template>
  <div class="example" data-v-5558831a>hi</div>
</template>
```

在组件中修改第三方组件库样式的其它方法

1. scoped穿透

```
外层 >>> 第三方组件
      样式

.wrapper >>> .swiper-pagination-bullet-active
 background: #fff
```

2. 在vue组件中的style标签上不使用scoped
3. 在vue组件中使用两个style标签，一个加上scoped属性，一个不加scoped属性，把需要覆盖的css样式写在不加scoped属性的style标签里
4. 建立一个reset.css(基础全局样式)文件，里面写覆盖的css样式，在入口文件main.js 中引入

25.js的防抖和节流（性能优化）

对于**短时间内连续触发**的事件（上面的滚动事件），**防抖的含义就是让某个时间期限（如上面的1000毫秒）内，事件处理函数只执行一次。**

```
    function debounce(fn,delay){
    var 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) // 进入该分支说明当前并没有在计时，那么就开始一个计时
    }
}
//如果鼠标不断滑动滚动条，那么就会不停的清除定时器和生成延时定时器，当松开后，会在最后一个定时器规定的延时时间后执行。
   function showTop(){
           var scrollTop=document.body.scrollTop||document.documentElement.scrollTop
           console.log('当前滚动条位置为:'+scrollTop)
       }
       window.onscroll=debounce(showTop,1000)
```

函数节流（throttle）：当持续触发事件时，保证一定时间段内只调用一次事件处理函数。

```
 function showTop(){
           var scrollTop=document.body.scrollTop||document.documentElement.scrollTop
           console.log('当前滚动条位置为:'+scrollTop)
       }
       window.onscroll=throttle(showTop,1000)
       function throttle(fn,delay){
             timer=null;
         return function(){
            if(!timer){//!null和!undefined都为true
              timer=setTimeout(()=>{
                   fn()
                   timer=null
               },delay)
            }
         }
```

```
        }
```

区别在于，假设一个用户一直触发这个函数，且每次触发函数的间隔小于wait，防抖的情况下只会调用一次，而节流的情况会每隔一定时间（参数wait）调用函数。

26.transition和animation

transition 过渡

1. `transition` 需要事件触发，所以没法在网页加载时自动发生。
2. `transition` 是一次性的，不能重复发生，除非一再触发。
3. `transition` 只能定义开始状态和结束状态，不能定义中间状态，也就是说只有两个状态。
4. 一条 `transition` 规则，只能定义一个属性的变化，不能涉及多个属性。

animation 动画

1. animation不需要触发任何事件， 他的实现不需要触发事件，设定好时间之后可以自己执行，且可以循环一个动画
2. 通过关键帧控制动画的每一步，控制更为精确
3. 对元素某个属性或多个属性的变化

28.ajax、axios、fetch的区别

- ajax

1. 本身是针对MVC的编程,不符合现在前端MVVM的浪潮
2. 基于原生的XHR开发，XHR本身的架构不清晰，已经有了fetch的替代方案
3. JQuery整个项目太大，单纯使用ajax却要引入整个JQuery非常的不合理

- fetch

1. 符合关注分离，没有将输入、输出和用事件来跟踪的状态混杂在一个对象里
2. 更好更方便的写法
3. 脱离了XHR，是ES6规范里新的实现方式

- axios

1. 从 node.js 创建 http 请求
2. 支持 Promise API
3. 自动转换JSON数据
4. 拦截请求和响应

29.cookie特点

1. 大小受限，4kb
2. 安全性较低
3. 有些状态不可能保存在客户端。
4. 每次访问都要传送cookie给服务器，浪费带宽。

相比较之下webStorage：

1. 存储的数据大小限制为5MB，空间更大
2. 节省网络流量，不会传送到服务器
3. 快速显示， 获取数据时可以从本地获取会比从服务器端获取快得多，所以速度更快
4. 不会随着HTTP header发送到服务器端 ，安全性要高一些，但仍然有伪造问题

#

30.Vue的组件的data为什么不能是对象?

当一个**组件**被定义，`data` 必须声明为返回一个初始数据对象的函数，因为组件可能被用来创建多个实例。如果 `data` 仍然是一个纯粹的对象，则所有的实例将**共享引用**同一个数据对象！通过提供 `data` 函数，每次创建一个新实例后，我们能够调用 `data` 函数，从而返回初始数据的一个全新副本数据对象。

31.delete和Vue.delete删除数组的区别

delete只是被删除的元素变成了 empty/undefined 其他的元素的键值还是不变。
Vue.delete 直接删除了数组 改变了数组的键值。

```
var a=[1,2,3,4]
var b=[1,2,3,4]
delete a[1]//下标
    console.log(a)//[1,empty,3,4]
this.$delete(b,1)
    console.log(b)//[1,3,4]
```

32.写 React / Vue 项目时为什么要在列表组件中写 key，其作用是什么?

vue和react都是采用***diff算法***来对比新旧虚拟节点，从而更新节点。

什么是虚拟节点?

在vue中，我们在对dom节点进行操作的时候，不是直接操作真实的dom节点，由于DOM操作很“昂贵”，频繁的进行重绘和回流，效率很低下，因此我们会使用js模拟dom结构，即**virtual dom**，达到渲染局部的效果。

virtual dom与真实dom的区别和优化:

1. 虚拟 DOM 不会立马进行重排与重绘操作。
2. 虚拟 DOM 进行频繁修改，然后一次性比较并修改真实 DOM 中需要改的部分，最后在真实 DOM 中进行排版与重绘，减少过多DOM节点重排与重绘损耗。
3. 虚拟 DOM 有效降低大面积真实 DOM 的重绘与重排，因为最终与真实 DOM 比较差异，可以只渲染局部。
4. 具备跨平台优势。由于 Virtual DOM 是以 JavaScript 对象为基础而不依赖真实平台环境，所以使它具有了跨平台的能力，比如说浏览器平台、Weex、Node 等。

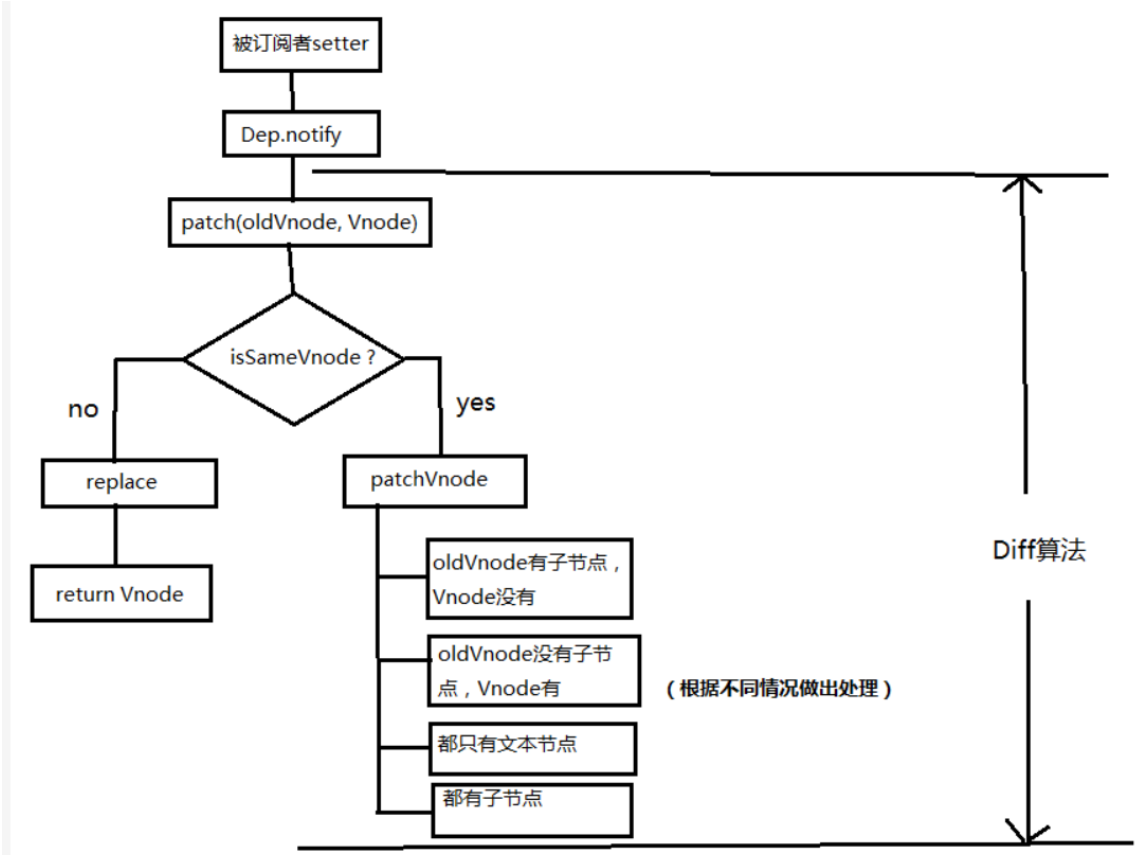
diff算法的实现过程

diff的实现过程: `patch(container, vnode)` 和 - `patch(vnode, newVnode)`

`patch(container, vnode)`:初次渲染的时候，将VDOM渲染成真正的DOM然后插入到容器里面。

`patch(vnode, newVnode)`: 再次渲染的时候，将新的vnode和旧的vnode相对比，然后之间差异应用到所构建的真正的DOM树上。

diff的实现的核心就是 `createElement` 和 `updateChildren`



updateChildren()做了什么?

- 将 `vnode` 的子节点 `vch` 和 `oldvnode` 的子节点 `oldCh` 提取出来
- 当新节点跟旧节点头尾交叉对比没有结果时，会根据新节点的**key**去对比旧节点数组中的key，从而找到相应的旧节点。 如果没找到就认为是一个新增节点。而如果没有key，那么就会采用遍历查找的方式去找到对应的旧节点。

key的作用是什么?

key是给每一个vnode的唯一id,可以依靠key,更准确, 更快的拿到oldVnode中对应的vnode节点

- **更准确**，有效避免“原地复用”带来的副作用。

Vue默认模式不带key只适用于渲染简单的无状态组件。对于大多数场景来说，**列表**组件都有自己的状态。

> 例子：
>
> 一个新闻列表，可点击列表项来将其标记为"已访问"，可通过tab切换“娱乐新闻”或是“社会新闻”。
>
> 不带key属性的情况下，在“娱乐新闻”下选中第二项然后切换到“社会新闻”，"社会新闻"里的第二项也会是被选中的状态，因为这里复用了组件，保留了之前的状态。要解决这个问题，可以为列表项带上新闻id作为唯一key，那么每次渲染列表时都会完全替换所有组件，使其拥有正确状态。

- **更快**， 利用key的唯一性生成map对象来获取对应节点，比遍历方式更快。

33.React 中 setState 什么时候是同步的，什么时候是异步的?

合成事件（react事件）和原生事件的区别

React合成事件一套机制：React并不是将click事件直接绑定在dom上面，而是采用**事件冒泡**的形式冒泡到document上面，然后React将事件封装给正式的函数处理运行和处理。

1. 当用户在某个dom节点上添加onClick函数时，并没有真正将事件绑定到该节点上
2. 而是在document处监听所有支持的事件，当事件发生并冒泡至document处时，React将事件内容封装交给中间层SyntheticEvent（负责所有事件合成）
3. 所以当事件触发的时候，对使用统一的分发函数dispatchEvent将指定函数执行。
4. 原生绑定快于合成事件绑定。

在React的setState函数实现中，会根据一个变量isBatchingUpdates判断是直接更新this.state还是放到队列中回头再说，而isBatchingUpdates默认是false，也就表示setState会同步更新this.state。

但是，有一个函数**batchedUpdates**，这个函数会把isBatchingUpdates修改为true，而当React在调用事件处理函数之前就会调用这个batchedUpdates，造成的后果，就是**由React控制的事件（react事件）处理过程setState不会同步更新this.state。**因此，就会造成一种异步的假象，但是可以以通过第二个参数 setState(partialState, callback) 中的callback拿到更新后的结果。

故：

`setState`只在合成事件和生命周期钩子函数中是“异步”的，在原生事件和`setTimeout`/`setInterval`中都是同步的。

34.实现数组去重的方式

1. 双循环去重
2. `IndexOf`方法
   - 定义空数组res，对原数组进行遍历，如果res数组中不存在原数组的元素，就push进去

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    let res = []
    for (let i = 0; i < arr.length; i++) {
        if (res.indexOf(arr[i]) === -1) {
            res.push(arr[i])
        }
    }
    return res
}
```

   - 利用`indexOf`检测元素在数组中第一次出现的位置是否和元素现在的位置相等，如果不等则说明该元素是重复元素.

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    return Array.prototype.filter.call(arr, function(item, index){
        return arr.indexOf(item) === index;
    });
```

3. 相邻元素方法

   首先调用了数组的排序方法sort()，然后根据排序后的结果进行遍历及相邻元素比对，如果相等则跳过改元素，直到遍历结束.

4. 利用对象属性去重

   创建空对象，遍历数组，将数组中的值设为对象的属性，并给该属性赋初始值1，每出现一次，对应的属性值增加1，这样，属性值对应的就是该元素出现的次数了.

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    let res = [],
        obj = {}
    for (let i = 0; i < arr.length; i++) {
        if (!obj[arr[i]]) {
            res.push(arr[i])
            obj[arr[i]] = 1
        } else {
            obj[arr[i]]++
        }
    }
    return res
}
```

5. set与解构赋值去重

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    return [...new Set(arr)]
}
```

6. `Array.from`和set去重

```
function unique(arr) {
    if (!Array.isArray(arr)) {
        console.log('type error!')
        return
    }
    return Array.from(new Set(arr))
}
```

36.垃圾回收机制

可达性

简单地说，“可达性” 值就是那些以某种方式可访问或可用的值，它们被保证存储在内存中。

```
// user 具有对象的引用
let user = {
  name: "John"//此时可通过全局变量user拿到其name属性值John，
};
//如果 user 的值被覆盖，则引用丢失：
user = null;//现在 John 变成不可达的状态，没有办法访问它，没有对它的引用。垃圾回收器将丢弃 John 数据并释放内存。
```

1）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。

- 根：

  1. 有一组基本的固有可达值，由于显而易见的原因无法删除。例如:

  - 本地函数的局部变量和参数
  - 当前嵌套调用链上的其他函数的变量和参数
  - 全局变量
  - 还有一些其他的，内部的

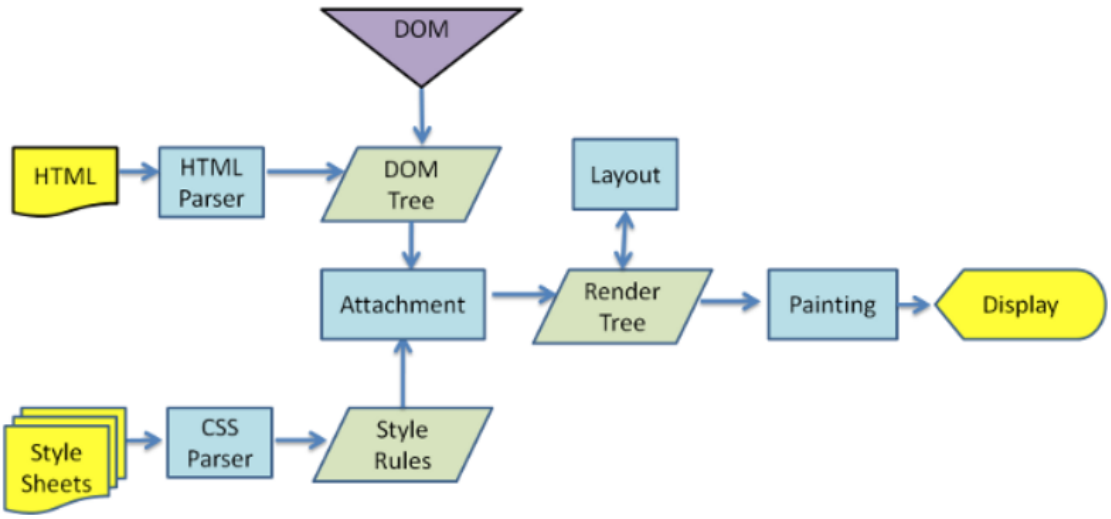
  这些值称为根。

2）如何检垃圾

一种算法是标记 **标记-清除** 算法，标记清除分两个阶段：标记和清除
标记阶段：从根节点遍历所有变量，将无法访问的变量打上标记
清除阶段：从根节点遍历所有变量，释放打上了标记的所有变量

37.重绘和回流

浏览器的渲染过程



- 解析HTML，生成DOM树，同时解析CSS，生成CSSOM树
- 将DOM树和CSSOM树结合，生成渲染树(Render Tree)
- 回流（Layout）：得到节点的几何信息（位置，大小）
- 重绘：根据渲染树以及回流得到的几何信息，得到节点的绝对像素
- 呈现

回流和重绘的区别

- 当布局和尺寸改变时，会发生回流和重绘
- 只有颜色发生变化时，只会发生重绘
- 回流一定会引起重绘，重绘不一定会引起回流

38.React的函数组件和类组件有什么区别?

区别	函数组件	类组件
是否有 `this`	没有	有
是否有生命周期	没有	有
是否有状态 `state`	没有(hook)	有

39.cookie、localStorage 和 sessionStorage 有什么区别?

共同点：都是保存在浏览器端、且同源的
区别：
1、cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递，而sessionStorage和localStorage不会自动把数据发送给**服务器**，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下
2、**存储大小**限制也不同，cookie数据不能超过4K，同时因为每次http请求都会携带cookie、所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大
3、数据**有效期**不同，sessionStorage：仅在当前浏览器窗口关闭之前有效；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie：只在设置的cookie过期时间之前有效，即使窗口关闭或浏览器关闭
4、**作用域**不同，sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面；localstorage在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的

40.MVVM和MVC的区别

MVVM：即Model-ViewModel-View的简写，模型至后端传递的数据，视图是所看到的页面，视图模型是mvvm模式的核心，它有两个方向：一、可以将模型转化为视图，就是将后端传递的数据转化成所看到的页面，实现方式是：数据绑定。二、还可以将视图转化为模型，即将所看到的页面转化为后端的数据。实习方式是dom事件监听。这两个方向都实现的，就叫做双向数据绑定。

MVC：Model-View-Controller的简写。 使用MVC的目的就是将M和V的代码分离。MVC是单向通信。也就是View跟Model，必须通过Controller来承上启下。

41.在 React 中如何实现实现父子组件、兄弟组件之间的传值?

1. 父传子

   父组件通过属性的方式传递参数，子组件通过props来接收父组件传递过来的参数

2. 子传父

父组件通过属性的方式传递一个自定义函数，子组件通过**this.props.事件名(参数)**的方式向父组件传递参数

```
子传父：父：<Child onHandleChild="函数"/>            父组件=>函数(参数){ }

        子：this.props.onHandleChild(传值)    //在子组件中执行这个函数,会传值到父件
```

3. 兄弟组件

   两个兄弟组件之间会有一个共同的父组件，我们都是结合父子传值的方式来实现兄弟之间的传值的，即先其中一个子组件（兄弟组件）向父组件传值，然后父组件接收到这个值之后再将值传递给另外一个子组件（兄弟组件）

42. 响应式和自适应的区别是什么?

响应式布局设计就是一个网站能够兼容多个终端，而不需要为每个终端做一个特定的版本；自适应布局设计指采取 百分比布局，宽度使用百分比，文字使用em或者rem，使得同一个页在不同尺寸的设备上呈现同样的页面。

43.Vue 中 watch 和 computed 的区别是什么?

computed： 在computed中定义的每一个计算属性，都会被缓存起来，只有当计算属性里面依赖的一个或多个属性变化了，才会重新计算当前计算属性的值，不支持异步。

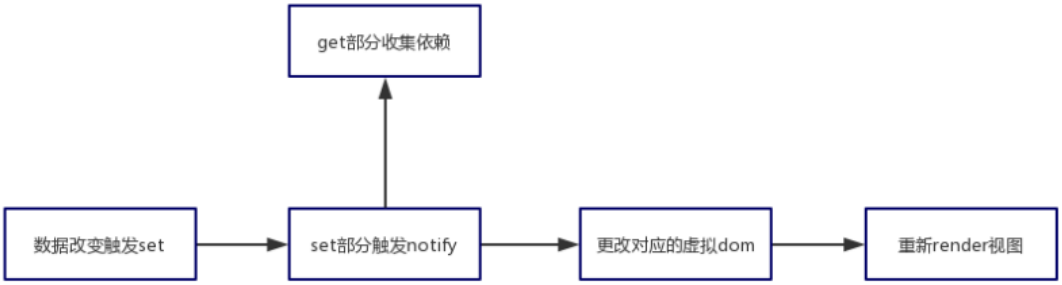
watch：属性监听器，一般用来监听属性的变化（也可以用来监听计算属性函数），并做一些逻辑，不支持缓存，支持异步。

44.在 React 的事件中，多次调用 setState 方法，render 会执行多少次?

执行一次。

原因：有一个函数**batchedUpdates**，这个函数会把isBatchingUpdates修改为true，而当React在调用事件处理函数之前就会调用这个batchedUpdates，造成的后果，就是**由React控制的事件（react事件）处理过程setState不会同步更新this.state。**而是将其放进队列。

45.简单描述 Vue 2.x 中响应式的原理。



当你把一个普通的 JavaScript 对象传入 Vue 实例作为 `data` 选项，Vue 将遍历此对象所有的属性，并使用 `Object.defineProperty` 把这些属性全部转为 `getter/setter`

```
function vue(){
```

```
  this.$data = {a: 1};//设置data
  this.el = document.getElementById('app'); //设置根节点
  this.dom = '';//虚拟dom
}

//往原型上绑定监听方法
vue.prototype.observe = function(obj){
    var self = this;
    var value;
    for(var key in obj){
        value = obj[key];
        if(typeof value === 'object'){ //如果是对象  需要递归循环执行此方法
           this.observe(value);
        }else{
          Object.defineProperty(this.$data,key,{
             get: function(){
              //依赖收集 这里略过 代替源码里的 depend
               return value;
             },
             set: function(newVal){
                value = newVal;
                self.render();//触发更新  代替源码里的dep.notify
             }
          })
        }
    }
}

vue.prototype.render = function(){
   this.dom = '我是' + this.$data.a;
   this.el.innerHTML = this.dom;
}
```

虚拟dom更改后再通过diff算法更新真实节点。

46.简单描述 JavaScript 中的事件循环。

javascript是单线程的语言，单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。

单线程的优劣

优势：
1、降低处理复杂性，简化开发。
2、作为用于预处理与用户互动的脚步语言，可以更加容易的处理状态同步的问题。
劣势：
无并发处理能力，任务处于I/O等待状态，导致CPU处理资源的浪费。
于是JavaScript语言将任务分为：**同步任务和异步任务。**

对于同步任务来说，按顺序执行即可；但是，对于异步任务，各任务执行的时间长短不同，执行完成的时间点也不同，主线程如何调控异步任务呢？这就用到了消息队列， 消息可以理解成注册异步任务时添加的回调函数。

Eventloop

人们把javascript调控同步和异步任务的机制称为**事件循环** 。

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"消息队列"。只要异步操作执行完成，就到消息队列中排队

3、一旦执行栈中的所有同步任务执行完毕，系统就会按次序读取消息队列中的异步任务，于是被读取的异步任务结束等待状态，进入执行栈，开始执行

4、主线程不断重复上面的第三步

47.什么是跨域？为什么浏览器要使用同源策略？你有几种方式可以解决跨域问题？了解预检请求嘛？

跨域，是指浏览器不能执行其他网站的脚本。它由于浏览器的同源策略所造成的，是浏览器对JavaScript实施的安全限制。同源是指协议、域名、端口三者相同

同源策略限制了一下行为：

- Cookie、LocalStorage 和 IndexDB （浏览器数据库）无法读取
- DOM 和 JS 对象无法获取
- Ajax请求发送不出去

解决跨域问题

1. jsonp跨域：动态的创建script标签，再去请求一个带参网址来实现跨域通信，只能实现get请求
2. document.domain+iframe跨域，这种跨域方式最主要的是要求域名相同
3. window.name+iframe跨域，
4. location.hash+iframe跨域

> location.hash和window.name都是差不多的，都是利用全局对象属性的方法，这两种方法和jsonp也是一样的，就是之恩够实现get请求。

5. CORS（主流的跨域解决方式）
    - 简单请求：对于简单请求，浏览器直接发出cors请求，就是在头信息之中加入一个origin字段，该字段用来说明本次请求来自哪个源（协议+域名+端口），服务器根据这个值，决定是否同意这次请求。

```
CORS请求默认不发送Cookie和HTTP认证信息，如果要把Cookie发送到服务器，需要指定

```
Access-Control-Allow-Credentials: true
```

还需要开发者在ajax中打开withCredentials属性
```

- 非简单请求

  非简单请求是那种对服务器有特殊要求的请求，比如请求方法是PUT或DELETE，或者Content-Type字段的类型是application/json。

  ### 预检请求

  非简单请求的CORS请求，会在正式通信之前，增加一次HTTP查询请求，称为**"预检"请求（preflight）**。

1.浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。

预检请求用的请求方法是OPTIONS，除了Origin字段，预检请求的头信息还包括两个特殊字段：

Access-Control-Request-Method：该字段是必须的，用来列出浏览器的CORS请求会用到哪些HTTP方法

Access-Control-Request-Headers：该字段是一个逗号分隔的字符串，指定浏览器CORS请求会额外发送的头信息字段，例如是X-Custom-Header

2.服务器收到有"预检"请求后，检查上述相关字段后，确认允许跨源请求，就可以做出回应。

区别：

> CORS与JSONP的使用目的相同，但是比JSONP更强大。JSONP只支持GET请求，CORS支持所有类型的HTTP请求。JSONP的优势在于支持老式浏览器，以及可以向不支持CORS的网站请求数据。

其他的跨域解决方案：WebSocket协议跨域、代理服务器跨域

48.什么是变量提升？什么是暂时性死区？

JavaScript在执行的前一刻会进行预编译，在预编译的过程中会将定义的变量的声明提升到所在作用域的顶部，但是只会提升变量的声明，而不会提升变量的赋值，并且，函数声明提升的优先级会高于变量声明提升，但是变量优先级高于函数（即都存在初始值，变量优先）

暂时性死区：let和const的特点之一就是存在暂时性死区。只要在块级作用域内存在let或者const，那么在声明之前使用这些变量就会报错。 ES6 规定暂时性死区和let、const语句不出现变量提升，主要是为了减少运行时错误，防止在变量声明前就使用这个变量，从而导致意料之外的行为。

暂时性死区的本质就是，只要一进入当前作用域，所要使用的变量就已经存在了，但是不可获取，只有等到声明变量的那一行代码出现，才可以获取和使用该变量。

49. == 和 === 的区别是什么？

==：该运算符称作相等，用来检测两个操作数的值是否相等，对于不同的数据类型会进行转换再比较

===：该运算符称作严格相等，既比较值又比较数据类型

50.如何正确的判断 this？箭头函数的 this 是什么？

在普通函数中，this指代的是调用该函数的对象，在构造函数中，this指代的是该构造函数所创建的实例，箭头函数本身是没有this的，它内部的this是在它声明时它所处作用域中的this。

51.new 的原理是什么？通过 new 的方式创建对象和通过字面量创建又什么区别？

在调用 new 的过程中会发生以下四件事

1. 新生成一个对象
2. 设置原型链
3. 改变this指向
4. 返回新对象

```
function new(func) {
    lat target = {};
    target.__proto__ = func.prototype;
    let res = func.call(target);
    if (typeof(res) == "object" || typeof(res) == "function") {
        return res;
    }
    return target;
}
```

`new Object()` 方式创建对象本质上是方法调用，涉及到在`proto`链中遍历该方法，当找到该方法后，又会生产方法调用必须的 堆栈信息，方法调用结束后，还要释放该堆栈，性能不如字面量的方式。通过对象字面量定义对象时，不会调用Object构造函数。

字面量创建对象，不会调用`Object`构造函数，简洁且性能更好；

52.你理解的原型是什么?

原型是一个可以被复制（或者叫克隆）的一个类，通过复制原型可以创建一个一模一样的新对象。通俗的说，原型就是一个**模板**，在设计语言中更准确的说是一个**对象模板**。

在JavaScript中原型是一个prototype对象，用于表示类型之间的关系。

原型（Person）定义了一些公用的属性和方法；利用原型（Person）创建出来的新对象实例会共享原型（Person）的所有属性和方法。

原型链就相当于原型对象创建过程的历史记录。

53.React 中 Component 和 PureComponent 的区别是什么?

`PureComponent`与`Component`在除了其`shouldComponentUpdate`方法的实现之外几乎完全相同。

对于`PureComponent`而言，当其`props`或者`state`改变之时，新旧`props`与`state`将进行浅对比（**shallow comparison**）。另一方面，`Component`默认的情况下其`shouldComponentUpdate`方法并不进行新旧`props`与`state`的对比。 因此相比于`Component`，`PureCompoent`的性能表现将会更好

56.Web安全

XSS：跨站脚本攻击，恶意攻击者往web页面里插入恶意可执行网页脚本代码

非持久型XSS

非持久型 XSS 漏洞，也叫反射型 XSS 漏洞，一般是通过给别人发送带有恶意脚本代码参数的 URL，当 URL 地址被打开时，特有的恶意代码参数被 HTML 解析、执行。

防护：

1 . Web 页面渲染的所有内容或者渲染的数据都必须来自于服务端。
2 . 尽量不要从 URL，document.referrer，document.forms 等这种 DOM API 中获取数据直接渲染。
3 . 尽量不要使用 eval, new Function()，document.write()，document.writeln()，window.setInterval()，window.setTimeout()，
innerHTML，document.creteElement() 等可执行字符串的方法。
4 . 如果做不到以上几点，也必须对涉及 DOM 渲染的方法传入的字符串参数做 escape 转义。

5 . 前端渲染的时候对任何的字段都需要做 escape 转义编码。 escape 转义的目的是将一些构成 HTML 标签的元素转义，比如 <，>，空格 等，转义成 <，>， 等显示转义字符。有很多开源的工具可以协助我们做 escape 转义。

持久型XSS

持久型 XSS 漏洞，也被称为存储型 XSS 漏洞，一般存在于 Form 表单提交等交互功能，如发帖留言，提交文本信息等，黑客利用的 XSS 漏洞，将内容经正常功能提交进入数据库持久保存，当前端页面获得后端从数据库中读出的注入代码时，恰好将其渲染执行。

主要注入页面方式和非持久型 XSS 漏洞类似，只不过持久型的不是来源于 URL，refferer，forms 等，而是来源于后端从数据库中读出来的数据。持久型 XSS 攻击不需要诱骗点击，黑客只需要在提交表单的地方完成注入即可，但是这种 XSS 攻击的成本相对还是很高。

防护：

1 . 后端在入库前应该选择不相信任何前端数据，将所有的字段统一进行转义处理。
2 . 后端在输出给前端数据统一进行转义处理。
3 . 前端在渲染页面 DOM 的时候应该选择不相信任何后端数据，任何字段都需要做转义处理。

CSRF跨站请求伪造攻击

攻击者可以盗用你的登陆信息，以你的身份模拟发送各种请求。

防护：

CSRF 的防御可以从服务端和客户端两方面着手，防御效果是从服务端着手效果比较好，现在一般的 CSRF 防御也都在服务端进行。

服务端的预防 CSRF 攻击的方式方法有多种，但思路上都是差不多的，主要从以下两个方面入手：

1 . 正确使用 GET，POST 请求和 cookie
2 . 在非 GET 请求中增加 token

57.同步加载、异步加载、预加载、延迟加载

同步加载： **同步模式**又称**阻塞模式**，会阻止浏览器的后续操作，相当于阻止了后续的文件的解析，执行等。

异步加载：**异步加载**又称**非阻塞加载**，浏览器在下载执行`js`的同时，还会继续进行后续页面的处理。主要有三种方式：

- async和await
- onload（把插入`script`的方法放在一个函数里面，然后放在`window.onload`方法里面执行，这样就解决了阻塞`onload`事件触发的问题）

延迟加载/懒加载：有些代码在某种特定情况下才需要，并不是一股脑子都加载出来了，这个时候就需要**延迟加载**

预加载： **预加载**是一种浏览器机制，使用浏览器空闲时间来预先下载/加载用户接下来很可能会浏览的页面/资源，当用户访问某个预加载的链接时，如果从缓存命中,页面就得以快速呈现

58.JavaScript中的事件代理/事件委托是什么?

事件代理（Event Delegation），又称之为事件委托。

是JavaScript中常用绑定事件的常用技巧。

“事件代理”即是把原本需要绑定在子元素的响应事件（click、keydown……）委托给父元素，让父元素担当事件监听的职务。事件代理的原理是**DOM元素的事件冒泡。**

事件传播

事件传播分成三个阶段：

- 捕获阶段：从window对象传导到目标节点（上层传到底层）称为“捕获阶段”（capture phase），捕获阶段不会响应任何事件；
- 目标阶段：在目标节点上触发，称为“目标阶段”；
- 冒泡阶段：从目标节点传导回window对象（从底层传回上层），称为“冒泡阶段”（bubbling phase）。

  事件代理即是利用事件冒泡的机制把里层所需要响应的事件绑定到外层；

事件委托的优点：

1. 可以大量节省内存占用，减少事件注册
2. 可以实现当新增子对象时无需再次对其绑定

59.prototype 和proto 区别是什么？对 JavaScript 中的原型链的理解，原型链的顶端是什么?

它们都有原型的意思，利用原型（Person）创建出来的新对象实例会共享原型（Person）的所有属性和方法。

prototype是函数所独有的，proto是所有对象都有的。对象的构造函数的prototype就等于该对象的proto，原型链就是一个查找方法的过程，如果对象本身没有定义某一方法或者属性，那么就会通过proto属性去其构造函数身上查找，而该构造函数本身又是一个对象，就这样往上查找，而最根部的原型就是一个Object，如果没找到会返回null。

60.Vue 和 React 的有什么异同?

- **监听数据变化的实现原理不同**
  - Vue通过 getter/setter以及一些函数的劫持，能精确知道数据变化。
  - React默认是通过比较引用的方式（diff）进行的，
- **数据流的不同**
  - Vue通过v-model指令实现双向数据绑定
  - React一直不支持双向绑定，提倡的是单向数据流，称之为onChange/setState()模式
- **组件通信的区别**
  - 在Vue中，父组件通过props向子组件传递数据；子组件通过事件向父组件发送消息；
  - React父组件通过props可以向子组件传递数据或者回调
- **模板渲染方式的不同**
  - Vue是在和组件JS代码分离的单独的模板中，通过指令来实现的
  - React是在组件JS代码中， 通过JSX，通过原生JS实现模板中的常见语法，更加纯粹原生
- **渲染过程不同**
  - Vue通过对比新旧虚拟节点的差异来进行渲染，不需要渲染整个组件树
  - React在应用的状态被改变时，全部子组件都会重新渲染。通过shouldComponentUpdate这个生命周期方法可以进行控制，但Vue将此视为默认的优化
- **Vuex和Redux的区别**
  - Vuex使用的数据是可变的，可以直接修改，在检测数据变化时原理跟Vue一样通过getter/setter来比较

- Redux使用的数据时不可变的，每次都时用新的state替换旧的state，在检测数据变化时，通过diff方式比较差异

总结： React更偏向于构建稳定大型的应用，非常的科班化。相比之下，Vue更偏向于简单迅速的解决问题，更灵活，不那么严格遵循条条框框。因此也会给人一种大型项目用React，小型项目用Vue的感觉。

61.什么是 JSX?

- React特有的，是一个 JavaScript 的语法扩展。具有 JavaScript 的全部功能。js里面可以写html代码
- 一些语法的改变。基本：{变量}，其次class为关键字 用className代替class htmlFor 代替for等，内联样式style用法的改变 style={{color："red"}}。双层括号的含义：js代码需要在{}中，style里面本身需要一个对象。

62.AJAX 的工作原理?

工作原理相当于在用户和服务器之间加了一个中间层（Ajax引擎），使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器,像一些数据验证和数据处理等都交给 Ajax引擎自己来做, 只有确定需要从服务器读取新数据时再由Ajax引擎代为向服务器提交请求。

63.深拷贝和浅拷贝分别是什么？什么时候需要深拷贝？如何实现深拷贝?

假设B复制了A，当修改A时，看B是否会发生变化，如果B也跟着变了，说明这是浅拷贝，拿人手短，如果B没变，那就是深拷贝，自食其力。

如果要复制一个对象，这个对象里面还有一个引用数据类型的对象，这种情况就需要使用深拷贝。

实现深拷贝的方式：

- 递归

```
function deepClone(obj){
    let objClone = Array.isArray(obj)?[]:{};
    if(obj && typeof obj==="object"){
        for(key in obj){
            if(obj.hasOwnProperty(key)){
                //判断ojb子元素是否为对象，如果是，递归复制
                if(obj[key]&&typeof obj[key] ==="object"){
                    objClone[key] = deepClone(obj[key]);
                }else{
                    //如果不是，简单复制
                    objClone[key] = obj[key];
                }
            }
        }
    }
    return objClone;
}
```

- **JSON对象的parse和stringify**

弊端： 无法处理**function**和 **undefined** （会丢失）、**RegExp**（得到空对象）、时间对象（返回字符串）、 循环引用，**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。

64.基本类型数据和引用类型数据在内存中如何进行存储的?

基本数据类型都是直接存储在内存中的内存栈上的，***数据本身的值就是存储在栈空间里面***。

引用类型的存储需要内存的栈区和堆区（堆区是指内存里的堆内存）共同完成，栈区保存该对象在堆内存的地址，而其值则保存在堆内存中。

65.数据类型判断

```
typeof [1,2,3]//object
typeof null//object
typeof undefined//undefined

3 instanceof Number//false
new Number(3) instanceof Number//true

toString.call(123)//[object Number]

console.log(fn.constructor === Function); //true 对象的constructor判断
```

66.JavaScript 异步解决方案的发展历程以及优缺点。

- 回调函数（callback）

  缺点：回调地狱，不能用try catch捕捉错误，不能return

- promise

  优点：解决了回调地狱

  缺点： 无法取消 Promise ，错误需要通过回调函数来捕获

- generator

  优点：可以控制函数的执行

- async/await

  优点：代码清晰，不用像 Promise 写一大堆 then 链，处理了回调地狱的问题

  缺点：await 将异步代码改造成同步代码，如果多个异步操作没有依赖性而使用 await 会导致性能上的降低