React

例子 react 原理

答：1. 虚拟DOM 2. 组件系统 3. 单向数据流 4. JSX 语法

1.虚拟DOM

vdom可以看作是一个使用javascript模拟了DOM结构的树形结构，相当于是真实dom和js之间的缓存。

2.react hooks 现实中应用场景 常用的 优点

代码可读性更强，原本同一块功能的代码逻辑被拆分在了不同的生命周期函数中，容易使开发者不利于维护和迭代，通过 React Hooks 可以将功能代码聚合，方便阅读维护
1、useState 保存组件状态
2、useEffect 处理副作用
3、useContext 减少组件层级
4、useReducer
在使用上几乎跟 Redux/React-Redux 一模一样，唯一缺少的就是无法使用 redux 提供的中间件。用法也很简单
5、useCallback 记忆函数
6、useMemo 记忆组件：
只有在第二个参数数组的值发生变化时，才会触发子组件的更新。
useCallback 的功能完全可以由 useMemo 所取代，
唯一的区别是：useCallback 不会执行第一个参数函数，而是将它返回给你，而 useMemo 会执行第一个函数并且将函数执行结果返回给你。所以在前面的例子中，可以返回 handleClick 来达到存储函数的目的。
所以 useCallback 常用记忆事件函数，生成记忆后的事件函数并传递给子组件使用。而 useMemo 更适合经过函数计算得到一个确定的值，比如记忆组件。
7、useRef 保存引用值：useRef 创建一个引用，不是拷贝
8、useImperativeHandle 透传 Ref
9、通过 useImperativeHandle 用于让父组件获取子组件内的
DOM
10、useLayoutEffect 同步执行副作用
会在 DOM 更新之后同步执行；useLayoutEffect 会在 render，DOM 更新之后同步触发函数，会优于 useEffect 异步触发函数。

3.高阶组件 啥东西 应用场景，优点，你在实际工作中有如何使用，哪些方面可能会用到

高阶组件是 React 中用于复用组件逻辑的一种高级技巧，高阶组件自身不是 React API 的一部分，它是一种基于 React 而形成的设计模式。
说的明白点就是：高阶组件是组件作为参数，返回一个新组件的函数。
例如 Redux 的 connect 与路由相关的高阶组件 ： withRouter

4.redux如何和组件链接数据呢， 如何通知组件更新呢 为啥redux要返回一个新的state. 以及如果异步请求呢 如何同步

1.store通过reducer创建了初始状态；
2.view通过store.getState()将store中保存的state挂载在了自己的状态上；
3.用户产生了操作，调用了actions 的方法；
4.actions的方法被调用，创建了带有标示性信息的action；
5.actions将action通过调用store.dispatch方法发送到了reducer中；

6.reducer接收到action并根据标识信息判断之后返回了新的state；
7.store的state被reducer更改为新state的时候，store.subscribe方法里的回调函数会执行，此时就可以通知view去重新获取state；

react-redux提供两个核心的api：
Provider：提供store
connect：用于连接容器组件和展示组件
1. Provider
根据单一store原则 ，一般只会出现在整个应用程序的最顶层。
2. connect
 redux通过connect高阶组件来连接数据
  // 因为包装器组件是通过检查新旧state是否相同来改变数据的，所以要返回一个全新的state

5.函数组件和类组件的区别

函数组件没有自己的状态，没有this，没有生命周期

- 类组件有自己的状态和生命周期钩子函数，也能使组件直接访问 store 并维持状态

6.react性能优化

1. 引用react的高阶组件react-loadable进行动态import。
2. 用shouldComponentUpdate， 避免重新渲染
3. 组件尽可能的拆分、解耦
4. 类组件在constructor中绑定this
5. 使用reactdomserver实现服务端渲染，更快速的渲染

7.react.createElement干啥的

react.createElement是用来创建react元素的，JSX 语法就是用 React.createElement()来构建 React 元素的。它接受三个参数，第一个参数可以是一个标签名，第二个参数为传入的属性，第三个参数作为组件的子组件

8.vue和react区别

vue数据双向绑定 、 html，css，js各有各的处理方式 、 通过mixin进行组件扩展（已被react废弃）

react单向数据流 一切皆是组件 、通过js来操作一切、通过HOC进行组件扩展

9.vue双向绑定原理

vue.js 则是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty() 来劫持各个属性的setter ， getter ，在数据变动时发布消息给订阅者，触发相应的监听回调。
优点：
用户在视图上的修改会自动同步到数据模型中去，数据模型中值的变化也会立刻同步到视图中去；
无需进行和单向数据绑定的那些相关操作；
在表单交互较多的场景下，会简化大量业务无关的代码。
缺点：
无法追踪局部状态的变化；

10.react的单数据流原理

单向数据流（Unidirectional data flow）方式使用一个上传数据流和一个下传数据流进行双向数据通信，两个数据流之间相互独立。单向数据流指只能从一个方向来修改状态
优点：
所有状态的改变可记录、可跟踪，源头易追溯；
所有数据只有一份，组件数据只有唯一的入口和出口，使得程序更直观更容易理解，有利于应用的可维护性；
一旦数据变化，就去更新页面(data-页面)，但是没有(页面-data)；
如果用户在页面上做了变动，那么就手动收集起来(双向是自动)，合并到原有的数据中。
缺点：
HTML 代码渲染完成，无法改变，有新数据，就须把旧 HTML 代码去掉，整合新数据和模板重新渲染；
代码量上升，数据流转过程变长，出现很多类似的样板代码；
同时由于对应用状态独立管理的严格要求(单一的全局 store)，在处理局部状态较多的场景时(如用户输入交互较多的“富表单型”应用)，会显得啰嗦及繁琐。

11.受控和非受控组件区别

在React中，所谓受控组件和非受控组件，是针对表单而言的
受控组件，表单元素的修改会实时映射到状态值上，此时就可以对输入的内容进行校验
受控组件必须要在表单上使用onChange事件来绑定对应的事件
非受控组件即不受状态的控制，获取数据就是相当于操作DOM
//常见受控组件　input、textarea、select获取value　checkbox、radio获取checked
//非受控组件　<input type="file" /> 始终是一个非受控组件，因为它的值只能由用户设置，而不能通过代码控制。

12.路由的两种方式hash/history 的区别；

Hash模式只可以更改#后面的内容， History 模式可以通过 API 设置任意的同源 URL
Hash模式只能更改哈希值，也就是字符串.History 模式可以通过 API 添加任意类型的数据到历史记录中
Hash模式无需后端配置，并且兼容性好。History 模式在用户手动输入地址或者刷新页面的时候会发起 URL 请求，后端需要配置 index.html 页面用于匹配不到静态资源的时候

13.react对页面render异常捕获：错误边界，看官网；

产生原因:过去，组件内的 JavaScript 错误会导致 React 的内部状态被破坏，并且在下一次渲染时 产生 可能无法追踪的 错误。

错误边界是一种 React 组件，这种组件可以捕获并打印发生在其子组件树任何位置的 JavaScript 错误，并且，它会渲染出备用 UI，而不是渲染那些崩溃了的子组件树。错误边界在渲染期间、生命周期方法和整个组件树的构造函数中捕获错误。
//注意错误边界仅可以捕获其子组件的错误，它无法捕获其自身的错误。

错误边界无法捕获以下场景中产生的错误：
事件处理（了解更多）
异步代码（例如 setTimeout 或 requestAnimationFrame 回调函数）
服务端渲染
它自身抛出来的错误（并非它的子组件）

14.react-router和react-router-dom的原理及区别；

react-router是跨平台的，内置通用组件和通用Hooks。 只提供了一些核心的API,如Router、Route、Switch等，但没有提供有关dom操作进行路由跳转的api；
react-router-dom是在react-router基础上提供了BrowserRouter、Route、Link、NavLink等api，可以通过dom操作触发事件控制路由，而且依赖history库提供了两个浏览器端适用的BrowserRouter和HashRouter。
//一般都用的react-router-dom，一些常用的组件都封装好了。

15.react里面key的作用；

Keys是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。 使diff算法更高效
//在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。在 React Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染。此外，React 还需要借助 Key 值来判断元素与本地状态的关联关系，因此我们绝不可忽视转换函数中 Key 的重要性

16.diff算法

作用：计算出虚拟DOM中真正变化的部分，并只针对该部分进行原生DOM操作，而非重新渲染整个页面。
原理：在数据发生变化，是先根据真实DOM生成一个虚拟DOM ，当虚拟DOM 某个节点的数据改变后会生成一个新的 Vnode ，然后 Vnode 和 oldVnode 作对比，发现有不 一样的地方就直接修改在真实的DOM上，然后使 oldVnode 的值为 Vnode ，来实现更新节点。
阶段：1、patch（oldVnode，vnode） 2、updateChildren 3、createKeyToOldIdx
React的diff算法
（1）什么是调和？
将虚拟DOM树转换成真实DOM树的最少操作的过程 称为 调和 。
（2）什么是React diff算法？
diff算法是调和的具体实现。

17.setState

不可变值
1、React通过管理状态实现对组件的管理，通过setState更新组件的状态state，state数据发生改变会重新调用render方法来重新渲染UI。

2、setState本质是通过一个队列机制实现state更新的。 执行setState时，会将需要更新的state合并后放入状态队列，而不会立刻更新state，队列机制可以批量更新state。

3、setState既可能是同步的，也可能是异步的。
在React内部机制能检测到的地方（例如合成事件和生命周期函数里）setState就是异步的，在React检测不到的地方，例如setInterval,setTimeout里，setState就是同步更新的。

18.useref

useRef它可以用来获取组件实例对象或者是DOM对象.
除了传统的用法之外，它还可以“跨渲染周期(组件被多次渲染之后依旧不变的属性)”保存数据。

19.解释useEffect 的第二个参数传不同值的区别；

useEffect的第二个参数规则：可选，如果用上的话，这个参数必须是一个数组
第二个参数不传值：默认的行为，会每次 render 后都执行。
第二个参数是空数组：只运行一次，等同于类组件中的componentDidMount
第二个参数有一个值：值有变化就执行。
第二个参数有2个或2个值以上的数组，会比较每一个值，有一个不相等就执行。

20.reducer是纯函数吗？是，状态可追溯；

reducer必须是纯函数.
什么是纯函数：1.相同的输入永远返回相同的输出，2.不修改函数的输入值，3.不依赖外部环境状态。4.无任何副作用。

原因：如果在reducer中，在原来的state上进行操作，并返回的话，并不会让React重新渲染。 完全不会有任何变化!
比较两个js对象中所有的属性是否完全相同，唯一的办法就是深比较，然而，深比较在真实的应用中代码是非常大的，非常耗性能的，需要比较的次数特别多，所以一个有效的解决方案就是做一个规定，当无论发生任何变化时，开发者都要返回一个新的对象，没有变化时，开发者返回旧的对象，这也就是 redux 为什么要把 reducer 设计成纯函数的原因。

21.react-router的实现原理

一句话：实现URL与UI界面的同步。
react-router包含3个库，react-router、react-router-dom和react-router-native。
react-router-dom和 react-router-native都依赖react-router，所以在安装时，react-router也会自动安装，

22.react事件和原生浏览器事件

DOM事件模型分为捕获和冒泡。一个事件发生后，会在子元素和父元素之间传播。事件传播分为三个阶段：
捕获（Capture）：事件对象从window对象传递到目标对象的过程。
目标（target）：目标节点在处理事件的过程。
冒泡（Bubble）：事件对象从目标对象传递到window对象的过程。
React中的事件机制与原生的完全不同，事件没有绑定在原生DOM上，发出的事件也是对原生事件的包装。
React内部事件系统可以分为两个阶段：事件注册和事件触发。
事件注册：React将所有的DOM事件全部注册到document节点上。
事件触发：事件执行时，document上绑定事件会对事件进行分发，根据之前存储的类型和组件标识找到触发事件的组件。
React事件机制的优点：
减少内存消耗，提升性能，一种事件类型只在document上注册一次
统一规范，解决ie事件兼容问题，简化事件逻辑
对开发者友好

23.执行两次setState，render几次，会不会立即触发。答：批处理；

只会render一次。
●原因:(通俗点讲)React会将多个this.setState产生的修改放在一个队列里，缓一缓，攒在一起，觉得差不多了再引发一次更新过程。
●react为了提高整体的渲染性能，会将一次渲染周期中的state进行合并，在这个渲染周期中你对所有setState的所有调用都会被合并起来之后，再一次性的渲染，这样可以避免频繁的调用setState导致频繁的操作dom，提高渲染性能。具体的实现方面，可以简单的理解为react中存在一个状态变量isBatchingUpdates，当处于渲染周期开始时，这个变量会被设置成true，渲染周期结束时，会被设置成false，react会根据这个状态变量，当出在渲染周期中时，仅仅只是将当前的改变缓存起来，等到渲染周期结束时，再一次性的全部render。

24.PureComponent与Component的区别

为什么要使用pureComponent?
- 当使用component时，父组件的state或prop更新时，无论子组件的state、prop是否更新，都会触发子组件的更新，这会形成很多没必要的render，浪费很多性能。

PureComponent与Component的区别？
- PureComponent自带通过props和state的浅对比来实现 shouldComponentUpate()，而Component没有。

浅比较（shallowEqual）：是react源码中的一个函数，它代替了shouldComponentUpdate的工作，只比较外层数据结构，只要外层相同，则认为没有变化，不会深层次比较数据。

pureComponent的优缺点？
- 优点：不需要开发者使用shouldComponentUpdate就可使用简单的判断来提升性能；
- 缺点：由于进行的是浅比较，可能由于深层的数据不一致导致而产生错误的否定判断，从而导致页面得不到更新。

为什么使用pureComponent可以提升性能？
- 主要在于pureComponent可以减少不必要的render，从而提高了性能，另外就是，不需要再手写shouldComponentUpdate里面的代码，从而节省了代码量；

25.react生命周期。初始化会执行那些 如果setState更新数据又会执行哪些呢 ajax请求在那个周期

- 初始化阶段：
  - getDefaultProps:获取实例的默认属性
  - getInitialState:获取每个实例的初始化状态
  - componentWillMount：组件即将被装载、渲染到页面上
  - render:组件在这里生成虚拟的 DOM 节点
  - componentDidMount:组件真正在被装载之后 // 发送ajax请求
- 运行中状态(setState更新数据时执行)：
  - componentWillReceiveProps:组件将要接收到属性的时候调用
  - shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）
  - componentWillUpdate:组件即将更新不能修改属性和状态
  - render:组件重新描绘
  - componentDidUpdate:组件已经更新
- 销毁阶段：
  - componentWillUnmount:组件即将销毁

26.webpack打包 原理 那些插件

打包原理：webpack是把项目当作一个整体，通过给定的一个主文件，webpack将从这个主文件开始找到你项目当中的所有依赖的文件，使用loader来处理它们，最后打包成一个或多个浏览器可识别的js文件

27.git经常用到的命令，至少五个，并且要说出每个命令是干什么的。 (git add .) (git commit -m “”) (git push origin **) (git clone) (git branch) (git checkout)

```
git helper -a # 查看全部git子命令
git -version # 查看git版本
git checkout -b "新建分支名" # 创建一个分支并切换到新创建的分支
git branch "新建分支名" # 创建分支
git branch # 查看本地所有分支
git branch -r # 查看远程所有分支
git branch -a # 查看本地和远程分支
git branch -d "某分支名" # 删除某分支
git checkout "某分支名" # 切换到某分支
git switch "某分支名" # 切换到某分支
git status # 查看状态
git add "文件名" # 将某个文件存入暂存区
```

```
git add . # 将所有文件存入暂存区
git commit -m "备注信息" # 提交到仓库
git diff # 查看变更 工作区和暂存区的差别比对
git push <远程主机名><本地分支>:<远程分支> # 完整写法
git push origin master # 将本地的master分支推送到远程的master分支，如果master不存在 则会创建master分支
git push <新建分支名> # 将新建分支推送到远程分支
git push origin :master # 如果忽略本地分支，则推送了一个空分支，相当于删除了分支。
                        # 等同于git push origin --delete <要被删除的分支名>
git push # 将本地分支推送到远程分支，如果当前分支和远程分支之间存在追踪关系，则本地分支和远程分支都可以省略
git push origin --delete <要删除的分支名> # 删除远程分支
git fetch origin master # 将远程分支下拉到本地
git pull <远程分支地址>  <远程分支名>:<本地分支名> # 完整写法
                                  # 例如，git pull origin next : master .将远程分支与next分支合并。
git pull origin master # 获取远程分支,并于当前分支合并
git fetch origin master # 获取远程分支master到本地,不合并
git clone "远程地址" # 将远程分支克隆到本地
git merge "分支名" # 把现有分支合并到分支上
git reset HEAD file # 文件 add后,撤销修改
git remote add origin git项目地址 # 本地仓库和远程仓库建立连接
git remote -v # 查看远程关联的地址
git remote remove origin # 移除远程关联
```

JS

28.浏览器跨域问题，如何解决 jsonP原理

同源策略：
- 协议相同
- 域名相同
- 端口相同

- Websocket

jsonP原理 ：就是利用 标签没有跨域限制的漏洞。通过 标签指向一个需要访问的地址并提供一个回调函数来 接收数据当需要通讯时。

//JSONP 使用简单且兼容性不错，但是只限于 get 请求。

CORS ： 需要浏览器和后端同时支持。IE 8 和 9 需要通过 XDomainRequest 来实现。浏览器会自动进行 CORS 通信，实现 CORS 通信的关键是后端。只要后端实现了CORS，就实现了跨域。

//谷歌插件cors

如果面试官问：“CORS为什么支持跨域的通信？”

答案：跨域时，浏览器会拦截Ajax请求，并在http头中加Origin。

document.domain：该方式只能用于二级域名相同的情况下，只需要给页面添加 document.domain = 'test.com' 表示二级域名都相同就可以实现跨域

29.深浅拷贝 优缺点 json.parse、 for循环递归方法拷贝

浅拷贝：

ES6 为 Object 对象新增一个方法 Object.assign(target, source1, source2)。第一个参数是目标对象，后面的是源对象。这个方法会把源对象上面的可枚举属性拷贝到目标对象上。但是这种拷贝属于浅拷贝。 //展开运算符 ... 赋值运算符 =

深拷贝：

对象的深拷贝可以通过 JSON 对象的两个方法，一是将 js 对象转成字符串对象的 JSON.stringify 方法，另一个是将 js 字符串对象转换成 js 对象的 JSON.parse 方法。

//插件使用 lodash 的深拷贝函数。

30.promise用法以及相关原理 用法 有那些API 例如问你promise.all怎么实现的呢，如果让你写一个呢

promise是异步编程的一种解决方案，解决多个异步方法串行的问题，比如回调地狱等.所谓的promise，简单地说就是一个容器，里面保存着某个未来才会结束的事件，从语法说promise是一个对象，从他可以获取异步操作的消息。promise提供统一的api，各种操作都可以用相同的方法进行处理.
1. 对象的状态不收外界影响,promise对象代表一个异步操作，有三种状态pending(进行中)，fulfilled(已成功)和rejected(已失败)。只有一步的操作结果，可以决定当前是哪一种状态,任何其他操作都无法改变这个状态。

2. 一旦状态改变，就不会在编，任何时候都可以得到这个结果。promise对象的状态改变只有两种可能：从pending变为fulfilled或从pending变为rehected.
API:
// Promise.prototype.catch()
catch() 方法返回一个Promise，并且处理拒绝的情况。它的行为与调用 Promise.prototype.then(undefined, onRejected) 相同
- **在异步函数中抛出的错误不会被catch捕获到**
- **在resolve()后面抛出的错误会被忽略**

//Promise.resolve()
- resolve一个数组
- resolve另一个promise

//Promise.reject()
返回一个带有拒绝原因参数的Promise对象

// Promise.all()
返回一个新的promise，只有所有的promise都成功才能成功，只要有一个失败了就直接失败

// Promise.race()
返回一个promise，一旦迭代器中的某个promise解决或拒绝，返回的 promise就会解决或拒绝。

31.前端性能优化，别总说些老掉牙的大家都知道的

SEO （Search Engine Optimization）**：直译为搜索引擎优化。是一种方式：利用搜索引擎的规则提高网站在有关搜索引擎的自然排名。
目的是：为网站提供生态式的自我营销的解决方案，让其在行业内占据领先地位，获得品牌效益。
- 简化代码结构，更利于搜索引擎分析抓取有用内容
- 重要内容优先加载
- 语义化 HTML 代码 ，符合 W3C 标准
- 重要内容不适用 js 输出
- 少用 iframe ：搜索引擎不会抓取 iframe 中的内容
- 图片加 alt 属性
- 适用网站统计功能优化
- 每个页面只出现一个 h1 标签，h2 可以出现多次

页面级优化:
- 使用CDN
- 减少 HTTP 请求数
- 设计从实现层面简化页面
- 合理设置 HTTP 缓存
- 资源合并与压缩
- CSS sprites
- 内联图像 雪碧图
- 图片懒加载
- 将外部脚本置底
- 预渲染

- 懒执行
- 懒加载 JavaScript
- 将 CSS 放在 head 中

32.es6 （新特性） 多说流畅

- 解构赋值
- 增强对象的字面量
- 箭头函数
- Promise
- 块作用域变量声明**let**和 **const**
- Class类
- Moudles(模块)
- Symbol()符号
- Generators （生成器）
- 新的数据构造对像MAP和set

33.css 定位布局 flex 具体属性

```
 display:flex 设置为flex布局
 flex-direction 主轴方向
 flex-wrap 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 主轴对齐方式
 align-items  交叉轴对齐方式
 align-content 多根轴线对齐方式
 项目属性
order
flex-grow
flex-shrink
flex-basis
flex
align-self
```

34.h5新特性

语义化标签：header、nav、footer 、strong、time 等：结构更好，更利于SEO的优化，可维护性更高，代码可读性较好

表单新元素：datalist选项列表、output不同类型的输出，比如脚本输出

多媒体标签：video 和 audio
canvas，video，webstorage，语义化标签，表单控件 等

35.判断数据类型的四种方法

typeof 判断简易数据类型，可以判断function ，但不能判断复杂数据类型
instanceof 因为A instanceof B 可以判断A是不是B的实例，返回一个布尔值，由构造类型判断出数据类型
根据对象的contructor判断
通过Object下的toString.call()方法来判断

36.事件队列事件循环机制 也就是宏任务微任务

宏任务包括 script ， setTimeout ， setInterval ， setImmediate 。
微任务包括 process.nextTick ， promise ， MutationObserver 。

首先执行同步代码，这属于宏任务
当执行完所有同步代码后，执行栈为空，查询是否有异步代码需要执行
执行所有微任务
当执行完所有微任务后，如有必要会渲染页面
然后开始下一轮 Event Loop，执行宏任务中的异步代码，也就是 setTimeout 中的回调函数
//这里很多人会有个误区，认为微任务快于宏任务，其实是错误的。因为宏任务中包括了 script ，浏览器会先执行一个宏任务，接下来有异步代码的话才会先执行微任务

37.定时器多的情况下 定时器会不准确 怎么处理

原因
这个其实就得提到js执行机制了，叫做事件循环Eventloop 循环机制中，异步事件 setInterval 到时后会把回调函数放入消息队列中Event Queue，主线程的宏任务执行完毕后依次执行消息队列的微任务，等微任务执行完了在循环回来执行宏任务。并且由于消息队列中存在大量任务，其他任务执行时间就会造成定时器回调函数的延迟，如果不处理则会一直叠加延迟
通过计算时差可以有效的解决

```
const _setInterval = (fn, delay, ...rest) => {
  let lastTime = Date.now();
  return setInterval(() => {
    let now = Date.now();
    if (now - lastTime >= delay) {
      lastTime = lastTime + delay;
      fn(...rest);
    }
  }, 1);
};

const timer = _setInterval(() => {
  console.log('执行了')
}, 500)
```

38.原型原型链

每个构造函数都有一个原型，每个原型对象又有一个construtor属性指向构造函数，每个实例都有__proto__指向原型对象，原型对象上的属性方法能被实例访问。

在JS中，用 __proto__ 属性来表示一个对象的原型链。当查找一个对象的属性时，JS会向上遍历原型链，直到找到给定名称的属性为止。

39.闭包 ~应用场景

闭包的特点
1.函数嵌套函数
2.函数内部可以引用外部的参数和变量
3.参数和变量不会被垃圾回收机制回收
因此闭包常会被用于
1可以储存一个可以长期驻扎在内存中的变量
2.避免全局变量的污染
3.保证私有成员的存在
那闭包又因为什么原因不被回收呢
简单来说，js引擎的工作分两个阶段，
一个是语法检查阶段，
一个是运行阶段。而运行阶段又分预解析和执行两个阶段。

40.call apply bind区别和实现原理

区别：

Call和apply的区别是传递的参数形式不一样　直接传参数　和传数组

Bind是只改变this指向但是不会调用方法

实现原理：

call是使用一个指定的this值和单独给一个或者多个参数来调用函数

apply和call的功能相同，只是参数形式不同

```js
Function.prototype.call = function(context, args1, args2, args3 ...)

Function.prototype.apply = function(context, [args1, args2, args3 ...])
```

bind是创建一个新的函数，在bind()方法被调用时，这个新函数的this会被bind的第一个参数指定，其余的参数将作为新函数的参数供调用时使用。

41.防抖 ~实现原理 闭包.定时器柯理化思想

即短时间内大量触发同一事件，只会执行一次函数，防抖常用于搜索框/滚动条的监听事件处理，如果不做防抖，每输入一个字/滚动屏幕，都会触发事件处理，造成性能浪费;实现原理为设置一个定时器，约定在xx毫秒后再触发事件处理，每次触发事件都会重新设置计时器，直到xx毫秒内无第二次操作，。

```
// func是用户传入需要防抖的函数
// wait是等待时间
const debounce = (func, wait = 50) => {
// 缓存一个定时器id
let timer = 0
// 这里返回的函数是每次用户实际调用的防抖函数
// 如果已经设定过定时器了就清空上一次的定时器
// 开始一个新的定时器，延迟执行用户传入的方法
return function(...args) {
if (timer) clearTimeout(timer)
timer = setTimeout(() => {
func.apply(this, args)
}, wait)
}
}
```

42.回流和重绘

重绘和回流会在我们设置节点样式时频繁出现，同时也会很大程度上影响性能。
重绘是当节点需要更改外观而不会影响布局的，比如改变 color 就叫称为重绘
回流是布局或者几何属性需要改变就称为回流。
回流必定会发生重绘，重绘不一定会引发回流。回流所需的成本比重绘高的多，改变父节点里的子节点很可能会导致父节点的一系列回流。
以下几个动作可能会导致性能问题：
改变 window 大小
改变字体
添加或删除样式
文字改变
定位或者浮动
盒模型

//并且很多人不知道的是，重绘和回流其实也和 Eventloop 有关。
1. 当 Eventloop 执行完 Microtasks 后，会判断 document 是否需要更新，因为浏览器是 60Hz 的刷
新率，每 16.6ms 才会更新一次。
2. 然后判断是否有 resize 或者 scroll 事件，有的话会去触发事件，所以 resize 和 scroll 事件也是至少 16ms 才会触发一次，并且自带节流功能。
3. 判断是否触发了 media query
4. 更新动画并且发送事件
5. 判断是否有全屏操作事件
6. 执行 requestAnimationFrame 回调
7. 执行 IntersectionObserver 回调，该方法用于判断元素是否可见，可以用于懒加载上，但是兼容性不好
8. 更新界面
9. 以上就是一帧中可能会做的事情。如果在一帧中有空闲时间，就会去执行 requestIdleCallback 回调。

43.节流

防抖是延迟执行，而节流是间隔执行，和防抖的区别在于，防抖每次触发事件都重置定时器，而节流在定时器到时间后再清空定时器,函数节流即每隔一段时间就执行一次，实现原理为设置一个定时器，约定xx毫秒后执行事件，如果时间到了，那么执行函数并重置定时器

```
// func是用户传入需要防抖的函数
// wait是等待时间
const throttle = (func, wait = 50) => {
// 上一次执行该函数的时间
let lastTime = 0
return function(...args) {
// 当前时间
let now = +new Date()
// 将当前时间和上一次执行函数时间对比
// 如果差值大于设置的等待时间就执行函数
if (now - lastTime > wait) {
lastTime = now
func.apply(this, args)
          }
     }
}
setInterval(
throttle(() => {
     console.log(1)
  }, 500),
  1
```

)
```

### 44.页面渲染的逻辑

```
1.HTML和CSS经过各自解析，生成DOM树和CSSOM树
2.合并成为渲染树
3.根据渲染树进行布局
4.最后调用GPU进行绘制，显示在屏幕上
```

45.浏览器缓存 强缓存 协商缓存

```
从缓存位置上来说分为四种，并且各自有优先级，当依次查找缓存且都没有命中的时候，才会去请求网络。
Service Worker
Memory Cache
Disk Cache
Push Cache
缓存策略
强缓存：不会向服务器发送请求，直接从缓存中读取资源，
协商缓存：就是强制缓存失效后，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程
```

46. http缓存 localstorage和sessionstorage

```
localstorage:除非被清理，否则一直存在    大小5M    不参与与服务器端通信
sessionstorage:只有在打开页面时存在，页面关闭就清理    大小5M    不参与与服务器端通信    //都存储在客户端
```

47.http协议 https http

```
主要的区别如下：
Https协议需要ca证书，费用较高。
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。
使用不同的链接方式，端口也不同，一般而言，http协议的端口为80，https的端口为443
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。
```

48.cookie 、sesstion和teaoken

```
cookie 存储在客户端： cookie 是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器下次向同一服务器再发起请求时被携带并发送到服务器上。
cookie 是不可跨域的： 每个 cookie 都会绑定单一的域名，无法在别的域名下获取使用，一级域名和二级域名之间是允许共享使用的（靠的是 domain）。

session :是另一种记录服务器和客户端会话状态的机制
session :是基于 cookie 实现的，session 存储在服务器端，sessionId 会被存储到客户端的cookie 中访问资源            接口（API）时所需要的资源凭证.

token 的组成： uid(用户唯一的身份标识)、time(当前时间的时间戳)、sign（签名，token 的前几位以哈希算法压缩成的一定长度的十六进制字符串）
特点：服务端无状态化、可扩展性好支持移动端设备.安全.支持跨程序调用
每一次请求都需要携带 token，需要把 token 放到 HTTP 的 Header 里
基于 token 的用户认证是一种服务端无状态的认证方式，服务端不用存放 token 数据。**用解析 token 的计算时间换取 session 的存储空间，从而减轻服务器的压力，减少频繁的查询数据库**
token 完全由应用管理，所以它可以避开同源策略
```
```

49.三次握手

客户端向服务端发送请求，服务端收到后给客户端发送返回值，客户端收到后再次给服务端发送表示客户端有接收能力

50.浏览器安全问题，https协议，https与http区别

http协议：是超文本传输协议，信息是明文传输。

https协议：是具有安全性的ssl加密传输协议，为浏览器和服务器之间的通信加密，确保数据传输的安全。

51.从输入网址到页面显示的过程

* URL输入 * DNS解析 * TCP连接 * 发送HTTP请求 * 服务器处理请求 * 服务器响应请求 * 浏览器解析渲染页面 * 连接结束

52.js会堵塞dom解析吗，css会堵塞dom解析吗

JavaScript 可以阻塞 DOM 的生成，也就是说当浏览器在解析 HTML 文档时，如果遇到`<script>`，便会停下对 HTML 文档的解析，转而去处理脚本。如果脚本是内联的，浏览器会先去执行这段内联的脚本，如果是外链的，那么先会去加载脚本，然后执行。在处理完脚本之后，浏览器便继续解析 HTML 文档。

css加载不会阻塞DOM树解析（异步加载时DOM照常构建）但会阻塞render树渲染（渲染时需等css加载完毕，因为render树需要css信息）

53.异步渲染和同步渲染

同步渲染：浏览器访问网址，浏览器创建新的tabpage，新的内存块，加载页面的全部资源并渲染全部资源。但只要有页面中的任何一个操作，就会从新的开端全部在创建请求渲染一次，浏览器自己控制的http

异步渲染：用之前已经渲染过的页面数据，与后台交互数据不需要重新来渲染页面，实现对页面的部分更新。自己来控制http

54.标准盒模型和怪异盒模型

标准盒模型：标准模式下总宽度=width+margin（左右）+padding（左右）border（左右）

怪异盒模型：怪异模式下总宽度=width+margin（左右）

55.HTTP请求返回状态码（至少说出常见的200,304,404）

消息：代表请求已被接收，需要继续处理。临时响应。
100 Continue 告知客户部分响应已被服务器接收，客户端应继续发送请求。
成功：服务器已经接收理解并接受请求。
200 (OK) 请求成功，返回想要的数据（正常状态）；

201 (Created) 已创建，请求成功且服务器创建了新资源；

202 (Accepted)已接受，但尚未处理；

203 (Non-Authoritative Information)非授权信息，服务器处理了请求，只是返回的信息来自于第三方；

204（No Content）无内容，服务器成功处理请求，但没有返回任何内容；

205（Reset Content)重置内容，同204，但要求请求者重置文档视图

206（Partial Content)服务器成功处理部分GET请求，

重定向：代表客户端采取进一步的操作才能完成请求；
301(Moved Permanently)永久移除，客户端请求的资源被永久移除到新位置，服务器返回响应时，（对GET/HEAD请求的响应）会自动将请求转向新URL；

302(Found Moved Temporarily)临时移除，服务器当前从不同位置响应请求，但请求者应继续请求原有位置来进行后续的请求；

304(Not Modified)未修改，客户端发送带有条件的GET请求且该请求被允许，文档并未改变时服务器返回304状态码；且返回响应中不包含网页内容；307(Temporary Redirect)临时重定向，请求的资源临时从不同的URL响应请求；

请求错误：表示客户端看起来可能发生了错误，妨碍了服务器的处理；
400(Bad Request)错误请求，语义错误，请求无法被服务器理解，或者请求参数有误；

401(Unauthorized)未授权，请求需要请求者验证；

403(Forbidden)禁止，服务器拒绝该请求；

404(Not Found)未找到，找不到请求网页；

服务器错误：代表服务器无法完成明显有效的请求；
500(Internal Server Error)服务器内部错误，服务器代码报错，无法完成请求；

502(Bad Gateway)错误网关，服务器作为网关或代理，从上游服务器收到无效响应；

503(Service Unavailable)服务器不可用，服务器目前无法使用（由于超载或停机维护），通常，这只是暂时状态；

56.get 和post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。

57.哪些操作会造成内存泄露（至少三个）

1、全局变量引起的内存泄露
2、闭包引起的内存泄露：慎用闭包
3、dom清空或删除时，事件未清除导致的内存泄漏
3、循环引用带来的内存泄露

58.rgba和opacity的区别

opacity属性的值，可以被它的子元素继承，给父级div设置opacity属性，那么所有的子元素都会继承这个属性，并且，这个元素和它的继承这个属性的所有子元素的所有内容透明度都会改变。
rgba设置的元素，只对这个元素的背景色有改变，并且，这个元素的后代不会继承这个属性

59.link和inport的区别

1.从属关系的区别
    @import是 CSS 提供的语法规则，只有导入样式表的作用；link是HTML提供的标签，不仅可以加载 CSS 文件，还可以定义 RSS、rel 连接属性等。
2.加载顺序的区别
    加载页面时，link标签引入的 CSS 被同时加载；@import引入的 CSS 将在页面加载完毕后被加载。
3.兼容性的区别
    @import是 CSS2.1 才有的语法，故只可在 IE5+ 才能识别；link标签作为 HTML 元素，不存在兼容性问题。
4.DOM可控性的区别
    可以通过 JS 操作 DOM ，插入link标签来改变样式；由于 DOM 方法是基于文档的，无法使用@import的方式插入样式。

60.伪数组转成真数组的方法（至少两种方法）

```
// 1.  ES6中数组的新方法 from()
 function test(){
2   var arg = Array.from(arguments);
3   arg.push(5);
4   console.log(arg);//1,2,3,4,5
6 }
7 test(1,2,3,4);

 // 2.arguments  无影响,只是限制了html对象
function test (){
2     arguments.__proto__ = Array.prototype;
3     arguments.push(10)
4     console.log(arguments)
5 }
6     test(1,2,3,4)
```

61.js的垃圾回收机制是什么原理

垃圾收集机制的原理很简单：找出那些不再继续使用的变量，然后释放其占用的内存，垃圾收集器会按照固定的时间间隔，或代码执行中预定的收集时间，周期性地执行这一操作
局部变量只在函数执行的过程中存在。而在这个过程中，会为局部变量在栈(或堆)内存上分配相应的空间，以便存储它们的值。然后在函数中使用这些变量，直到函数执行结束。此时，局部变量就没有存在的必要了。因此可以释放它们的内存以供将来使用。在这种情况下，很容易判断变量是否还有存在的必要；但并非所有情况下都这么容易就能得出结论
垃圾收集器必须跟踪哪个变量有用哪个变量无用，对于不再有用的变量打上标记，以备将来收回其所占用的内存。用于标识无用变量的策略通常有标记清除和引用计数两种

62.promise和async的区别：async可以return await的promise对象；

函数前面多了一个aync关键字。await关键字只能用在aync定义的函数内。async函数会隐式地返回一个promise，该promise的reosolve值就是函数return的值。

63.null和undefined区别；typeof null typeof undefined 结果是什么；== ===的结果是否相同

```
//(1)
null：Null类型，代表“空值”，代表一个空对象指针，使用typeof运算得到“object”，所以可以认为它是一个特殊的对象值。
undefined：Undefined类型，当一个声明了一个变量未初始化时，得到的就是 undefined。
//(2)
typeof undefined   //  undefined
typeof null        // object
//(3)
null == undefined   // true
null === undefined   // false
typeof null == typeof undefined    //false
typeof null === typeof undefined    //false
```

64.map, filter,reduce 用法区别；

```
.map()
对数组的每个元素都遍历一次，同时返回一个新的值。返回的这个数据的长度和原始数组长度是一致的。
.filter()
如果 callback 函数返回 true，这个元素将会出现在返回的数组中，如果返回 false 就不会出现在返回数组中。
.reduce()
直观的返回数组里面指定的一个值或者对象
```

65.css垂直水平居中方式，最好能打出来3种或以上；

```
1. position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);
2.#father{
         position:relative;
       }
   #son{
         position: absolute;
         left: 0;
         top: 0;
         right: 0;
         bottom: 0;
         margin:auto;
        }
3.  父元素必须给高 dispiay:flex；align-item:center；justify-connect
```

66.position的值有哪些；讲清楚；

```
答：（1）position:absolute；【绝对定位：基于自己最近的一个非默认定位的父元素进行定位】
      （2）position:relative；【基于自己本身应该出现的位置进行偏移】
      （3）position:fixed；【基于浏览器进行定位】
      （4）position:static；【从上往下，从左往右，流式布局】
```

67.事件传播流程；捕获，目标，触发；

答：（1）捕获阶段：事件从根节点流向目标节点，途中流经各个DOM节点，在各个节点上触发捕获事件，直到达到目标节点。

（2）目标阶段：在此阶段中，事件传导到目标节点。浏览器在查找到已经指定给目标事件的监听器后，就会运行该监听器。

（3）事件冒泡： 当为多个嵌套的元素设置了相同的事件处理程序，它们将触发事件冒泡机制。在事件冒泡中，最内部的元素将首先触发其事件，然后是栈内的下一个元素触发该事件，以此类推，直到到达最外面的元素。如果把事件处理程序指定给所有的元素，那么这些事件将依次触发。

68.es6 Map与Object的区别；

答：Map类继承了Object，并对Object功能做了一些拓展，Map的键可以是任意的数据类型。

Map实现了迭代器，可用for…of遍历，而Object不行。

Map可以直接拿到长度，而Object不行。

填入Map的元素，会保持原有的顺序，而Object无法做到。

Map可以使用省略号语法展开，而Object不行。

69.react添加富文本；

```
dangerouslySetInnerHTML={{ __html: this.state.newHtml }}
```

70.箭头函数和普通函数区别；this指向，不能作为构造函数，不能访问arguments

答：（1）箭头函数是匿名函数，不能作为构造函数，不能使用new。

（2）箭头函数没有原型属性。

（3）箭头函数通过 call()或apply()方法调用一个函数时，只传入了一个参数，对this并没有影响。

（4）箭头函数不能当做Generator函数,不能使用yield关键字。

（5）相比普通函数，箭头函数有更简洁的语法。

71.函数声明和函数赋值语句的差别；

答：赋值式函数需要给变量赋值,以var、let；

声明式是不需要给变量赋值，以function开头；

72.rem em vw

答：【px】:长度单位，网页设计常用的基本单位。像素px是相对于显示器屏幕分辨率而言的；

【em】:长度单位,相对于当前对象内文本的字体尺寸（参考物是父元素的font-size）,em的值是不固定的，会继承父级元素的字体大小；

【rem】:是CSS3新增的一个相对单位，rem是相对于HTML根元素的字体大小（font-size）来计算的长度单位。如果你没有设置html的字体大小，就会以浏览器默认字体大小，一般是16px；

【vw】:是相对视口的宽度而定的，长度等于视口宽度的1/100；假如浏览器的宽度为200px，那么1vw就等于2px；

73.移动端1px问题

原因：Retine屏的分辨率始终是普通屏幕的2倍，1px的边框在devicePixelRatio=2的retina屏下会显示成2px，在手机上缩小呈现时，导致边框太粗的效果

解决方式：

```
.scale-1px{
  position: relative;
  border:none;
}
```

```
.scale-1px:after{
  content: '';
  position: absolute;
  bottom: 0;
  background: #000;
  width: 100%;
  height: 1px;
  -webkit-transform: scaleY(0.5);
  transform: scaleY(0.5);
  -webkit-transform-origin: 0 0;
  transform-origin: 0 0;
}
```

74.判断两对象是否相等

1、通过JSON.stringify(obj)来判断两个对象转后的字符串是否相等
2、getOwnPropertyNames方法返回一个由指定对象的所有自身属性的属性名组成的数组。先进行长度的比较，然后进行遍历
3.Object.is(a,b)

手写：

```
function diff(obj1,obj2){
        var o1 = obj1 instanceof Object;
        var o2 = obj2 instanceof Object;
        // 判断是不是对象
        if (!o1 || !o2) {
            return obj1 === obj2;
        }

        //Object.keys() 返回一个由对象的自身可枚举属性(key值)组成的数组,
        //例如：数组返回下表：let arr = ["a", "b",
"c"];console.log(Object.keys(arr))->0,1,2;
        if (Object.keys(obj1).length !== Object.keys(obj2).length) {
            return false;
        }

        for (var o in obj1) {
            var t1 = obj1[o] instanceof Object;
            var t2 = obj2[o] instanceof Object;
            if (t1 && t2) {
                return diff(obj1[o], obj2[o]);
            } else if (obj1[o] !== obj2[o]) {
                return false;
            }
        }
        return true;
    }
```

75.设计模型 常见的

工厂模式、单例模式、外观模式、代理模式、策略模式、观察者模式、迭代器模式、中介者模式、访问者模式

76.登录原理 控制权限

登录原理：
每次你在网站的登录页面中输入用户名和密码时，这些信息都会发送到服务器。服务器随后会将你的密码与服务器中的密码进行验证。如果两者不匹配，则你会得到一个错误密码的提示。如果两者匹配，则成功登录。
控制权限：
首先在我前端页面中我会配置好所有的路由节点，每一个节点都会添加一个自定义属性用来标识当前页面的访问权限。每一次用户登录成功之后，服务器端会返回当前用户的权限信息，然后我根据返回的权限信息动态设置我的管理系统的导航菜单
我们除了客户端验证之外还有会服务器端验证，每一次调用接口的时候服务器端都会验证用户的的权限信息，如果没有权限就会返回401状态码，我在页面中弹提示

77.怎么实现页面实时更新

因为 HTTP 协议有一个缺陷：通信只能由客户端发起。
轮询：定时发送请求,响应请求
//轮询的效率低，非常浪费资源（因为必须不停连接，或者 HTTP 连接始终打开）。
websocket:最大特点就是，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，属于服务器推送技术的一种。 //看官网配置
特点：
（1）建立在 TCP 协议之上，服务器端的实现比较容易。
（2）与 HTTP 协议有着良好的兼容性。默认端口也是80和443，并且握手阶段采用 HTTP 协议，因此握手时不容易屏蔽，能通过各种 HTTP 代理服务器。
（3）数据格式比较轻量，性能开销小，通信高效。
（4）可以发送文本，也可以发送二进制数据。
（5）没有同源限制，客户端可以与任意服务器通信。

78.浏览器安全问题 XSS CSRF

XSS：跨站脚本攻击，指黑客往HTML文件或者DOM中注入恶意脚本，从而在用户浏览页面时利用注入的恶意脚本对用户实施攻击的一种手段
常见的注入方式：
一、存储型 XSS 攻击：
1.利用站点漏洞将一段恶意代码提交到 网站数据库中
2.用户向网站请求包含恶意代码的页面
3.当用户浏览该页面的时候，恶意脚本就会将用户的 Cookie 信息等数据上传到服务器
二、反射型 XSS 攻击：反射型 XSS 攻击过程中，恶意脚本属于用户发送给网站请求中的一部分，随后网站又把恶意脚本返回给用户，当恶意脚本在用户页面被执行时，就可以利用该脚本做一些恶意操作。
解决方法：
1.服务器对输入脚本进行过滤或转码
2.充分利用 CSP
限制加载其他域下的资源文件
禁止向第三方域提交数据
禁止执行行内脚本和未授权的脚本
提供上报机制，帮助尽快发现 XSS 攻击，以便尽快修复问题
3.使用 HttpOnly 属性来保护重要的 Cookie 信息（无法通过脚本读取 Cookie）
4.通过添加验证码防止脚本冒充用户提交危险操作，对于一些不受信任的输入，可以限制其输入长度，这样可以增大 XSS 攻击的难度。
//xss.js 插件

CSRF 攻击就是黑客利用用户登录状态，并通过第三方的站点来做一些坏事。
和 XSS 不同的是，CSRF 攻击不需要将恶意代码注入用户页面，仅仅利用服务器的漏洞和用户登录状态来实施攻击
解决方法：
主要的防护手段是提升服务器的安全性
加Token 验证
隐藏令牌：把 token 隐藏在 http 的 head头中

79.模块化开发 优点 es和common.js 区别

模块化开发：指文件的组织、管理、使用的方式。即把一个大的文件拆分成几个小的文件，他们之间相互引用、依赖
优点：1.可以加速渲染页面，所有资源加载的时间不会因为模块化而加速，但是模块化能加速渲染
2.避免命名冲突
3.代码重用高
4.思路更为清晰，降低代码耦合

```
es和common.js  区别
1、es 是静态引入，编译时引入  //异步
es6 {
    export: '可以输出多个，输出方式为 {}' ,
    export default : ' 只能输出一个 ，可以与export 同时输出，但是不建议这么做',
    解析阶段确定对外输出的接口，解析阶段生成接口，
    模块不是对象，加载的不是对象，
    可以单独加载其中的某个接口（方法），
    静态分析，动态引用，输出的是值的引用，值改变，引用也改变，即原来模块中的值改变则该加载的值也改变，
    this 指向undefined
}
2、common.js 动态引入，执行时引入  //同步导入
commonJS {
     module.exports =  ...   :       '只能输出一个，且后面的会覆盖上面的' ,
    exports. ...  : ' 可以输出多个',
    运行阶段确定接口，运行时才会加载模块，
    模块是对象，加载的是该对象，
    加载的是整个模块，即将所有的接口全部加载进来，
    输出是值的拷贝，即原来模块中的值改变不会影响已经加载的该值，
    this 指向当前模块
}
```

80.继承 共四个 原理 不带class

1、原型链继承
核心：将父类的实例作为子类的原型
2、构造继承
核心：使用父类的构造函数来增强子类实例，等于是复制父类的实例属性给子类（没用到原型）
3、实例继承
核心：为父类实例添加新特性，作为子类实例返回
4、组合继承
核心：通过调用父类构造，继承父类的属性并保留传参的优点，然后通过将父类实例作为子类原型，实现函数复用
5、寄生组合继承
核心：通过寄生方式，砍掉父类的实例属性，这样，在调用两次父类的构造的时候，就不会初始化两次实例方法/属性，避免的组合继承的缺点

81.new 操作符调用构造函数具体做了什么

1、创建一个新对象
2、将构造函数的作用域赋给新对象（因此this指向了这个新对象）
3、执行构造函数中的代码（为这个新对象添加属性）
4、返回新对象

82.for in for of 区别 返回值

一般用for in遍历对象 、//返回键　　遍历数组返回下标
for of 遍历数组 ； //返回每一项值
也不是说不能用for in遍历数组、只是用for in遍历数组的时候会遍历数组所有的可枚举属性、也包括它的原型方法、也就是说for in遍历的是数组的索引（即键名），而for of遍历的是数组元素值；for of的话不能遍历对象、因为没有迭代器对象（注：迭代器，提供一种访问一个集合对象各个元素的途径，同时又不需要暴露该对象的内部细节）

for in 返回对象的key或者数组、字符串的下标； for of和forEach一样直接返回值

83.BFC 原理 怎么清除 怎么创建bfc

BFC是块级格式化上下文；

原理：感觉只需要回答：BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会穿透去影响到外面的元素；属于同一个BFC的两个相邻Box的margin会发生重叠
（具体注：1）内部的Box会在垂直方向，一个接一个地放置。
2）Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠
3）每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。
4）BFC的区域不会与float box重叠。
5）BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。
6）计算BFC的高度时，浮动元素也参与计算）

清除：　1）在父元素中添加一个属性：overflow:hidden
2）在浮动的盒子之下再放一个标签，在这个标签中使用clear:both

创建一个BFC： 1）根元素
2）float属性不为none
3）position不为static和relative
4）overflow不为visible
5）display为inline-block, table-cell, table-caption, flex, inline-flex

84.浏览器hack 兼容IE

浏览器hack就是使代码在功能上符合新的需求、针对某种浏览器进行样式设置，从而达到所有浏览器显示的效果一致，这种标识不同浏览器的方法就是 hack

兼容IE：兼容高版本IE （？？？不太确定）
如果系统只支持低版本的IE，但是用户的电脑的IE版本比较高，可以限定浏览器对文档的解析到某一特定版本，或者将浏览器限定到一些旧版本的表现中。可以用如下的方式：

```
<meta http-equiv="x-ua-compatible" content="IE=EmulateIE9" >
<meta http-equiv="x-ua-compatible" content="IE=EmulateIE8" >
<meta http-equiv="x-ua-compatible" content="IE=EmulateIE7" >
```

兼容低版本IE
系统兼容低版本IE比较困难，能做到的是指定浏览器按照最高的标准模式解析页面。主要是用来解决有些用户的电脑明明用的是IE8、IE9，但是确实用的IE7的文档模式。

```
<meta http-equiv="x-ua-compatible" content="IE=edge" >
```

85.不同分辨率的兼容问题

利用媒体查询判断屏幕的分辨率、然后进行响应调整

86.禁止双指缩放

移动端开发时、给页面头部添加一个meta标签、在标签内添加上 user-scalable = no,initial-scale=1,maximum-scale=1, minimum-scale=1, 四个属性即可（代码如下：<meta name="viewport" content="initial-scale=1,maximum-scale=1, minimum-scale=1, user-scalable=no">）

但有一些移动浏览器为了更好的用户体验、并没有遵循这个开发者禁止双指缩放的指定，比如：Safari、UC、QQ浏览器也就是说设置这四个属性已经实现不了这个功能了、这是可以通过 **touchmove** 事件判断多个手指（**touches.length**），并通过阻止事件冒泡 **event.preventDefault()** 来实现（代码如下：）

```
document.documentElement.addEventListener('touchmove', function(event) {
 if (event.touches.length > 1) {
 event.preventDefault();
 }
 }, false);
```

但是在我们多次双指操作之后、还是会突破限制按照用户的意愿进行缩放、所以要通过web代码完全实现禁止双指缩放、暂时还不能实现。

88.数组去重 5种

1、使用ES6中的set方法
2、indexOf()方法
3、利用数组的sort()方法（相邻元素比对法）
4、利用数组的includes
5、利用双重for循环、再用splice方法去重（ES5常用）
、函数递归

89.移动端开发注意事项

1. 页面布局
   1) 适配(rem)，屏幕特别大的时候字体和页面都会变得很大，显的很奇怪(山寨)。监听window的resize事件，动态改变html标签的字体大小
   2) flex+百分比布局（推荐这个）

2. 移动端300ms延迟问题
   在移动端中为了解决滑动事件和点击事件
   fastclick.js是一个js插件，用来解决这个问题
   或者可以使用移动端事件touch,tap

3.一些情况下对非可点击元素如(label,span)监听click事件，ios下不会触发，css增加cursor:pointer就搞定了。

4.CSS动画页面闪白,动画卡顿
   解决方法:
   1.尽可能地使用合成属性transform和opacity来设计CSS3动画，不使用position的left和top来定位
   2.开启硬件加速

```
 -webkit-transform: translate3d(0, 0, 0);
 -moz-transform: translate3d(0, 0, 0);
 -ms-transform: translate3d(0, 0, 0);
```

```
    transform: translate3d(0, 0, 0);
```

90.判断是不是数组

```
let a = [1,3,4];

Array.isArray(a)  //true
a instanceof Array; //true
a.constructor === Array;//true
Object.prototype.toString.call(a) === '[object Array]';//true
```