# 1、使用css水平垂直居中有几种实现方法?

- 已知高度可以使用 line-height 等于 高度实现垂直居中；使用 text-align:center实现水平居中
- display:flex; align-items:center;justify-content:center;
- 绝对定位的话，给父元素 设置定位属性 relative，子元素设置 absolute，然后设置 子元素 top:0;left:0;right:0;bottom:0;margin:auto；

# 2、flex布局

**flex布局即为弹性布局，也就是弹性盒模型，给元素开启弹性盒之后，子元素的float、clear、vertical-align等失效**

**flex-direction:决定主轴方向**

- row(默认值)：主轴为水平，起点在左端
- row-reverse：主轴为水平，起点在右端
- column：主轴为垂直方向，起点在上沿
- column-reverse：起点在下沿

**flex-wrap:是否换行**

- nowrap(默认) 不换行
- wrap 换行 首行在上
- wrap-reverse 换行 首行在下

**justify-content:子元素在主轴上的对齐方式**

- flex-start(默认):左对齐
- flex-end：右对齐
- center：居中
- space-between：两端对齐，子元素之间的间隔都相等
- space-around：两端对齐(但子元素不与父元素边框贴合)，子元素两侧的间隔相等;故子元素之间的间隔比子元素与父元素边框的间隔大一倍

**align-item:子元素在交叉轴上对齐方式**

- flex-start:交叉轴的起点对齐
- flex-end:交叉轴的终点对齐
- center:中点对齐
- baseline:子元素的第一行
- stretch(默认):若项目为设置高度或设置为auto，将占满整个父元素

**设置在子元素的属性**

- order:定义子元素的排列顺序，数值越小，排列越靠前，默认为0
- flex-grow:定义子元素的放大比例，默认为0，即如果存在剩余空间，也不放大；如果所有子元素的flex-grow属性为1，则它们将等分剩余空间
- flex-shrink:定义子元素的缩小比例，默认为1，如果空间不足，该子元素将缩小；如果所有子元素的flex-shrink属性都为1，当空间不足时，都将等比例缩小
- flex-basis：定义了在分配多余空间之前，子元素占据的主轴空间，默认值为auto
- flex:前三者的缩写，默认值为 0 1 auto。

# 3. rem的理解 移动端设计稿上的固定尺寸如何转化为实际的rem值

rem是以html里的font-size为基准值的长度单位，一般用于移动端适配

**px、em和rem的区别**

- em是相对父元素的字体大小，如果父元素的字体大小是14px，那么它子元素的2em就是28px，不同父元素的子元素的2em的实际大小是可能不同的。
- rem是相对于根元素，即html元素，如果html的字体大小是14px，那么在任何地方的2rem都是28px。
- px像素（Pixel）,相对长度单位。像素px是相对于显示器屏幕分辨率而言的。

**为什么要用rem?**

rem的出现及使用多用于移动端开发中，我们知道，移动端的设备宽度是不定的，如果我们使用固定的大小，那么在不同大小的设备上就会出现布局错乱、留白、残缺等现象的出现

**动态计算**

因为设备的大小我们是无法预知的，所以1rem的大小在不同设备上也就不同，如果我们在加载时知道了设备的宽度，我们就可以根据这个宽度来动态的计算出在该设备上1rem究竟应该是多少，然后设置到html元素上。

假设设计图宽度为designWidth，实际设备宽度为windowWidth，那么可以计算出实际的1rem = （designWidth/windowWidth）*100。这里的100为我们在设计图中设置的1rem的大小，也叫基准值

# 4. this指向

- 普通函数调用，此时this指向window
- 构造函数调用，this指向实例对象
- 对象方法调用，this指向该方法所属对象
- 事件绑定时，this指向绑定事件的对象
- 定时器函数，this指向window
- 箭头函数，this指向上下文

*更改this指向的三个方法*

1. call()方法
2. apply()方法
3. bind()方法

**三者区别**

- bind 会有一个返回值，返回值是个函数，因此要加上()才能调用；call，apply是没有返回值，当改变函数this指向的时候，不需要加()就会执行
- call 传递参数的时候是一个一个传递的， apply是传递一个数组

# 5. 搜索框输入时频繁 发出请求怎么处理?

**使用防抖处理**

**防抖实现原理**:

如果在500ms内频繁操作，则每次都会清除一次定时器然后重新创建一个。直到最后一次操作，然后等待500ms后发送ajax。

**实现方式**:

1. 防抖函数主要利用了闭包、高阶函数、定时器等特性
2. 首先我们可以定义一个高阶函数debounce,接受一个回调函数和延迟时间,在函数内部定义一个定时器变量,用于记录当前的定时器
3. debounce内部我们返回一个函数,函数执行的时候会检查当前是否有定时器,有的话会清除当前的定时器,重新赋值一个新的定时器给定时器变量,并设置定时器执行时间为用户传入的第二个参数

4. 定时器内部通过apply调用用户传入的函数,并传入执行上下文和arguments
5. 这样就能保证在规定时间内,不会高频的触发回调函数

**节流**

事件触发之后，在规定时间内，事件处理函数不能被再次调用，也就是说，在规定时间内，事件处理函数只能被调用一次，且是最先被触发调用的那次. 主要使用场景是滚动加载更多、搜索框的搜索联想功能

**闭包引起的内存泄漏:**

- 原因：闭包可以维持函数内局部变量，使其得不到释放
- 解决：将事件处理函数定义在外部，解除闭包，或者在定义事件处理函数的外部函数中，删除对dom的引用

# 6. ES6新特性

- let、const 声明关键字，块级作用域
- 箭头函数，this指向指向上下文，相较于传统函数语法更为简洁
- 解构赋值 可以快速获取对象中想要的值赋值给指定变量
- Set对象 可以自动排除重复项
- Promise、async、await用来解决异步回调问题，可以使异步代码写的跟同步代码一样

```
promise常用API
实例方法
.then()得到异步的正确结果
.catch()获取异常信息
.finally()成功与否都会执行(尚且不是正式标准)
对象方法
Promise.all()同时处理多个异步函数，当所有任务都执行完成后才得到结果
Promise.race()同时处理多个异步函数，只要有一个任务执行完成就能得到结果
```

- import export模块化
- class类语法糖

```
在es5中,生成实例对象的方法是通过构造函数
在es6中,通过class关键字,可以定义类,
类必须要用new调用否则会报错

constructor是类的默认方法,通过new命令生成对象实例时,自动调用这个方法
一个类必须有constructor方法,如果没有显示定义,一个空的constructor方法会默认添加
constructor默认返回实例对象(this),还可以指定返回另外一个对象

类的静态方法static关键字
所有在类中定义的方法,都会被实例继承,
如果在一个方法前,加上static关键字,就表示这方法不会被实例继承,而是直接通过类调用

extends关键字实现继承
子类必须在construct方法中调用super方法,否则新建实例时会报错,在子类的构造函数中,只有调用了
super之后,才可以使用this关键字,否则
会报错,super()只能在constructor中执行
super()当做函数使用的时候
1.执行父类的constructor方法
2.把this指向子类的实例
super当做对象使用(在普通函数中)
1.super指向父类的原型对象
2.方法里的this指向子类的实例
```

```
super当做对象使用(在私有方法中)
1.super指向父类
2.方法里的this指向子类
父类的静态方法,会被子类继承
```

## 数组方法

```
1. shift 删除数组中的第一个元素
2. pop 删除数组中的最后一个元素
3. unshift 增加元素在数组的前面
4. push 增加元素在数组的后面
5. map 循环，并返回新的数组
6. forEach 循环遍历
7. filter 过滤，筛选出数组中满足条件的，并且返回新的数组
8. contact 合并数组
9. find 查找出第一个符合条件中的数组元素
10. findIndex查找出第一个符合条件中的数组元素
11. flat 将多维数组转为一维数组
12. join 将数组转为字符串
13. reverse 颠倒数组中的顺序
14. every 检测数组中元素是否都符合条件，返回值为--boolean
15. some 检测数组中元素是否有满足条件的元素，返回值为---boolean
16. splice(start，n，添加元素) 开始位置，删除个数，添加元素
17. sort 排序
18. slice(start,end)选中[start,end)之间的元素
19. indexOf 查找值所在的位置
20. includes 查看数组中是否存在此元素
```

## 7. react hooks的理解

- 解决了什么问题?

  它可以让你在不编写class的情况下使用state以及其他的React特性，

  Hook 使你在无需修改组件结构的情况下复用状态逻辑

- 如何用hooks实现willunmount的效果

- useState是react自带的一个hook函数，它的作用就是用来声明状态变量。useState这个函数接收的参数是我们的状态初始值（initial state），它返回了一个数组，这个数组的第[0]项是当前当前的状态值，第[1]项是可以改变状态值的方法函数。
- useEffect

两个参数

1. 第一个参数是一个函数，是在第一次渲染以及之后更新渲染之后会进行的副作用。
   这个函数可能会有返回值，倘若有返回值，返回值也必须是一个函数，会在组件被销毁时执行。
2. 第二个参数是可选的，是一个数组，数组中存放的是第一个函数中使用的某些副作用属性。用来优化 useEffect
   如果使用此优化，请确保该数组包含外部作用域中随时间变化且 effect 使用的任何值。 否则，您的代码将引用先前渲染中的旧值。
   如果要运行 effect 并仅将其清理一次（在装载和卸载时），则可以将空数组（[]）作为第二个参数传递。 这告诉React你的 effect 不依赖于来自 props 或 state 的任何值，所以它永远不需要重新运行。

整合了类组件componentDidMount、componentDidUpdate、componentWillUnMount等钩子函数的能力，而且代码显得更加简洁.

## react 生命周期函数

- 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（新版本移除）
  - 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
- 运行中状态：
  - componentWillReceiveProps:组件将要接收到属性的时候调用（新版本移除）
  - 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（新版本移除）
  - 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

## react diff 原理（常考，大厂必考）

- 把树形结构按照层级分解，只比较同级元素。
- 给列表结构的每个单元添加唯一的 key 属性，方便比较。
- React 只会匹配相同 class 的 component（这里面的 class 指的是组件的名字）
- 合并操作，调用 component 的 setState 方法的时候, React 将其标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制.
- 选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

## 虚拟DOM原理

react 在内存中生成维护一个跟真实DOM一样的虚拟DOM 树，在改动完组件后，会再生成一个新得DOM，react 会把新虚拟DOM 跟原虚拟DOM 进行比对，找到两个DOM不同的地方，然后将之统一更新到真实DOM节点上

-优点：提高渲染速度

-缺点：由于多了一层虚拟DOM计算，就会比html渲染慢

## 了解 redux 么，说一下 redux吧，

- redux 是一个应用数据流框架，主要是解决了组件间状态共享的问题，原理是集中式管理，主要有三个核心方法，action，store，reducer，工作流程是 view 调用 store 的 dispatch 接收 action 传入 store，reducer 进行 state 操作，view 通过 store 提供的 getState 获取最新的数据。
- 新增 state,对状态的管理更加明确，通过 redux，流程更加规范了，减少手动编码量，提高了编码效率，同时缺点时当数据更新时有时候组件不需要，但是也要重新绘制，有些影响效率。一般情况下，我们在构建多交互，多数据流的复杂项目应用时才会使用它们

**redux解决了什么问题**

redux是为了解决react组件间通信和组件间状态共享而提出的一种解决方案

1. store：用来存储当前react状态机（state）的对象。connect后，store的改变就会驱动react的生命周期循环，从而驱动页面状态的改变

2. action: 用于接受state的改变命令，是改变state的唯一途径和入口。一般使用时在当前组件里面调用相关的action方法，通常把和后端的通信(ajax)函数放在这里
3. reducer: action的处理器，用于修改store中state的值，返回一个新的state值

**主要解决什么问题**

1、组件间通信

由于connect后，各connect组件是共享store的，所以各组件可以通过store来进行数据通信，当然这里必须遵守redux的一些规范，比如遵守 view -> aciton -> reducer的改变state的路径

2、通过对象驱动组件进入生命周期

对于一个react组件来说，只能对自己的state改变驱动自己的生命周期，或者通过外部传入的props进行驱动。通过redux，可以通过store中改变的state，来驱动组件进行update

3、方便进行数据管理和切片

redux通过对store的管理和控制，可以很方便的实现页面状态的管理和切片。通过切片的操作，可以轻松的实现redo之类的操作

## 应该在 React 组件的何处发起 Ajax 请求

在 React 组件中，应该在 componentDidMount 中发起网络请求。这个方法会在组件第一次“挂载”(被添加到 DOM)时执行，在组件的生命周期中仅会执行一次。更重要的是，你不能保证在组件挂载之前 Ajax 请求已经完成，如果是这样，也就意味着你将尝试在一个未挂载的组件上调用 setState，这将不起作用。在 componentDidMount 中发起网络请求将保证这有一个组件可以更新了

## 类组件(Class component)和函数式组件(Functional component)之间有何不同

- 类组件不仅允许你使用更多额外的功能，如组件自身的状态和生命周期钩子，也能使组件直接访问 store 并维持状态
- 当组件仅是接收 props，并将组件自身渲染到页面时，该组件就是一个 '无状态组件(stateless component)'，可以使用一个纯函数来创建这样的组件。这种组件也被称为哑组件(dumb components)或展示组件

## shouldComponentUpdate 是做什么的，（react 性能优化是哪个周期函数？）

shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新描绘 dom。

因为 dom 的描绘非常消耗性能，如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能

## React 性能优化的方法

在react方面的话，使用shouldcomponentupdate，purecomponent，usememo（我可以展开详谈），在浏览器方面的优化常用的有svg，减少http请求，防抖和节流也是可以减少http请求的，使用浏览器缓存（强缓存或者协商缓存），将script标签放在下面，或者是用defer和async来让它执行异步，使用http2，压缩代码，服务端渲染（可选，比较耗费服务端的性能），做cdn的静态资源托管也是可以优化性能的，使用iconfont图标来代替图片图标，用css3效果来替换那些阴影图片，减少重绘重排，使用事件委托（react的事件机制也是将事件添加给最上层的dom统一管理），降低css选择器的复杂性

## 项目流程

一、项目流程

说一下最近做的一个后台管理项目，该项目是一个后台管理系统，针对于心随礼动这个项目，做的一个后台管理，其中主要包括了首页，员工业绩信息展示的模块，商品分类模块，用户角色模块以及权限管理模块。（模块的功能实现与解释写在下方）

二、项目的亮点和难点

1、用户角色的权限管理，包括用户操作数据时候的权限，输入路径可以进入的解决办法
通过与后端配合，每个角色都有一个数据，需要后台返回来，后台返回的这个数据包含了用户所拥有权限的路由的url，将这个数据遍历出来，来控制左侧菜单栏内容的显示与隐藏，不过当时在做这个的时候，后期的测试有一个弊端，就是如果用户没有某个权限，直接输入url也是可以进入的，经过和后端的沟通，通过后端返回的数据直接遍历搭建路由，而不是控制显示和隐藏，

2、登录注册的拦截，以及登录的过期时间
用户在没有登陆的时候，重定向到登录页面，通过一个admin页面统一进行重定向，在登陆的时候后端人员会给我返回一个token值，我将这个token值通过cookie的方式存入到浏览器中，默认不给cookie设置过期时间，在浏览器关闭的时候，cookie会被清除，用户下次打开浏览器的时候需要重新登陆，还有一种解决办法需要后端人员配合，后端人员通过redis来存放token的过期时间，用户每次在发送http请求时，都会在请求头中携带这个token，服务端判断token，并可以为token设置延长的过期时间

3、better-scorll实现左右联动（做移动端）
在实现商品展示页面的时候有一个左右栏联动的效果，最初尝试用原生js编写，出了很多bug，有的卡顿，有的从头开始走，之后是用到better-scorll这个插件，想要实现左右联动的功能需要两个功能，一个是手指点击左边菜单栏，右边食物栏会联动到菜单栏下面的内容，另一个是手指滑动右边食物栏，左边菜单栏会随着右侧的滚动而相应出现active样式。先实现第一个目的，我需要在左侧的目标li上绑定click事件，点击事件触发move，还需要初始化两个better-scorll对象，一个左边的meun，一个右边的food。在move函数里面执行food.scrollToElement(le，time)

这个方法简直逆天：能food里的目标元素el在time毫秒内滚动到最顶部。el可以通过move（index）来获取，实现目的2的话，也就是右侧带动左侧的联动比较复杂一点，首先我需要定义一个数组，来记录一下food中list的高度（offectheight），通过scroll事件实时监听滚动位置，并且将位置付给scrollY,scrollY变化执行回调函数来获取索引，通过索引来动态添加class

4、对用户的数据之类的使用echart展示（vue项目）
最初在使用echart的时候，在控制删除图表的时候考虑的是v-if，不过后来了解到这个v-if是通过控制删除节点，创建节点，这样对性能的消耗太大，之后了解到echart有一个clear功能，清空绘画内容，清空后实例可用，因为并非释放示例的资源，释放资源我们需要dispose()，这样算是对项目做了一项性能的优化

5、在和后端交互的时候，商讨返回数据格式

6、ant-design3和ant-design4的时候遇到的难点
在使用ant4的时候碰到了一些坑，在ant3中图标的在安装完ant插件后就可以直接调用的，但是在ant4中图标需要另外的下载，下载iconfont，以及ant4在获取数据的时候也是有一些坑的，包括获取每一条数据的那个render函数

7、自己在项目中做一些性能的优化or代码的优化
在react方面的话，使用shouldcomponentupdate，purecomponent，usememo（我可以展开详谈），在浏览器方面的优化常用的有svg，减少http请求，防抖和节流也是可以减少http请求的，使用浏览器缓存（强缓存或者协商缓存），将script标签放在下面，或者是用defer和async来让它执行异步，使用http2，压缩代码，服务端渲染（可选，比较耗费服务端的性能），做cdn的静态资源托管也是可以优化性能的，使用iconfont图标来代替图片图标，用css3效果来替换那些阴影图片，减少重绘重排，使用事件委托（react的事件机制也是将事件添加给最上层的dom统一管理），降低css选择器的复杂性

8、一级分类和二级分类
请求数据：请求分类列表的时候，一级列表的id为0，二级列表的传入参数为自身ID，根据id名来对请求道的后端数据进行判断对应的级数数据。需用到async,await（一级列表数据和二级列表数据会）再componentDidMount生命周期里调用该方法。

点击查看子分类操作：再setState里接收到一级分类的name和id，因为setState时异步操作，调用分类列表的时候需再加个