1. 自定义指令Vue.directions

Vue2:

bind : 指令绑定到元素后调用

inserted: 元素插入父DOM后调用  
 update: 元素更新，但子元素尚未更新，调用

componentUpdated: 一旦组件和子级被更新，就会调用

unbind: 一旦指令被移除，就会调用

Vue.directions(‘Name’, {

Inserted(el, binding, vNode, oldVNode){

const { value } = binding

const roles = store.getters && store.getters.roles

if( value && value instanceof Array && value.length > 0) {

const permissionRoles = value

const hasPermission = roles.some(role => {

return permissionRoles.includes(role)

})

If(!hasPermission) {

el.parentNode && el.parentNode.removeChild(el)

}

}

}

})

Vue3:

created:

beforeMount: 同bind, 在指令绑定到元素时，调用一次

mounted：元素挂载到父元素下调用

beforeUpdate：元素本身更新前调用

updated：组件和子级更新后调用

beforeUnmount：元素卸载前调用

unmounted：指令被移除时调用

Vue.directions(‘Name’, {

mounted(el, binding, vNode, oldVNode) {

}

})

1. Mixins / Vue.mixin全局混入

就是一个js文件, 我们在开发的时候经常会遇到一些相同的代码和逻辑，可以把这些东西给提取出来放在一个mixin.js文件中，然后通过mixins注入他们，注入的mixin的生命周期钩子函数会先于当前组件的钩子函数执行，当前组件中和mixin中相同的data，会对mixin覆盖

Vue2中，mixin的注入通过配置mixins传入一个mixin数组，这样的注入方式是隐示的注入，变量和方法的来源不明确很难知晓，而Vue3中是通过写hooks，useMixin的方式注入，变量和方法的来源都是明确的。

Vue2：(缺点：隐试引入，不知道变量和方法来源，不同mixin的同名方法，同名属性，可能会导致冲突。)

export default {

data() {

return {

aa: 0

}

}

}

Vue3:

export default function useMixin() {

const aa = ref(0)

return {

aa,

}  
}

1. Vue.Extend

Extend 传入的是一个对象，它会去继承Vue上的一些功能，然后返回一个构造器，由这个构造器构造出来的实例对象可以去调用Vue上的一些方法，比如$mount，然后将extend的内容挂载在某个dom节点下。

它的原理就是通过Object.create去继承Vue的prototype，然后将constructor指回子类，并且初始化了子类的props, computed等，返回子类构造器。

function Extend(extendOptions) {

const Super = this

const Sub = function VueComponent(options) {

this.\_init(options)

}

Sub.prototype = Object.create(Super.prototype)

Sub.prototype.constructor = Sub

Sub.options = mergeOptions(Super, extendOptions)

initProps(Sub)

initComputed(Sub)

// 继承静态方法

Sub.extend = Super.extend

Sub.mixin = Super.mixin

Sub.use = Super.use

return Sub

}

1. 防抖和节流 => 闭包 => 垃圾回收机制

防抖是一段时间内，频繁去触发某个事件，事件不会去响应，只有在触发了这个事件，等待n秒后，没有重复触发，才会去响应最后一次触发的事件。

节流是一段时间内，频繁触发某个事件，它不会一直响应，只会每n秒响应一次。

防抖应用场景 比如search搜索联想，window触发调整浏览器窗口大小。

function debounce(fn, wait) {

let timer = null

return function (…args) {

if(timer) {

clearTimeout(timer)

timer = null

}

timer = setTimeout(() => {

fn.apply(this, args)

}, wait)

}

}

节流应用场景 比如鼠标不断点击触发，监听滚动事件，比如滑动到底部自动加载更多。

function throtten(fn, wait){

let curTime = Date.now()

return function(…args) {

if(Date.now() – curTime >= wait) {

fn.apply(this, args)

curTime = Date.now()

}

}

}

function throtten(fn. wait) {

let timer = null

return function(…args) {

if(!timer) {

timer = setTimeout(() => {

fn.applt(this,args)

timer = null

}, wait)

}

}

}

闭包应用场景： 创建私有变量，对外不可见。 防抖，节流，柯里化，模块化，异步操作中的回调函数

异步操作中的回调函数：

function fetchData(url, cb) {

setTimeout(() => {

const data = fetch(url)

cb(data) // 闭包

})

}

纵然很多场景下，我们需要使用到闭包，但是闭包也有很多缺陷，比如可能会导致内存泄漏，可能引发一些性能问题。因为他们可以长时间保持对外部作用域的引用，从而阻止垃圾回收机制释放，这可能会占用更多的内存和处理时间。所以为了避免这种情况，要谨慎的使用闭包，我在编码的过程中一般会打开f12的performance 查看里面的js heap max，当它一直呈现一个上升不下降的情况，可能就导致了内存泄漏，从而去排查自己的代码，并且在开发的时候也会去注意哪些定时器的清除，事件的销毁，以及不需要使用闭包的时候手动断开等。

垃圾回收机制：

见brower篇章

1. Vue3中watch 和watchEffect的使用区别

<https://juejin.cn/post/7109009230132150280?searchId=202312141543422FE4BAF2CD74B246BA99>  
watch是指定去监听一个和多个响应式数据的变化，而watchEffect是隐式的去监听响应式数据，只要里面的响应式数据发生变化，就是重新调用回调函数。Watch在不手动配置immediate的时候初始不会执行，而watchEffect会执行,且watchEffect会返回一个函数，调用返回的函数就能停止监听。而computed是计算属性，依赖于其他的响应式数据，只有依赖的响应式数据变化才会去重新计算，且具有缓存，返回计算值

1. 三栏布局  
   [https://juejin.cn/post/7284160850457821239?searchId=20231215152748E92DDAFD50A94F8CD7A3#heading-4](https://juejin.cn/post/7284160850457821239?searchId=20231215152748E92DDAFD50A94F8CD7A3" \l "heading-4)

三栏布局就是左右两列固定宽度，中间栏自适应

Float : 排列左/右/中

l-float/r-float/( overflow:hidden 可有可无)

Float : 排列左/右/中

l-float/r-float/margin

Float : 排列左/右/中

l-float/r-float/width: calc(100% -左的width)

flex布局: 排列左/中/右

grid布局: 排列左/中/右

display: grid;

grid-template-columns: 200px auto 200px

position: 布局: 排列左/中/右

父: position: relative

子: position:absolute; 左 : left: 0, 右 : right:0, 中 : left: 左宽,right:右宽

1. npm / yarn /pnpm的区别

其实这些包的出现都是为了解决之前工具的一些问题，npm下载node\_module文件，里面的包的关系是树关系，比如node\_module中有A，B两个包，两个包又同时出现了C包，那么C包就会被安装到A和B下，就被装了两次，造成了磁盘的资源浪费。而yarn的出现就解决了这个问题，它会把树文件扁平化，不管树与树之间层级有多深，多个文件依赖一个包的话都会指向一个地方，就是yarn使用的单独的缓存目录，把每个包的所有版本都下载到本地。而pnpm也是使用的单独的缓存目录，但是只会下载每个包的最新版本。  
  
8. 前端⼯程化的理解

10.  websocket心跳机制，断线重连机制。  
文章：   
<https://juejin.cn/post/6945057379834675230?searchId=20231219132633B8510E5B56AF5AE1A6AC>  
代码：  
<https://gitee.com/DieHunter/myCode/tree/master/WebSocket>

心跳机制就是 前端在WS连接成功的情况下，开始执行心跳函数，会向服务器发送心跳信息比如’Ping’信息，服务器若正常疏导信息则会返回心跳信息比如’Pong’信息。一定时间内，如果按时接受到，则连接正常，则重置心跳(开始第二轮触发心跳).若前端在一定时间内没有收到心跳函数，则表明连接失败，就关闭ws，进行重新连接。

11 登录无感认证（token无感刷新）要怎么实现，在什么地方部署。

Token设置短期过期时间例如几个小时，同时用户认证后后台返回refresh\_token有效期一两个月，如果token过期那么就通过refresh\_token认证，成功则返回新的token保存在本地。可以在beforeach全局前置守卫和单独组件路由beforenter中设置

1. 项目难点呀

我近期做的一个项目的难点就是针对PBC绩效管理系统进行了性能优化，我在这过程中有查很多资料，然后总结了一下其实可以从我们输入url查询到页面呈现出来的整个过程中来进行优化，这个过程主要涉及到缓存判断，网络请求，以及页面渲染。但是主要涉及到的就是对于资源进行合并和压缩，减少请求数量，加快每次请求的速度。然后我把项目进行了打包，分析了打包结果可视化图，可以明显看到包资源的大小，所以对一些第三方插件比如elementPlus, echart等进行了按需引入，并且针对路由和组件都进行了按需加载，vite中配置gzip压缩，把一些第三方资源通过CDN引入等。然后通过lighthouse和performance都进行了性能测试，把第三方引入的资源放入了pulic中，自己的资源放在static中，并且对小图片进行了base64处理，对图片进行压缩和图片格式转换，还针对css进行了优化，对Js长任务进行了优化，还针对首批处理骨架屏，图片懒加载等。

1. excel 导出

主要采用的就是分片的方式，可以获取到chunk列表，然后把他们根表头组装在一起通过Blob将他们转换成流对象，放入FileSaver.saveAs来进行下载导出。

1. 国际化注意事项？

国际化处理就是去配置不同的语言包json文件，然后将当前系统语言/你切换的语言存储在localStorage或cookie中(我们的项目是localStorage)，然后写了一个translate函数，由于我们json文件语言包是用中文作为key值做映射的，所以使用的使用调用translate函数传入中文key即可，这个translate函数会获取到当前language的语言包，然后映射出该语言包的文字。但是由于不同国家语言的文字长度，方向，字体，时间等不一样，还会相应的进行处理。

1. RBAC模型权限管理模块

我们这个项目因为涉及到公司不同职位，不同等级，所以针对不同人员的角色进行分配设置了不同的菜单和接口权限。就是通过给角色分配菜单和接口权限，给人分配角色的模式来实现的，以及还有一些按钮权限是通过指令的方式来实现的。菜单权限就是后台返回的该用户具有的所有菜单然后动态追加在路由当中去。