# resume\_project

## 自我介绍

，

## 主题色切换功能

一般主题色不同所涉及到的多是背景，字体，激活状态上颜色的不同，我们项目里面设置了三个主题色，每个主题色放一个scss文件，利用@mixin和@include实现某些样式的复用，并且把样式包裹在body的该主题色类下，来实现了一个样式隔离。然后有一个主题色切换的组件，并将主题色变量存放在store中，使全局都可以直接访问到，并且通过document.getElementsByTagName('body')和这个主题色变量来设置body类名，进而作用上该主题色。 然后还有一个黑夜模式的切换，黑夜模式下通过设置html的类名为dark，然后加载相应的样式文件实现，除此之外我们项目还结合了用户自己通过picker拾取选择主题颜色，但这个覆盖的范围只涉及到一些具有背景色的地方，这个主要是利用css变量结合setProperty的方式实现的

方案对比：

1. CSS变量 + 类名切换 (Vue3官网采用方式) :root下定义css变量， 在不同类下去修改变量值，需要用到的时候直接通过var(变量)则可

2. 提前引入所有变量，做类名切换 - css，直接给引入所有主题样式，做类名切换 - scss， 通过混入@mixin 和@include 实现，在需要进行主题色切换的地方，用@mixin 混入不同主题的样式, 使用@include引入即可(切换时不需要重新加载样式文件，也不会有卡顿，优先级问题 但是在首屏加载的时候需要耗费一定时间去加载样式文件)

3. Css变量+setProperty方式切换主题色(Vue-element-plus-admin)，这种主要用于需要用户通过picker选择颜色来拾取主题色，不是固定主题色的情况。  
3. 语言国际化功能

因为项目是基于vue3的，组件库采用了element-plus，所以国际化配置直接利用了element-plus中的语言包+自己配置语言包json文件，每种语言配置一个json文件。以及结合了vue-i18n插件. 通过pinia + localStorage来存储当前语言变量，我刚开始采用的是为每个需要翻译的词都定义了一个英文的key，但这样写当要翻译的东西变得庞大之后，逐渐发现内容很复杂，并且取名也很麻烦，所以我在网上查询了一下，发现了一个库Voerkal18n， 借鉴里面的思路，写了一个translate函数，直接传入的中文，然后自动将其翻译成想要的语言。 怎么翻译的：因为机翻会有很多不准确的地方，所以采用的人工翻译模式，对于那些静态的文字，封装translate函数的时候，可以获取到当前系统语言，以及所有的语言配置文件，然后通过messages[language]去获取到当前语言包，再通过键和值对应取传入的中文的值，这个值就是对应的被翻译后的语言。对于一些动态表头的翻译，就是通过后端返回的，然后我这边去根据当前语言取值

国际化落实问题： 1. 不同语言的文字大小 尽量考虑竖向排列而不是横向排列 尽量不使用固定宽度，如果固定宽度，考虑超出部分... 表示，或者tooltip展示所有 比如按钮，不设置宽度，通过设置padding 2. 不同标准的语言标识符之间的转换  
4. 关于Echart中问题

1. 柱状图 / 折线图：x Axis 的 x 轴名称显示不全问题:

xAxis.axisLabel.interval 默认采用的是标签不重叠的策略间隔显示标签，将它设置成0的时候可以强制显示

1. 饼状图：图例文本和icon不在同一水平线上问题  
    修改fontFamily legend: { icon: 'circle', textStyle: { fontFamilt: 'serif' } }
2. 自定义tooltips：配置Echart中的formatter回调，因为tooltips比较复杂， 所以新创建了一个Vue文件

由于Echars的formatter回调十分频繁，而在formatter中引入组件是通过createApp 结合mount的方式引入的，所以每次都要重新创建APP实例，导致性能问题，所以为了避免每次都创建app实例，我把createApp(toolTip)放在回调外只调用一次，每次更新实例中的响应式变量即可。但这就涉及到了外部代码和App实例通信的问题，这里借鉴了ElementUI里面编程式调用一个弹窗的思路， https://juejin.cn/post/7262312427763990588?searchId=20231211150721FE3AA40A32C5DE67ABBF <https://juejin.cn/post/7271651372771065912#heading-8>

5. 虚拟列表技术

虚拟列表是一种技术，他只渲染可视区域内的列表项，而不是渲染整个列表，当用户滚动容器时，虚拟列表会根据滚动位置和可视区域的大小去计算出当前应该显示的列表项。

从后端请求获取到10000条数据，将它存在data中，而初始渲染list为data的前containerHeight / itemSIze条，通过ref去获取元素，并且获取到元素的scrollTop，假设每一条数据的高度已知，且容器的高度已知，通过scrollTop / itemSize 和 (scrollTop + containerHeight ) / itemSize 可以计算出可视区域内要渲染的元素索引。

对于高度不固定的情况下，可以设置一个默认的虚拟itemSIze，通过this.$refs.container.clientHeight可以获取容器高度，然后计算出一个虚拟的positions数组，这个数组包含了从data数据的每一项的height，index，top，bottom，然后在update钩子函数中去更新positions数组中当前可视区域内item的真实高度和位置，这个updateItem可以通过this.$refs.items来获取所有的nodes，并遍历node节点通过getBoundingClienrRect获取它的height，oldHeight是我们默认设置的高度, 当oldHeight和height不等时，去重置position里的height为height， bottom 为oldHeight - 差值。然后便宜量等于start位置的bottom值，更新之后再重置偏移量。

<https://juejin.cn/post/6844903982742110216#heading-4>

在从position 中查找start值 和 end 值的时候，用到了二分查找算法。

用到了算法：二分查找算法

Positions.some(item => item && item.bottom > this.scrollTop)

由于positions中的数据改为二分查询算法(O(logn), 最差为O(n)), 而Array.finds的时间复杂度为O(n)

找第一个大于value的值:

function(position, value) {

let start = 0

let end = position.length;

while(start <= end) {

const centerIndex = Math.floor(start + (end - start) /2)

if(position[centerIndex].bottom == value) {

return centerIndex + 1

}

if (position[centerIndex].bottom < value) {

start = centerIndex + 1

} else {

end = centerIndex

}

}

return start

}

找到第一个小于value的值:

function(position, value) {

let start = 0

let end = position.length

while(start <= end) {

const centerIndex = Math.floor(start + (end - start) /2)

if(position[centerIndex].bottom == value) {

return centerIndex + 1

}

if (position[centerIndex].bottom < value) {

start = centerIndex

} else {

end = centerIndex - 1

}

return end

}

}

追问：一个列表项元素很大时，出现跳跃式滚动怎么优化，白屏怎么优化.

白屏的情况可能是由于数据请求延迟导致的，这是时候我们可以设置一个虚拟数据放置在页面上，等请求完成再去移除虚拟数据，更新为真实数据。还可以采用增加加载提示。除此之外，还可以应用预加载技术，提前加载当前页前一页和后一页的数据。  
**5. 首屏优化(包体积，首页，白屏)  
  
FCP：**首次内容绘制(first content paint)

**LCP：**最大内容绘制(largest content paint) **TBT：**总阻塞时间(total block time) **SI：**速度指数(speed index) **CLS：**累积布局偏移(cumulative layout shift)

<https://juejin.cn/post/7188894691356573754?searchId=2023121210475077B0E9C9480B03FA32F9>

我做这个性能优化的时候，是因为发现每次第二天来公司上班启动项目发现项目在本地加载时间很久，然后我去问了我的导师是否需要做性能优化，他给我说项目上线nginx开启gzip压缩首页加载就会变得很快，但因为正好那几天比较闲，所以我还是自己尝试了进行性能优化，由于我们是vite项目，所以我先是用了rollup-plugin-visualizer插件对包进行了分析，发现主要是element-plus和Echarts占比比较大，以及项目中存在一些没有使用的代码，然后去lighthouse上去分析了页面加载情况，针对诊断提示，于是我首先进行了按需引入和删除没用的包，并且对资源文件进行了整理，将第三方的静态资源放在pulic下，自己的资源放在static下，并且把一些静态引入的图片进行了格式转换png转webp，以及对图片进行了压缩，然后就是关乎减少javascript的执行时间，由于Vue3自带了treeShaking就没有进行配置了，然后对路由和组件进行了按需懒加载，以及对于某些长任务手动进行延时处理，这些长任务是通过performance测试出来的，然后针对首屏还使用了骨架屏插件，来优化可能因加载速度过慢产生白屏的情况，以及在路由跳转都增加了loading加载提示。然后针对网络请求上面，把一些第三方资源通过CDN的方式进行导入(vite-plugin-cdn-import插件)，以及使用rollup自带的external, 设置别名结合output.globals解决重新打包报错的问题。我大概就做了这些事，其实还有些可以对首屏进行虚拟列表渲染，预加载，图片懒加载，自定义分包策略，以及对缓存策略进行一定的操作，http协议升级等来优化，但是我能力有限，且项目暂时不需要这么多东西，所以就没有继续做了。但是正式由于做这次优化我去把H5端的项目的下拉分页加载下一页数据改成了虚拟列表的形式进行渲染，也优化了一定的性能。

路由按需引入：

Vue3 + Vite中: 在拿到所有的路由文件后，通过import.meta.glob(`../\*\*/\*.vue`)

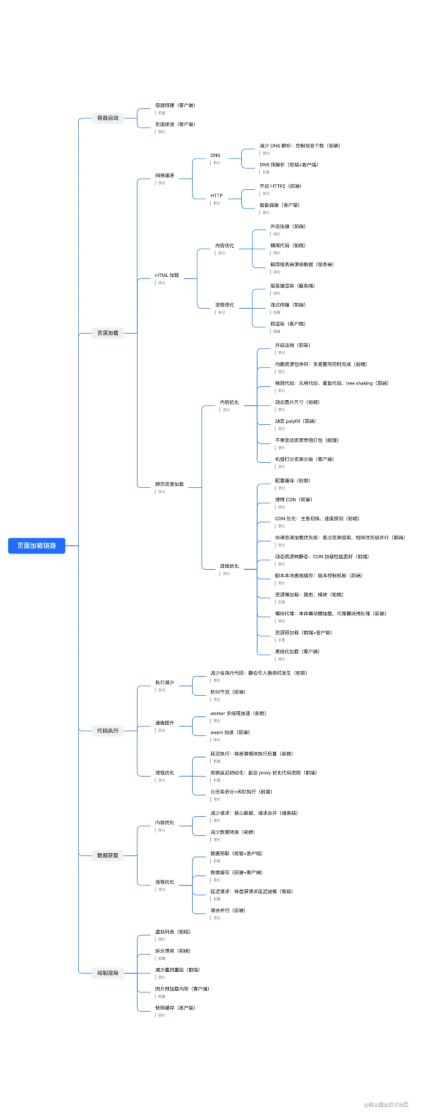
可以拿到所有的.vue文件，然后去匹配我们的路由文件，就对所有的路由进行了按需引入。

Webpack中： () => import(/\*webpackChunkName: “Name”\*/, ‘url’);

原理是：把路由分成了不同的chunk包，只有在需要用到的时候才会去加载这个chunk

进阶：如何实现秒开？

1. 超大图片(100M)图片秒开
2. H5秒开
3. 所以：() => import的原理是怎样的，它是如何实现分包的？ 太难了，先放着oAo
4. 十万条数据，如何处理：<https://juejin.cn/post/7205101745936416829?searchId=202312141425093FEAD2FFA645BA3FCF84>



如何优化长任务？

https://juejin.cn/post/7155151495694909447#heading-3

1. 长任务拆分，利用setTimeout将长任务拆分成多个宏任务来执行
   1. 原理：长任务造成页面卡顿是因为js线程占用了控制权，UI线程无法进行渲染，而利用setTimeout是宏任务，每一次执行完之后会把控制权交给UI线程，等UI线程渲染任务结束后才会进行下一次宏任务。这解决了页面卡顿问题，但这会导致总耗时增加
2. 使用requestIdleCallBack 函数，这个函数是由浏览器决定的，在空闲的时候执行队列任务的能力，从而不会影响UI线程的渲染，然后它的第二个参数还可以设置就是超出多长时间后直接放在执行队列里执行
3. webWorker：

new 一个webWorker, 开启一个运行在后台的javascript，把长任务放在webWorker里， 它是独立于脚本，不会影响主线程的运行，当执行完成之后通过postMessage通知主线程执行完成。

**6. 复杂表格联动，多级选择框数据联动，动态配置数据项，表格内数据操作。**

[https://blog.csdn.net/Sabrina\_cc/article/details/123230346?ops\_request\_misc=&request\_id=&biz\_id=102&utm\_term=table%E4%B8%89%E8%A1%A8%E6%95%B0%E6%8D%AE%E8%81%94%E5%8A%A8&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduweb~default-0-123230346.142^v96^pc\_search\_result\_base5&spm=1018.2226.3001.4187](https://blog.csdn.net/Sabrina_cc/article/details/123230346?ops_request_misc=&request_id=&biz_id=102&utm_term=table%E4%B8%89%E8%A1%A8%E6%95%B0%E6%8D%AE%E8%81%94%E5%8A%A8&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-123230346.142%5ev96%5epc_search_result_base5&spm=1018.2226.3001.4187)

处理数据用到的算法

1. 树型结构转换

组件封装：考虑配置，考虑权限，考虑安全，考虑复用。

1. <https://juejin.cn/post/7300872843587469327?searchId=2023121214192189C1B7804417AF0A5345#heading-7>

考虑配置项，哪些属性，方法，事件，方法

为何要进行组件封装：

1. 二次封装是为了样式统一
2. 方便复用，降低耦合性，使单个组件和功能的变更不会影响其他部分
3. 增强安全性，可以隐藏数据和实现的细节，防止外部的非法访问和修改
4. 方便维护，将修改局限在组件内部。

组件拆分和组件封装的区别：

a: https://segmentfault.com/a/1190000039846269#item-5

1. 组件拆分更多关注于如何将大的功能模块划分为小的，可管理的单元，以便管理和复用。
2. 组件封装更多则关注于如何将组件的内部细节隐藏起来，提供清晰的接口，以保证组件的独立性和安全性。

封装流程组件：

我们的业务是需要在11个流程页面里面都展示这个流程组件，并且每个流程页面都可以推动流程的更新且数据的展示来源于同一个接口，内容是根据流程阶段作区分的，所以就封装了一个流程组件，在里面展示流程执行阶段图，并且根据流程阶段调接口更新该流程数据传递给父组件，父组件可以直接渲染数据，除此之外在里面监听了流程阶段更新事件，当用户操作流程进入到下一步，触发流程更新，流程组件则会自动执行重新获取新流程信息，以导致路由更新。

拆分公共的人员选择弹窗组件：

滑动选择组件：

动态表，表头的封装(涉及到动态表头的中英文翻译)



对于动态表的封装，只需要把某个指标项对应的数据结构传入到动态表以及权限控制是否可修改。每个动态表头对应下的可编辑数据控件也是动态获取，通过在任务项的规则id项和指标项规则id进行比对，相同则渲染，来避免element-plus表格组件重复渲染的情况，且表格里的数据编辑其实是不调用接口的，只有在确认编辑以及编辑每一项任务评分标准确认后才会调用接口更新，到数据更新，通过emit更新事件，触发父组件重新调用getInfo接口更新表格数据。

针对动态表格进行配置主要是涉及到业务中需要，表格头根据不同的目标项进行配置，然后涉及到动态表头的渲染以及动态表头下的表格项动态框的操作，还有个性化的操作项。我只需v-model:taskInfo传入当前这条目标项任务数据，当保存的时候触发update:taskInfo更新。

**7. 处理权限管理数据的时候**

递归算法

1. 权限管理主要是分为几大部分，菜单权限，路由权限，按钮权限以及数据权限，接口权限。

采用给用户分配角色， 给角色分配菜单和按钮权限等的模式。菜单权限是通过跟后端商量好，通过token 请求接口，由后端根据用户token返回对应的权限菜单，然后动态向页面中去加载动态路由，路由权限同理。而按钮权限是通过指令来控制的，比如我写了一个v-permission指令，在mounted里面去通过binding获取到传入的value值，来匹配该用户是否具有该value权限，如果不具有，那么通过el.parentNode.removeChild(el)的方式移除该权限按钮的展示，数据权限也就是我们根据用户的角色去判断，比如表格某一列的显示或者隐藏，然后接口权限就是某些接口是否需要身份验证等才能请求。

动态加载路由，就是在路由导航守卫的时候，去设置由后端请求获取到的路由权限对象(vuex中去调接口请求并拼接好动态路由和静态路由，然后重置路由，resetRoute在route/index.ts文件中先清空路由，再通过router.addRoute添加获取到的路由)。一般前置路由守卫就是处理无token跳转登录，登录状态下登录页自动重定向到首页，以及动态路由的加载，然后在视觉上处理路由跳转加载。后置导航守卫中清除掉跳转加载。

<https://juejin.cn/post/7215043206035750968?searchId=20231213162035CDAA6F23C33139AEB661#heading-0>

**7.5 自定义 Echart 表格 tooltip**

**8. 项目搭建中的问题**

<https://juejin.cn/post/7156214249369370655?searchId=202312131725530B7E4CFBAB7AEFB8739C>  
**## 安装uniapp + vue3 + ts 项目**

- npx degit dcloudio/uni-preset-vue#vite-ts vue3-vite-uniapp

- yarn i

- yarn dev:h5

**## 卸载不需要的包**

**## 安装sass预处理器**

yarn add -D sass

**## 配置eslint + prettier 自动格式化代码**

yarn add eslint --dev

yarn add prettier --dev

yarn add eslint-plugin-prettier --dev

yarn add eslint-plugin-vue --dev

yarn add @typescript-eslint/eslint-plugin --dev

yarn add @typescript-eslint/parser --dev

yarn add @vue/eslint-config-prettier --dev

yarn add @vue/eslint-config-typescript --dev

**## 配置.eslintrc.js文件 和 .prettierrc.js文件 和 .vscode/settings.json文件**

https://eslint.vuejs.org/rules/max-attributes-per-line

- eslinttrc.js 记两个规则

-- 不强制使用全等 eqeqe: 0

-- 行尾不使用分号 semi: ['error', 'never']

-- 处理组件名不是 mutil-word问题

-- 处理json文件，报错Unexpected token:问题

- .prettierrc.js 记两个

-- tabWidth：2  一个tab几个空格

-- printWidth: 80 一行字符数，超出自动换行

-- singleQuote；true  字符串是否用单引号

-- semi: false 行末是否使用分号

-- trailingComma: "none" 行末是否使用逗号

-- prettierr，eslintrc的配置不生效问题，要先在vscode中安装插件

**## 配置package包检测器**

- "lint": "eslint --ext .ts,tsx,vue src/\*\* --no-error-on-unmatched-pattern --quiet",

- "lint:fix": "eslint --ext .ts,tsx,vue src/\*\* --no-error-on-unmatched-pattern --fix"

**## 配置husky 和 commitlint 进行git代码提交的检查**

- 安装husky 以及commitlint相关的插件，然后添加prepare脚本，在yarn install 的时候会自动去执行它然后生成.husky文件，并且通过husky add 去添加pre-commit 和 commit-msg, 在pre-commit里面去执行script里配置的lint-fix就会针对我们的eslint去对代码格式进行检查，commit-msg里配置commitlint就会根据我们自定义的.commitlint.config去进行commit检查。

**## 后序**

基本项目搭建完成就去引入相应的组件库, 引入状态管理工具pinia等，封装uni.request请求。

1. **水印组件封装，Hooks**

组件：  
<https://codesandbox.io/p/sandbox/virtuallist-1-forked-kfzhc5?file=%2Fsrc%2Fcomponents%2FVirtualList.vue%3A9%2C42>

组件封装主要考虑需求有哪些配置，水印主要的配置就是动态传入水印文字，间距，字体，颜色，旋转量，宽，高等，然后通过这些动态值去绘制canvas, 通过$ref获取canvas,canvas.getContext(‘2d’)绘制一张图层，根据传入的动态值设置图层的宽高，字体，字体颜色，旋转量，填充文本等然后通过canvas.toDataURL(“image/png”)将它转成base64格式，再动态给div添加background-image以及background-size。

水印防篡改：

组件中：在onMounted生命周期中通过MutationObserver去监听节点的更改，在onUnmounted里面去disconnect掉我们的监听事件。当判断更改的节点有我们的水印节点的时候，flag++，这个flag是拿来收集依赖的，因为我们把生成水印base64图片放在watchEffect当中，并且在里面使用一次flag,当flag更改的时候，就会去触发依赖重新调用watchEffect重新生成水印，就实现了防篡改。

Hooks中，直接设置dom.style.pointerEvents = ‘none’避免对用户的交互产生影响。

1. **uni.request的封装**
2. **H5端在App上做适配**

iphone4/5: 320px

iphone6/7/8: 375px

iphone6/7/8+: 414px

iphone12 pro：390px

iphone14 pro max： 430px  
像素大全：https://uiiiuiii.com/screen/  
调试方式：https://blog.csdn.net/u010412719/article/details/77460362

1. 简单粗暴：

<meta

name="viewport"

content="

width=320,

maximum-scale=1.3,

user-scalable=no

"

>

2.只适配iphone6+可以使用

根html的fontSize + media查询不同设备设置fontSize

3.所有width设备都适配，可以根据初始屏幕宽度，动态设置根html的fontsize + rem

(function(doc, win){

const dolEl = doc.documentElement;

resizeEvt = 'orientationchange' **in** window ? 'orientationchange' : 'resize',

// 重新计算fontSize.

const reCalc = () => {

const clientWidth = doc.clientWidth

if(!clientWidth) return

dolEl .style.fontSize = 100 \* (clientWidth / 320) + ‘px’

}

if(!doc.addEventListenr) return

win.addEventListener(resizeEvt, reCalc, false)

// DOMContentLoaded事件，所有的html解析玩，以及延迟script脚本下载，执行完，触发reCalc

doc.addEventListener(‘DOMContentLoaded’, reCalc, false)

})()

4.高清图片：

1. 根据media query查询不同设备像素比下，设置不同倍数图片

2. 如果支持webkit私有属性image-set可以直接配置多倍图及其位置，它由浏览器自动选择加载n倍图，这样在网络差的时候，可以降分辨率加载图片

5.兼容问题：  
<https://juejin.cn/post/6901940698518732808?searchId=2023121500513447D21422774215435EF0>  
<https://juejin.cn/post/7103835385280593957?searchId=2023121500513447D21422774215435EF0#heading-3>

1. 日期处理

*//IOS不能处理日期中的-，要把-变成/*

**let** date = **new** **Date**('2019-02-28 18:33:24'); *// null*

2. android部分版本input的placeholder偏上，

设置Input { line-height: normal }

1. ios手机设置overflow: hidden 和border-radius，子元素超出部分不隐藏

父元素添加transform: rotate(0deg)

4. transform导致z-index失效的问题

1. 父元素，任意父元素，非body，添加overflow: hidden

2. 通过设置translateZ()一个足够大的值，比如translateZ(120px)，近大远小的原则，可以让z-index元素覆盖在transform元素上。

5. ios手机input输入框设置opacity会导致聚焦拉起键盘时，不会把输入框挤进可视区域

opacity小于等于0.01时，就会出现这个情况，直接设置opacity: 0.011  
  
6. 移动端300ms延迟问题

历史遗留问题，早期的网站主要是为大屏幕电脑设置的，而在手机上预览就会导致内容缩小，所以就约定了双击等比例缩小，然乎就是利用首次点击后300ms判断是否又一次点击了，来判断双击。

解决方法是在HTML文档头部添加meta标签，添加user-scalable=no禁止缩放，或者给html设置touch-action: none. 或者直接引入fastClick库，在load完之后，触发fastClick.attach(document.dody)

7. ios手机上将图片转成base64的失败

给image对象设置允许跨域，再给image对象的src赋值

imageObj.setAttribute(“crossorigin”,”anonymous”)

imageObj.src = this.url

8. vite3创建的项目在ios12.1版本及以下的手机上白屏，这是因为globalThis为undefine, globalThis不支持ios 12.2以下版本，解决方法是在 globalThis = undefined的时候，设置globalThis = window

9. android系统下，border-raduis: 50%不圆问题

因为使用了rem布局，部分机型出现问题，设置具体的px数值就可以。

10. android系统下，line-height = height的时候不居中。

原因是android在排版计算的时候参考了primyfont字体相关属性，而primyfont查找是看font-family哪个字体在fonts.xml里第一个匹配上，而原生android下中文字体没有family name,无法匹配上中文，所以需要去声明显示中文

其他解决：把字号内外边距等设置为需求的2倍，使用zoom进行缩放。

其他解决：使用flex布局

1. 弹窗的内容长过屏幕，需要弹窗内容滚动，主页不滚动

解决方法：

弹出弹窗时，使背景主页内容作为固定定位，这样页面的滑动对其就没有影响，关闭弹窗的时候，移除这个固定定位的类名

1. **ChatGPT接入问题**1. 返回数据实现逐字呈现效果

从后端请求获取到数据之后，去遍历data文本的长度，通过createElement()去给每一个字动态创建span标签，然后append到Dom上去，并且给它的opacity属性添加上过度动画，并且给每个字都添加上动画延迟属性，比前一个字多0.2s.

2．chatgpt stream 流式处理网络请求

1. 用socket，建立即时消息通信

如何处理socket的数据呢？

2．事件流Server-sent  
<https://juejin.cn/post/7238493216207192122?searchId=202312141410460436E4F1C5C3F5392508#heading-10>

只需要new EventSource()创建一个连接，通过onmessage来接受后端推送的数据。

1. **单点登陆**

单点登陆就是多个系统去共享一个登陆状态，我们系统的单点登陆是通过在一个页面登陆之后将token保存在cookie下，由于公司的项目都是同一个父域下，所以cookie可以实现共享，然后其他的几个后台就可以拿到这个token 直接登陆来。但是很多时候其实单点登录会涉及到不同域下，那么现在的问题就变成来既要考虑如何去保存状态又要考虑如何跨域，那么就可以通过iframe的形式来实现，比如说我们现在有A，B两个系统，我在B系统里面去通过iframe嵌入A系统的登陆界面，登陆之后去设置cookie在A的domain下，再通过postMessage和onMessage来实现父子组件通讯，这样我们就能去调用A项目认证下的接口，从而带上A域名下的cookie了，来实现了登陆状态共享。

**大文件分片上传，断点续传，错误重传，实现过程，其中有哪些优化**

一般来说，当文件大小超过5M,我们就可以做大文件上传，我做这个需求的时候，采用的是分片上传的形式，这是因为File是Blob的子集，所以可以直接使用slice方法进行分片，再对切片后的切片数组遍历进行上传，但是这就有一个问题是如何告诉后端，我们上传的多个文件切片是否是来自于同一个文件，这个时候就想到了用md5对文件加密获取唯一hash值的方式，去调用spark-md5库去计算hash值，当切片都上传完了，手动发送一个合并切片请求通知服务器帮助我们合并切片。 断点续传是服务器需要记住已上传的切片，这样在下次上传之前，就可以跳过已经上传的部分。我采用的是每次上传前，向服务器获取已上传的切片。 错误重传可以利用一个数组来记录切片重传的次数，并设置一个最大重传次数，当超出这个重传次数，就reject掉，代表上传失败。但是针对很大的文件，比如100GB单位的文件，其实计算hash值这一过程就会浪费很多时间，所以我把计算hash值得过程给放在了webWorker中，不影响主线程得进行，当计算完成，通过postMessage和onmessage进行通信将结果值传递给主线程，然后针对计算过程，也进行了优化，采用了抽样取chunk的方式 ，采用第一个chunk和最后一个chunk，中间的元素取前中后2字节，然后将他们组合起来，计算hash值，这样大大减少了hash计算的时长。

**self.importScripts("spark-md5.min.js");**

**self.onmessage = e => {**

**const { chunks } = e.data;**

**const spark = new self.SparkMD5.ArrayBuffer();**

**let count = 0;**

**const loadNext = index => {**

**const reader = new FileReader();**

**reader.readAsArrayBuffer(chunks[index].fileChunk);**

**reader.onload = e => {**

**count++;**

**spark.append(e.target.result);**

**if (count === chunks.length) {**

**self.postMessage({**

**hash: spark.end()**

**});**

**} else {**

**loadNext(count);**

**}**

**};**

**};**

**loadNext(0);**

**};**

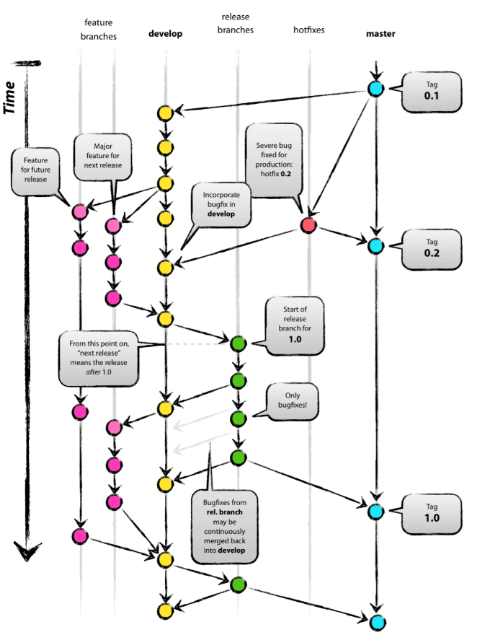
**100张图片，如何实现并发上传**

<https://juejin.cn/post/7307587537295343631>

浏览器的最大并发数量是6，实现并发上传可以直接利用Promise.allSettle的方式传入一个promise 数组集，拿到切片数组后，每次都通过splice取0~6个数据，放入allSettle中，这个时候有可能会出现其中一张图片/切片上传失败的问题，可以定义一个上传失败的切片数组，取出allSettle返回值里面失败的切片重新放入到失败数组中，让切片上传结束之后，再去上传失败切片数组中的切片，直到切片和失败切片数组长度均为0时，就上传结束了。除此之外，还可以去定义一个记录切片失败次数的数组，当切片index上传失败一次给它的计数+1，当计数>一定值，直接抛出错误，跳出while循环。

**Git工作流，GitHub工作流，GitLab工作流**Git常用命令：  
<https://juejin.cn/post/7304898485156806675?searchId=20231215150502B6705413C6C7938DEBB4#heading-55>

Git工作流：



master, hotfix, release, develop, feature

Git工作流主要维护两种分支， master和develop主要分支, hotfix,release,feature辅助分支。代码库中只有一个主要分支master,它是最稳定的，可以随时发布到生产环境下，develop分支是由master分支签出来(git checkout develop)，用于日常开发，保持最新代码的分支，然后feature分支是我们开发特定的功能时从develop分支签出来的，(git checkout -b feature-\* develop)当开发完成，合并到develop分支下，当项目发布前，还有一个预发布分支release(git checkout -b release-1.x develop)，用于版本测试，并且可以在上面做一些小的bug修复，预发布结束后将其合并到develop分支和master分支，并且对合并生成的新节点做一个标签(git tag -a 1.1),而hotfix分支是版本上线后发现了一些线上的bug，从master分支签出来的(git checkout -b hotfix-0.x)，进行修复结束后合并到develop分支和master分支(git merge –no-ff hotfix-0.x)并且对合并生成的新节点做一个标签(git tag -a 1.1)

GitHub工作流：

只有一个master分支，其他的所有分支都是从master签出来，然后代码写完了，会向master发起一个pull request请求(PR),请求通过了就会合并代码并且删除掉签出来的这个分支。

GitLab工作流：

结合了Git工作流和GitHub工作流，有一个原则是上游优先原则，只存在一个主分支master,它是所有其他分支的上游分支，只有上游分支采纳的代码变化，才能应用到其他分支。

**18. 直播相关东西  
重要的文档：**[**https://github.com/tencentyun/TUILiveRoom/tree/main/Web**](https://github.com/tencentyun/TUILiveRoom/tree/main/Web)

[**https://cloud.tencent.com/document/product/647/63830**](https://cloud.tencent.com/document/product/647/63830) **1. IM的初始化：**

通过TencentCloudChat.create()传入SDKAPPID, websocket服务器地址创建IM，如果需要支持图片，视频，文件等，还需要加入文件上传代理地址和文件下载代理地址。然后创建出了im实例chat，Im的其他接口方法都是通过这个im实例来进行调用的，根据userId， 和userSig实现登陆。通过chat.on去监听事件，以及调用实例方法去处理操作来实现的IM交互，

**2. IM里面做了哪些东西**

1. 由于项目是集成的腾讯 基于即时通信 IM 和 基于实时音视频, 所以基本的主播端和用户端已经有了，我在这里面主要做的是对界面布局进行改版，以及增加了点击常用语发送，支持文件。聊天信息的滑动组件，主播端开播控制，用户端等待直播倒计时控制，以及弹幕的设置和优化。

**3. 拉流，推流做了哪些**

增加直播推流到腾讯CDN，在中台创建房间的时候，增加了一个推流到腾讯CDN的推流模式，主播根据创建的直播链接（包含code）及邀请码进行身份验证进入直播间，检测页面( 包括摄像头、音频、网络）等，进入之后开始初始化直播，并且通过eventBus注册系统事件监听及全体事件监听(开关音频)，根据配置的推流到腾讯CDN方式开始推流，根据直播流ID去调用推流到CDN的Api即可。

然后用户端，根据分享直播间链接或者点击直播进入，都可以拿到这场直播的code，获取该场直播的详细信息，同时拿到快直播地址，HLS直播地址，通过这些source去初始化播放器，再调用播放器的src即可进行播放。

**4. 弹幕配置，弹幕优化？  
直播弹幕问题：** 1. 实时性高，你发我收，毫秒之差  
 2. 并发量大，一人发，万人观看

3. 数据一致性的要求不高高并发优化  
1. 针对用户本身，控制用户发送弹幕间隔时间，每间隔5s才能继续发送下一跳弹幕，减少可能因为不断重复快捷发送而导致的高并发请求。  
2. 针对数据量大，做缓冲池

**4.4. 怎么实现的弹幕功能**<https://juejin.cn/post/6888488657372594183?searchId=202312171426570F91F995A05AADBAADD4#heading-8> **1. css 实现**

针对弹幕功能，我是采用的css动画来实现的，用了一个缓冲池来缓存服务器端推送过来的弹幕，以及一个数组来放置显示上屏的弹幕，在mounted挂载的时候，设置定时器轮询取缓冲池弹幕上屏，且对上屏弹幕做了一个溢出处理，比如设置了一个最大数量值30，当上屏弹幕超出阈值，则清除掉头部的弹幕，用户在刚开始进来的时候，只能获取到最新的30条弹幕，针对自己发送的弹幕，则直接进行上屏，不需要放入缓冲池处理。

**2. canvas实现**

**5. 奖励机制相关？**这个直播奖励机制是根据主播配置的金币奖励方案，除了登陆奖励，回放观看奖励，在线时间奖励外，还有观看直播奖励，当用户进入直播间开始计时，根据主播配置的时长和金币以及奖励上限进行金币发放。到达对应时间会自动弹出领取奖励的浮动组件。这里面主要涉及到的记时判断(开始记时，暂停，结束，重置)和用户观看状态切换(签到打卡，观看奖励，提问奖励，领完)。

**6. 倒计时怎么做的？**

实现倒计时，其实只要用setInterval或者setTimeout, 每隔1s减少1剩余时间就行，但是这样做的倒计时是不准确的，所以我使用了Date来纠正偏差，通过在调用当前的countdown减去开始定时器的时间为偏差值，在剩余时间里面，减去这个偏差值。

1. **Vedio播放器相关**

[https://blog.csdn.net/qq\_36380426/article/details/107527378?ops\_request\_misc=&request\_id=&biz\_id=102&utm\_term=%E5%89%8D%E7%AB%AFSourceBuffer&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduweb~default-1-107527378.142^v96^pc\_search\_result\_base5&spm=1018.2226.3001.4187](https://blog.csdn.net/qq_36380426/article/details/107527378?ops_request_misc=&request_id=&biz_id=102&utm_term=%E5%89%8D%E7%AB%AFSourceBuffer&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-107527378.142%5ev96%5epc_search_result_base5&spm=1018.2226.3001.4187)

1. **Excel会用到的相关函数**

Sheet\_to\_json workbook转json数组对象

Sheet\_to\_html workbook转html  
decode\_range (sheet[‘!ref’]) 获取workbook范围

Const cell = encode\_cell({c: column, r: row}) 生成单元格信息

format\_cell(cell) 生成单元格文本值

**设计模式？**

熟知的设计模式有单例模式，工厂模式，迭代器模式，策略模式，观察者模式，发布订阅者模式，装饰器模式等。

单例模式：在Vue中比如状态管理器vuex就是一个全局的单例模式，只有一个vuex store实例被创建。

工厂模式，经常用到的是用它去创建一个对象，在里面去封装创建这个对象的具体细节。

迭代器模式，就是给一个容器增加迭代属性，可以进行遍历，比如一个可迭代对象，类数组，对于非迭代的可以通过Symbol.iterator给他添加上迭代器。

策略模式，就是根据策略去选择不同的算法，比如我们经常写的switch, 根据不同的case来选择不同的行为，其实可以把不同的case给封装到一个Object对象中，当有新的选择方案的时候直接通过object对象添加，而不需要重新去修改switch内部，还有就是我们的sort排序算法也是策略模式，当I < 10的时候它是采用的插入排序算法，大于10采用三路快速排序算法。

观察者模式：就是有一个观察者和被观察者，当被观察者发生变化时，观察者去做出某些操作，比如intersectionObserver就是采用的观察者模式，还有DOM事件的监听等

发布订阅者模式，它和观察者模式有点类似，但是多了一个消息队列作为 中介，比如Vue中的响应式，组件之间进行通信$bus，$emit等都是发布订阅者模式。

装饰器模式，就是可以给一个类，或者对象保证开并原则的同时增加属性，方法等。

**通用组件方案制定**

**弹幕处理**

处理弹幕，主要涉及到一个是弹幕轨道，弹幕数组，弹幕移动(这里涉及到canvas绘图)，还有就是通过websocket获取弹幕以及弹幕缓存队列。

弹幕轨道的计算，因为弹幕没有特别的样式，然后是固定的高度，多轨道是通过窗口大小来除以每条轨道弹幕高度来计算的，然后，在从websocket获取到弹幕以后，将弹幕放入到弹幕缓存队列中，每s从缓存队列取出20条弹幕放入弹幕数组，来起到一个缓冲作用避免弹幕数组数量过大，造成性能问题。在取出弹幕放入弹幕数组之前，先给弹幕包装上对应的top值和随机的left值，通过遍历弹幕数组，拿到每条弹幕的文字，left,top值，并不断增加left的值就可以在ctx上将进行绘制了，但是在每次绘制之前都需要先清除canvas，来避免弹幕的移动过程重复绘制出来。

**弹幕不遮挡人像：**

.video {

  width: 668px;

  height: 376px;

  position: relative;

  -webkit-mask-image: url("mask.svg");

  -webkit-mask-size: 668px 376px;

}

.bullet {

  position: absolute;

  font-size: 20px;

}

**周一，周二，周三复习手写+八股+算法！！！**