# resume\_project

## 自我介绍

面试官你好，我叫唐亚婷，来自四川南充，我的技术栈主要是Vue，做过uniapp端，面向C端用户，B端用户，后台等相关的项目，

## 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.some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)**

<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`)

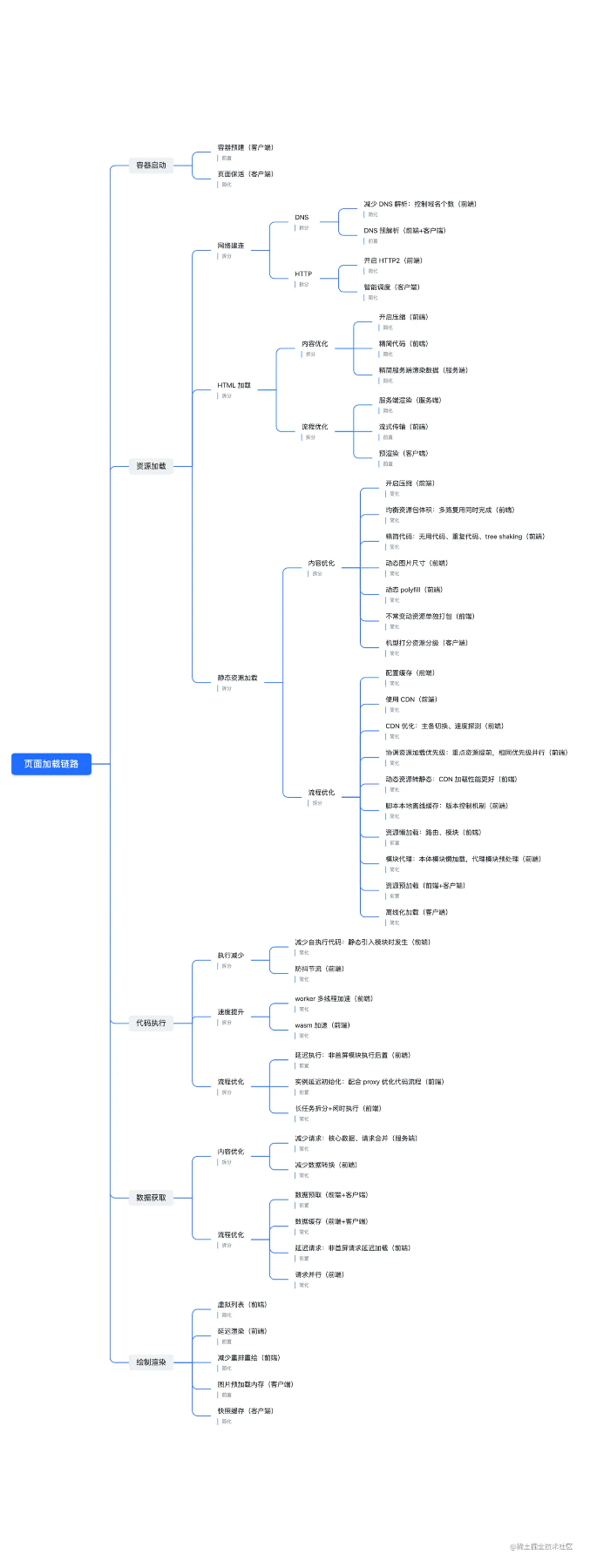
可以拿到所有的.vue文件，然后去匹配我们的路由文件，就对所有的路由进行了按需引入。

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



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。**

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>

// 周三来不及来，周四，周五两天搞！！！  
**8. 项目搭建中的问题**

<https://juejin.cn/post/7156214249369370655?searchId=202312131725530B7E4CFBAB7AEFB8739C>

1. **水印组件封装**
2. **uni.request的封装**
3. **ChatGPT接入问题**
4. **单点登陆**
5. **generator格式接收**

**不着急，周六，周日再搞直播的项目！！！**

**周一，周二，周三复习手写+八股+算法！！！**