优化简历内容

面试速通 https://github.com/arnoldczhang/fe-guide/blob/master/src/fe-interview/README.md

大文件上传 https://www.jianshu.com/p/dd287a60fef5

单点登录 https://blog.csdn.net/qq_46110252/article/details/137470096

脚手架实现 https://blog.csdn.net/m0_57301042/article/details/139219048

`monorepo` 管理组件库 https://blog.csdn.net/qq_36362721/article/details/127993063

`webpack` 热更新 https://blog.csdn.net/weixin_44116302/article/details/137911381

自我介绍

> 面试官你好，我叫王欣，是25届中国科学院大学的硕士，专业是网络与信息安全，技术栈主要以`Vue`为主，对于计算机基础、网络、前端三件套、基本的数据结构与算法、项目的工程化等方面都有一定的了解；去年10月份有过一段实习的经历，从0-1参与了提升气象局日常工作流程效率的后台管理系统，参与了项目需求评审、技术调研以及开发和发布等流程。最近的两个项目是科研管控平台和内部组件库，在科研管控平台中我参与了项目管理、资源管理和权限管理等多个模块，开发了20多个页面，在组件库项目中我独立开发了`confirm`、`message`、瀑布流组件和路由切换组件在内的十多个组件，这就是我大致的情况

专业技能

`ES6` 新特性

> 变量声明
>
> 异步处理
>
> ...

`Vue3` 源码

`webpack` vs `Vite` vs `Rollup`

> `webpack` ——静态资源打包器，将多个资源以模块的方式进行引入并打包成一个`js`文件供客户端使用，支持代码分割(`SplitChunksPlugin`插件)和热更新
>
> `Vite` ——将代码打包成`esm`，借助浏览器进行模块解析，实现按需编译，速度更快，同时借助`esbuild`进行预构建打包不常用的第三方包并进行缓存，速度更快
>
> `Rollup` ——更加简单和轻量级，常用于组件级别的打包

模块热更新

> `webpack`热模块替换是基于`webpack-dev-server`这个开发工具来实现的，首先`webpack`的`wacth`模式会监听系统文件变化，并对修改的模块进行重新编译。通过`webpack-dev-server`与客户端建立`websocket`连接，并将更改后的模块`hash`值推送给客户端，`HotModuleReplacement`拿到`hash`后向服务端发送`fetch`请求获取到最近更新的模块，对比新旧模块进行更新，在决定更新模块后，检查模块之间的依赖关系，更新模块的同时更新模块间的依赖引用

实习经历

- 项目描述

> 此项目主要方便内部人员通过拖拽的方式完成作业流创建与管理，并可以对作业的运行状态进行实时监控，我主要实现了作业管理任务管理以及权限管理，并通过`Echarts`柱状图、折线图、饼图以及性能分析图统计作业的状态
>
> 任务管理、作业管理、日志管理、服务器管理、权限管理
>
> 性能分析图：以时间为横坐标，作业`ID`为纵坐标，通过不同颜色区分作业在每个时间的状态

- 部分功能

> 参与了任务管理、作业管理、日志管理和`Echarts`图表统计功能的开发
>
> 任务管理：任务的增删改查，任务文件上传
>
> 作业管理：作业的增删改查，作业文件上传，作业运行状态图表统计
>
> 日志管理：日志统计，日志增删改查等功能

- 优化工作

> 虚拟列表：只渲染当前页面可视区域的数据，对于不可视的区域不进行渲染，优化页面滚动卡顿问题
>
> 数据懒加载：只加载当前可视区域的数据，对于还未访问到的数据不进行加载
>
> 图片懒加载：对于当前不可视的图片不进行加载，通过`InteractionObserver API`进行判断

- `VueDraggable`调研

> 参考了很多应用的拖拽实现方案最终选择`VueDraggable`第三方拖拽组件来实现项目的作业流拖拽功能
>
> `Vue.Draggable`是一款基于`Sortable.js`实现的`vue`拖拽插件，支持移动设备、拖拽和选择文本、智能滚动，可以在不同列表间拖拽，可以实现单列拖拽、多列拖拽、表格拖拽的功能。其底层是根据`HTML5`提供的`dragEvent`事件来实现
>
> `Sortable.js`——`JS`拖拽库

- `Echarts`使用

  `Echarts`是什么?

> 百度开源的**可视化图表库**，是一个基于`JavaScript`的开源可视化图表库，可以用来创建丰富、交互式的数据可视化图表，以数据为驱动进行图表展示并可以进行动态更新和实时展示

  使用步骤

> 引入`Echarts`第三方库
>
> 通过`echarts.init()`传入一个`element`，初始化一个`dom`节点，创建图表实例
>
> 配置`options`对象，如`title`、`name`、`legend`、`xAxis`、`yAxis`、`series`
>
> 通过`setOpitons()`方法设置`options`对象进行图表渲染

  注意

> 在`echarts`组件销毁之前要手动清除`echarts`实例，否则会导致`echarts`实例占用的内存无法被清除

- 项目难点

科研管控平台

- 项目描述

> 研导安排的项目，维护、升级科学院内部人员信息管理平台，参与项目管理、页面权限管理、资产管理、人力资源管理等八个模块，20多个页面的开发。其中老师可以进行项目审批、项目资源和人员分配、资产管理，学生可以进行项目查看、资产管理。

- 大文件上传

> 实现思路：前端将大文件分片并将切片依此上传至服务器，服务器拿到切片后按顺序组合成完整的文件
>
> 在此过程中有四个需要注意的点分别为：分片、上传、断点续传、完整性校验

分片

> 前端在上传文件时得到的是一个`Blob`二进制对象，通过`Input`或者拖拽导入的`File`对象本质上也是一个`Blob`对象，能够通过`slice`方法对其进行分片

上传

> 将得到的分片列表通过`http`请求上传，可以按照切片顺序以此上传，服务器拿到的是有序的切片信息，也可以以并发的形式进行上传，此时服务器得到的切片是无序的，需要通过一个文件标识表示当前切片的顺序，一般通过`md5`算法根据切片内容计算当前切片的文件标识

断点续传

> 断点续传可以从上次传输失败的文件切片除开始上传，而不必重新上传整个文件
>
> 实现：在文件传输前先向服务端发送一次请求判断当前服务器保存的切片信息，返回保存的切片，客户端只上传未被保存的切片

完整性校验

> 前端与后端约定完整性校验条件，可以通过发送文件标识和切片标识，服务端根据标识信息来判断接收文件的完整性，如果切片已全部接收则进行文件合并操作。

- 页面权限控制

> 用户登录并进行身份验证后服务端返回用户的权限列表，根据权限列表控制其能访问到的路由不同，从而使得不同的用户看到的页面不同，实现权限控制
>
> 如果当前输入的`URl`在权限列表中则正常跳转，否则通过路由导航守卫`router.beforeEach`进行拦截并指定跳转页面

- 身份验证

> 用户登录后服务器会携带`Cookie`信息返回给客户端，客户端保存`Cookie`
>
> 并在下次请求时携带`Cookie`信息进行身份验证
>
> 携带`Cookie`的请求为复杂请求，在请求前会发送一次**预请求**

- 预请求

> 触发条件：额外的请求头/`Content-Type`类型改变
>
> 请求头：
>
> - `Origin`——请求源
> - `Access-Control-Request-Headers`——真实请求的额外请求头
> - `Access-Control-Request-Method`——真实请求的请求方法

> 响应头：
>
> - `Access-Control-Allow-Credentials`——允许请求携带`Cookie`
> - `Access-Control-Allow-Headers`——允许携带的请求头
> - `Access-Control-Allow-Methods`——允许请求的方法
> - `Access-Control-Allow-Origin`——允许跨域的请求源
> - `Access-Control-Expose-Headers`——允许浏览器访问的额外请求头
>
> 浏览器默认访问请求头：`Cache-Control` `Content-Language` `Content-Type` `Expires` `Last-Modified` `Pragma`

- 虚拟列表优化表格展示

  虚拟列表

  > 按需显示，根据滚动容器元素的可视区域来渲染长列表数据中的某一部分数据的技术，虚拟列表就是可视区域渲染的列表。只渲染长列表的一部分数据，不可见的不进行渲染。
  >
  > 1. 不把长列表数据一次性全部直接显示在页面上
  > 2. 截取长列表一部分数据用来填充屏幕容器区域
  > 3. 长列表数据不可视部分使用使用空白占位填充
  > 4. 监听滚动事件根据滚动位置动态改变可视列表
  > 5. 监听滚动事件根据滚动位置动态改变空白填充
  > 6. 分页从服务器请求数据，将一次性请求所有数据变为滚动到底部才再次向服务器发送获取数据的请求

  实现

  > 根据每个可视区域和`item`的高度计算应该渲染的`item`数量，在`ItemList`数据列表中维护一个滑动窗口，表示当前应该渲染的数据，通过滚动来动态改变滑动窗口的位置，实现虚拟列表的渲染

- 功能引导

  > 指引用户如何操作系统
  >
  > 类似于一个`confirm`，通过高亮某个`dom`并进行文字说明，用户可以操作指引或者关闭指引功能
  >
  > 本项目采用第三方库——`diver.js`实现
  >
  > 1. 对外暴露出一个`diver`选对象
  > 2. 将需要指引的功能步骤以对象列表的形式传递给`dirver`对象
  > 3. 需要指定要高亮的`element-id`

- 模糊搜索

  > 作为页面导航的一部分，`search`的主要功能
  >
  > 1. 对当前应用中的所有页面进行搜索
  > 2. `select`的形式展示被搜索的页面
  > 3. 通过点击`options`能够快速进入相应的页面
  >
  > 通过`Fuse.js`对数据源进行模糊搜索
  >
  > 1. 简单、执行速度快、适合一些中小型数据库搜索
  > 2. 数据源——所有`route`数据，将多级路由`list`展开成单条数据

- 自定义菜单处理

> 监听元素的`contextMenu`事件，阻止默认行为
>
> 实现了`MoustEvent`接口，接收到的`contextMenu event`对象中有鼠标的位置属性
>
> 在`menu`菜单中实现了页面刷新、删除当前`tagview`、删除其他功能

- 国际化

> 通过一个变量来控制语言环境，根据变量选择对应的**数据源**，通过一个方法来获取当前语言环境下的指定属性值，该值即为国际化下展示值
>
> 本项目使用`vue-i18n`来实现国际化功能
>
> 1. 安装`vue-i18n`
> 2. 初始化`i18n`实例，并变量、**数据源**传递进实例对象
> 3. 注册`i18n`实例到`vue`实例中

- `ESlint`

> 通过将`JS`代码解析成`AST`，并通过`AST`去匹配`ESLint`配置文件中定义的规则项，根据匹配结果返回相应的描述信息。

内部自研组件库

- `Storybook`

> `Storybook`：开源的构建`UI`组件库使用文档的第三方工具库
>
> 在项目中引入第三方库`StoryBook`，形成组件的描述和说明文档，方便组件库快速上手使用
>
> 为每个组件配置`story`文件，创建`story`实例并传入组件和配置信息生成组件的说明，所见即所得

- 瀑布流组件

> 将`list`列表像瀑布流的形式展示在前端页面，首先把第一行的数据依次排开，记录每一列`dom`的高度，然后把下一个`item`放到最短的那一列上
>
> 通过**绝对定位**的方式定位每个`item`的位置，同时需要记录每一列高度，并在添加`item`后更新此列的高度，`left`值根据屏幕宽度计算；`top`根据列高计算
>
> 注意：`watch`监听数据变化，更新数据后重新布局，监听屏幕宽度变化，更新`item`的`left`值

- 长列表组件

> 通过虚拟列表对长列表的渲染进行优化，通过监听滚动事件并且根据滚动的位置来动态改变可视列表
>
> 并且当滚动到底部时进行在向服务器请求数据，实现分页操作

- 懒加载指令

> 原理：利用浏览器提供的`IntersectionObserver API`来监听当前元素是否可见，对于用户看不到的图片不进行加载，等图片可视时在进行加载
>
> 实现：通过自定义指令的形式实现图片懒加载的能力
>
> 通过自定义指令的`mounted`回调方法可以拿到`el`和`binding`（指令的属性和其他绑定值）
>
> 指定`el.src`的值为指定展位图，即可达到初始不加载的目的
>
> 然后利用`IntersectionObserver`监听当前元素，可见时为`el.src`赋值
>
> 为了方法使用采用`vueuse`提供的`useIntersectionObserver`进行处理

- `message`如何实现

  > 通过方法调用的形式来渲染组件并将其插入到指定的位置，全局组件可动态随时调用
  >
  > 创建`.vue`模板文件，在此组件中完成`comfirm`的主要逻辑
  >
  > 创建`confirm.js`文件暴露出一个方法，利用此方法来实现组件展示
  >
  > 1. 此文件中利用`h`函数将模板文件编译成`VNode`
  > 2. 然后通过`render`函数进行渲染并指定渲染位置

- 动态渲染`vs`模板渲染

  > 前者使用声明式的形式进行组件的动态渲染，更具有灵活性，方便开发人员随时调用定义的全局组件
  >
  > 后者以命令式的形式将组件定义在模板中，更有利于`diff`算法进行对比，减少渲染开销
  >
  > 前者更具有灵活性但是不利于`UI`逻辑较为复杂的组件，每次都要重新渲染，开销较大

- 路由切换组件

  > 通过`routeType`字段来指定当前路由是`进入`或者`退出`状态，并根据此字段动态改变当前路由的过渡效果来实现过场动画
  >
  > 并且通过一个数组来存储当前要保存的组件状态