简历

专业技能

- **网络通信原理**
- 熟悉**ES6+新特性**
- 了解**Vue3源码**
- 熟悉项目构建工具，包括：Webpack、Vite、**Rollup** 等，并拥有**实际项目优化经验**
- **Pinia使用**
- git，**svn使用**

实习经历

- 项目需求分析
- 项目**公共组件**
- 项目**公共方法**
- 项目**部分优化**
- **新技术调研**
    - `XLSX`——`excel`文件解析模块
    - `Fuse.js`——模糊搜索
    - `file-saver`——文件下载
    - 大文件上传下载

项目经历1——中国科学院科研管控平台

- 项目介绍
- 文件上传——**自定义栈**文件批量上传
    - 提供两种文件上传方式：点击上传和拖拽上传
    - 具体流程为：
        - 用户通过点击上传或拖拽的方式上传文件
    - 解析文件——异步操作返回一个`Promise`对象
        - 通过`FileReader`接口来实现对`File`或者`Blob`内容的读取操作
            - `new FileReader()`返回一个`reader`对象
            - `reader.readerAsArrayBuffer()`将读取指定的`File`和`Blob`中的内容
            - 读取成功后`result`中将包含一个表示文件的`ArrayBuffer`对象
        - 读取成功后返回读取到的文件内容
    - 调用文件上传接口进行文件上传
    - 拖拽上传文件
        - `HTML`实现了拖放功能，在要放置的元素中监听拖拽事件，`drop`、`dragover`、`dragenter`
        - 这些方法都能取得`DragEvent`对象，对象的`dataTransfer`属性指向一个`DataTransfer`从中拿到拖拽文件，然后进行文件解析操作
- 个人信息打印
    - **局部打印**详情信息
    - 以表格的形式展示员工详情信息
    - 利用第三方包`vue-print-nb`来实现局部打印功能

    - `vue-print-nb`以指令的形式存在，首先进行指令创建
    - 绑定指令时传入一个配置对象，指定打印店区域、打印标题、打印前回调和打印时回调
- **SSE 协议**、柱形图、日历图、散点图、数据云图展示内容
    - SSE 协议：`server-send Events`是服务器主动向浏览器发送数据的一种方式，使用`http`协议，发送的数据以流的形式传递给客户端，本质上是服务器以流信息的形式与客户端进行通信，完成一次用时很长**下载操作**
    - 客户端API——`EventSource`，新建对象并指定要通讯的服务器url，监听服务器发送的消息。
    - 服务端——`Content-Type`指定为`text/event-steam`，与发送`http`响应串类似
- 如何实现**多语言切换**
    - 一般步骤
        - 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
        - 安装`vue-i18n`
        - 初始化`i18n`实例，并变量、**数据源**传递进实例对象
        - 注册`i18n`实例到`vue`实例中
- 重新构建代码标准化方案，涉及到：ESLint、Prettier、**Husky** 等，并整理出一套**可复用文档**
    - `ESLint`——检查代码是否规范的插件，可以在`.eslintrc.js`文件中配置检查规则
    - `perttier`——代码格式化工具
    - `Husky`——`git hooks`工具
- 封装多个表单模块，3 个可复用的业务表单组件，**封装表单的实现逻辑**
- XLSX 模块——导入、导出表格实现逻辑
    - XLSX 模块——支持解析`base64`、`binary`、`buffer`、`file`类型的文件
    - 导入表格
        - 与文件上传类似
        - 将读取到的文件数据利用XLSX模块进行解析
        - 解析流程
            - `XLSX.read(file)`模块读取返回的`ArrayBuffer`文件并指定文件类型`array`返回`workbook`对象——包含多个工作簿对象
            - 通过`XLSX.utils.sheet_to_json()`将工作簿解析成json对象
            - 将得到的对象进一步处理上传至服务器
    - 导出表格
        - 获取所有用户数据
        - 将数据处理成`XLSX模块`能够处理的`json`格式
        - 利用`XLSX.utils.json_to_sheet()`将`json`对象转化成`Excel`数据格式
        - 下载`Excel`文件——通过第三方库`file-saver`下载文件
        - `file-saver`
            - 一种客户端保存文件的方式
            - 大小有限制，只适合小文件下载
            - 适用于`blob`、`file`和`url`类型的文件
- 功能引导、模糊搜索、自定义菜单实现逻辑
    - 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
            - 对外暴露出一个`diver`选对象
            - 将需要指引的功能步骤以对象列表的形式传递给`dirver`对象
            - 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
            - 对当前应用中的所有页面进行搜索
            - `select`的形式展示被搜索的页面
            - 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
            - 简单、执行速度快、适合一些中小型数据库搜索
            - 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
        - 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

项目经历2——内部自研组件库

- 项目介绍，初衷
- 组件库开发流程，如何进行组件封装
    - 分析组件的功能
    - 根据功能定义应该接收的数据
    - 定义方法并指定触发的时机
- 基于 **Storybook** 构建**组件库文档**，描述组件示例代码，使组件库可快速上手使用
- 瀑布流组件实现
    - 目的：将`list`列表像瀑布流的形式展示在前端页面
    - 分析：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
    - 瀑布流组件：
        - 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
- 图片懒加载实现——`IntersectionObserver`
    - 原理：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
    - 利用浏览器提供的`IntersectionObserver API`来监听当前元素是否可见
    - 由于图片元素较多因此采用自定义指令的方式处理懒加载的逻辑
    - 自定义指令

        - 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
    - 图片懒加载的实现就是通过：指令 和 `IntersectionObserver`的监听来实现的。
- 数据懒加载实现——`IntersectionObserver`
    - 数据懒加载指长列表滚动加载更多数据
    - 创建长列表组件
        - 通过`<slot/>`插入内容，同时底部放置一个`laoding`中的`dom`元素
        - 监听此`dom`元素是否可见，如果可见意味着滑到底部，则加载数据
        - 同时定义两个`props`和一个方法
            - `isLoading`——是否正在加载
            - `isFinished`——数据是否加载完成
            - `onLoad`——当元素可见、数据未加载完成时触发数据加载
- **虚拟任务栈**是指什么
    - 虚拟任务栈的概念来自于**安卓中的任务栈**，在H5页面的的开发中，不同的页面都是独立的html，在进行页面跳转的时候存在页面跳转没有动画效果而且退出的页面无法保存（如数据以及滚动的位置）状态等问题
    - 解决
        - 对于页面跳转的问题，在路由出口处，监听路由的切换，根据当前是进入还是退出页面来添加对应的动画效果
        - 对于组件缓存，采用数组来模拟栈，并在`keep-alive`组件的`include`属性上绑定这个数组来实现组件缓存
        - 对于页面滚动位置无法被主动缓存的问题，记录页面滚动的位置，并在`onActivate`的回调中进行设置
- **dialog**具体实现
    - 首先定义数据
        - 对于`dialog`只需要接收一个`modelValue`表示是否展示，且是双向绑定的
    - 然后定义模板
        - 包含一个**遮罩层**和**内容框**都用`<tansition>`组件进行嵌套并可以指定相应的动画效果
        - 最外层通过`<teleport>`组件进行嵌套并指定挂载的根元素
    - 定义并抛出方法
        - 定义确定和取消按钮并指定其可以进行的操作
- **popover**具体实现
    - 鼠标移入某个区域后显示一个弹出层，并可以指定弹出层的位置
    - 分析：
        - 具有两个插槽：指定触发弹出层的视图和弹出层内容
        - 控制视图是否显示的变量：双向绑定数据
        - 可以指定弹出层的位置：如左上、左下、右上、右下
    - 实现：
        - 监听触发视图的鼠标移入移出事件，控制视图显示
        - 根据传入的位置和弹出层来定位气泡的位置
    - 注意：通过**防抖**来控制气泡显示被频繁触发
        - 防抖：触发事件好多次只执行最后一次

        - 节流：单位时间内只执行一次
- 基于**自定义逻辑**完成瀑布流开发，支持 PC 端与移动端动态切换、支持图片预加载与懒加载的动态切换
    - 通过封装的瀑布流组件+长列表组件实现
        - 瀑布流组件：
            - 将`item`依此排列在第一行，将之后的`item`项依此定位在高度最小的那一列，来实现瀑布流展示
            - 通过一个`map`维护每一列的高度，计算出每个`item`的`top`和`left`之后就要更新当前的高度对象
            - 可以指定图片是否进行预加载
                - 如果进行预加载则需要等图片加载完成之后才能计算每个`item`的位置，进行然后进行渲染
                - 如果不进行预加载则需要提前获取到图片的宽高来计算`item`的位置
        - 长列表组件
            - 主要用来实现数据懒加载功能
            - 通过`slot`插入要渲染的内容，并在底部放置一个`loading`的`dom`元素
            - 用`intersectionObserver`来监听元素是否可见，可见且数据未加载完成时触发数据加载
- **Render 函数**
- **message**、**confirm** 组件具体实现
    - 通过方法调用的方式来渲染组件并将其插入到指定的位置
    - 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
        - 此文件中利用`h`函数将模板文件编译成`VNode`
        - 然后通过`render`函数进行渲染并指定渲染位置
        - 注意函数整体返回一个`promise`，当用户进行操作（点击取消或确认按钮）改变`promise`状态
    - `message`与`comfirm`实现逻辑类似
- 基于自定义栈逻辑完成**虚拟任务栈**构建，配合 KeepAlive + Transform 可实现 APP 跳转逻辑的 H5 切换效果
    - 通过用数组模拟栈来记录当前缓存的组件名，缓存失活组件的状态
    - 具体步骤
        - 通过`routeType`字段来记录**当前组件**是进入还是退出操作，并将其作为全局属性保存在`vuex`中
        - 根据`routeType`类型动态更改`Transform`过渡动画类型
        - 在每次进行`router.push()`和`router.back()`前更改`routeType`属性
        - 在路由前置守卫中监听路由的状态，根据`routeType`进行入栈和出栈操作，始终保存栈内组件的状态
        - 需要单独记录页面滚动的位置，并在组件下次激活时设置