开发文档-前端

本次的前端开发使用vue3的一系列列工具链。作者曾经做过vue开发，但是对一些细节有所遗忘。这个文档既作为开发的记录，也作为一些常用知识的备忘录。

前端的UI在前期设计的时候已经准备好（详见设计文档），因此内容主要对常用的工具链进行展开。

整体的工作流是：搭建需要用到的组件 ----> 设置好views 用vue-router设置页面跳转 ----> 用vuex做好全局状态管理 ----> 和后端对接实现最终结果

源代码目录结构如下：

```
├── App.vue
├── assets
│   ├── ...
├── components              // 自己搭建的可复用组件
│   ├── TopBar.vue
│   ├── login
│   │   └── LoginCard.vue
│   ├── main
│   │   ├── DeviceCard.vue
│   │   ├── HistoChart.vue
│   │   └── LineChart.vue
│   ├── map
│   └── register
│       └── RegisterCard.vue
├── main.js
├── router
│   └── index.js
├── store
│   └── index.js
├── style.css
└── views                   // 视图
    ├── LoginView.vue
    ├── MainView.vue
    ├── MapView.vue
    └── RegisterView.vue
```

组件规划

页面 views

1. LoginView✅
2. MainView✅
3. MapsView✅

组件库

1. Common
    1. Topbar ✅

       注意登陆前后两种形态 ✅
2. LoginView
    1. LoginCard ✅
3. MainView
    1. 设备卡片 两种size(使用props来控制) ✅
        1. 设备类型
            1. light✅
            2. music✅
            3. door_window✅
            4. curtain✅
            5. tv✅
            6. temperature✅
            7. car✅
            8. other✅
        2. DeviceCard✅
    2. DeviceInfo 设备信息页✅
    3. AddDevice 添加设备页✅
    4. HistoChart 柱状图 使用d3实现✅
    5. LineChart 折线图 使用d3实现✅
4. MapsView
    1. LeftBar✅
    2. CurrentMap✅
    3. HistoryMap✅

vue-router

在main.js中：

```
import router from "./router"
const app = createApp(App)
app.use(router)
```

在src下建立router目录，增加文件index.js

配置用到的路由

```
import { createRouter, createWebHistory } from 'vue-router'
import store from '../store'

const router = createRouter({
    history: createWebHistory(),
    routes: [
        {
            path: '/',
            name: 'login',
            component: ()=>import('../views/LoginView.vue')
        },
        {
            path: '/main',
            name: 'main',
            component: ()=>import('../views/MainView.vue')
        },
        {
            path: '/map',
            name: 'map',
            component: ()=>import('../views/MapView.vue')
        },
        {
          path:'/register',
          name:'register',
          component:()=>import('../views/RegisterView.vue')
        }
    ]
})



export default router
```

更改App.vue

```
<script setup>

</script>

<template>
  <router-view></router-view>
</template>

<style>
@import './style.css';
</style>
```

在视图组件中可以用`this.$router.push('/main')`来进行路由

vuex 和 localstorage

vuex用来全局化管理用户的信息，登陆状态，包括用户id，用户名，邮箱。localstorage用来本地化保存登陆状态，防止因为刷新而丢失登陆状态。

vuex首先要在main.js中配置

```
import store from './store'
const app = createApp(App)
app.use(store)
```

在src下建立store目录，增加文件index.js

配置用到的状态和mutations。mutations即定义好的更改状态的方法。

```
import {createStore} from "vuex"

export default createStore({
    state:{
        loginState: 0,
        username: "",
        email: "",
        user_id: null
    },
    mutations: {
        changeLogin(state, {loginState, username, email, user_id}){
            Object.assign(state,  {loginState, username, email, user_id})
        }
    }
})
```

在每一个页面加载时，如果发现vuex中的登陆状态为未登陆，需要到localstorage中去查找是否有存放的登陆状态，根据登陆状态来判断是否可以跳转到当前页面。如果没有登陆，跳转到用户主页等页面。在每一次登陆成功的时候把登陆状态存放到localstorage中，每一次退出登录时从localstorage中清空。

```
//mainView.vue 中的部分代码
// 进行登陆状态检查
if(this.$store.state.loginState===0){
  let loginInfo = JSON.parse(localStorage.getItem('loginInfo'))
  if(loginInfo===null){
    this.$router.push('/')
    return
  }else{
    this.$store.commit('changeLogin', loginInfo);
  }
}
```

```
// loginCard.vue 中的部分代码
if (response.ok) {
  const jsonResponse = await response.json();
  // 根据服务器返回的status和message进行响应
  // 更新user_id字段
  if (jsonResponse.status === 'success') {
    let loginInfo = {
      loginState: 1,
      username: jsonResponse.username,
      email: jsonResponse.email,
      user_id: jsonResponse.user_id
    }
    this.$store.commit('changeLogin', loginInfo);
    localStorage.setItem('loginInfo',JSON.stringify(loginInfo))
    this.$router.push('/main');
  } else {
    alert(`注册失败：${jsonResponse.message}`);
  }
} else {
  alert(`网络错误：${response.status}`);
}
```

leaflet GIS地图服务

leaflet是一个轻量级的图像服务库，可以从各大地图厂商获得数据源，包括百度地图，高德地图等等，然后通过给定的参数渲染地图瓦片图。leaflet也提供了进行位置标记，路线标记的功能。

使用leaflet的时候要测试注意一些特殊情况，测试的时候发现缩放会影响路线的显示，后来发现和vue的更新机制有关，加上相关的配置后解决。

```
//MapView.vue 的部分代码
import L from 'leaflet'
```

```
import 'leaflet/dist/leaflet.css'

initMap(){
  let map = L.map("map", {
    center: [30.23086372926422, 120.21121938720704], // 中心位置
    zoom: 10, // 缩放等级
    // zoomAnimation:false,fadeAnimation:true,markerZoomAnimation:true,
    zoomControl: true //缩放控件
  });
  L.Popup.prototype._animateZoom = function (e) {//解决缩放时的变形问题
    if (!this._map) {
      return
    }
    var pos = this._map._latLngToNewLayerPoint(this._latlng, e.zoom, e.center),
        anchor = this._getAnchor()
    L.DomUtil.setPosition(this._container, pos.add(anchor))
  }

  this.map = map; // data上需要挂载
  L.tileLayer(
    "http://wprd04.is.autonavi.com/appmaptile?lang=zh_cn&size=1&style=7&x={x}&y={y}&z=
{z}"
  ).addTo(map) // 加载底图
},
  moveToLocation(latitude, longitude) {
    this.map.setView([latitude, longitude], this.map.getZoom());
  },
    addMarkerAndPopup(latitude, longitude, label) {
      let marker = L.marker([latitude, longitude]).addTo(this.map);
      let popup = L.popup().setContent(label);
      marker.bindPopup(popup).openPopup();
    },
      showLabel(latitude,longitude){
        this.map.eachLayer((layer) => {
          if (layer instanceof L.Marker && layer.getLatLng().equals([latitude,
longitude])) {
            layer.openPopup();
          }
        });
      },
        async showHistory(device){
          // get history by id
          await this.getHistoryList(device)
          try{
            this.history_list.sort((a, b) => {
              const timeA = new Date(a.timestamp);
              const timeB = new Date(b.timestamp);
              // 根据时间从早到晚排序
              return timeA - timeB;
```

```
            });
            this.historyDevice = device
            this.map.eachLayer((layer) => {//去除标记和标签
              if (layer instanceof L.Marker &&
!layer.getLatLng().equals([device.latitude, device.longitude])) {
                layer.closePopup();
                this.map.removeLayer(layer);
              }
            });
            const historyLatLngs = this.history_list.map(({ latitude, longitude }) =>
[latitude, longitude]);
            if (this.historyDevicePolyline!=null) {
              this.map.removeLayer(this.historyDevicePolyline);
            }
            this.historyDevicePolyline = L.polyline(historyLatLngs, { color: 'red'
}).addTo(this.map);
            if (historyLatLngs.length > 0) {
              const [firstLat, firstLng] = historyLatLngs[0];
              this.moveToLocation(firstLat, firstLng);
            }
            for (const position of this.history_list) {
              const options = { year: 'numeric', month: '2-digit', day: '2-digit',
hour: '2-digit', minute: '2-digit', second: '2-digit' };
              const formattedTime = (new
Date(position.timestamp)).toLocaleDateString('en-US', options);
              const label = `<b>Time:</b> ${formattedTime}`;
              this.addMarkerAndPopup(position.latitude, position.longitude, label);
            }
          }catch(e){
            console.log(e)
          }

        },
```

D3 构建直方图和折线图

D3是一个用来构建向量图的库，使用起来和canvas很类似，都是通过往一个容器元素里面放置向量图来实现相关功能的。注意设置style的时候有时候需要用向量图的属性来设计，比如填充颜色是fill。

```
// HistoChart.vue 的部分代码

drawHistogram(){
  d3.select('.d3HisChart svg').remove();
  // 统计类型
  const typeCounts = this.devices.reduce((acc,device)=>{
    const type = device.type;
    acc[type] = (acc[type]||0) + 1;
```

```
    return acc;
  },{})


  // 产生渐变颜色
  const colorScale = d3.scaleLinear().domain([0, 1]).range(['#fff', '#144E2E']);

  // 生成渐变颜色数组
  const lenghthOfType = Object.keys(typeCounts).length;
  const gradientColors = Array.from({ length: lenghthOfType }, (_, index) => {
    const t = index / (lenghthOfType - 1); // 生成 0 到 1 之间的分数
    return colorScale(t);
  });

  // 随机打乱数组顺序
  const shuffledColors = d3.shuffle(gradientColors);
  const data = Object.keys(typeCounts).map((type, index) => ({
    type: this.deviceType.find(device => device.type === type).cn,
    count: typeCounts[type],
    color: shuffledColors[index % lenghthOfType] // 循环使用颜色数组
  }));


  //设置svg的尺寸
  const width = 350;
  const height = 200;
  const margin = {top:20,right:0,bottom:20,left:35}

  //创建svg容器
  const svg = d3.select('.d3HisChart')
  .append('svg')
  .attr('width', width + margin.left + margin.right)
  .attr('height', height + margin.top + margin.bottom)
  .append('g')
  .attr('transform', `translate(${margin.left},${margin.top})`);

  // 设置比例尺
  const x = d3.scaleBand().range([0, width]).padding(0.3).domain(data.map(d =>
d.type));
  const y = d3.scaleLinear().range([height, 0]).domain([0, d3.max(data, d =>
d.count)]);


  // 绘制条形图
  svg.selectAll('.bar')
    .data(data)
    .enter().append('rect')
    .attr('class', 'bar')
    .attr('x', d => x(d.type))
```

```
    .attr('width', x.bandwidth())
    .attr('y', d => y(d.count))
    .attr('height', d => height - y(d.count))
    .attr('fill', d => d.color);

  // 绘制x轴
  svg.append('g')
    .attr('transform', `translate(0,${height})`)
    .call(d3.axisBottom(x))
    .style('color','#144E2E');

  // 绘制y轴
  svg.append('g')
    .call(d3.axisLeft(y))
    .style('color','#144E2E');
}
```

```
// LineChart.vue的部分代码
async fetchTrafficData(){
  let newData
  try{
    const response = await fetch(`http://127.0.0.1:8000/mqtt/bytes`,{
      method: 'GET',
    });
    if (response.ok) {
      const jsonResponse = await response.json();
      newData = jsonResponse.sent_bytes/1000
    } else {
      alert(`网络错误: ${response.message}`);
    }
  } catch (error) {
    console.log(error)
    alert(`网络错误`);
  }
  // 保持 trafficData 长度为 10
  this.trafficData.shift(); // 去掉第一个数据
  this.trafficData.push(newData); // 将新数据添加到末尾; // for test
},
  setupChart(){
    //设置svg的尺寸
    const width = 350;
    const height = 200;
    const margin = {top:20,right:20,bottom:20,left:30};
    //创建svg容器
    const svg = d3.select('.d3LineChart')
    .append('svg')
    .attr('width', width + margin.left + margin.right)
    .attr('height', height + margin.top + margin.bottom)
    .append('g')
```

```
    .attr('transform', `translate(${margin.left},${margin.top})`);
    // 添加横纵轴
    svg.append('g').attr('class', 'x-axis').attr('transform',
`translate(0,${height})`).style('color','#144E2E');;
    svg.append('g').attr('class', 'y-axis').style('color','#144E2E');;
    svg.append('text')
      .attr('y', -15)
      .attr('x', 0)
      .attr('dy', '1em')
      .style('font-size', '12px')
      .style('fill','#144E2E')
      .text('单位 (KB)');


    // 定义线段函数
    const line = d3
    .line()
    .x((d, i) => xScale(i))
    .y(d => yScale(d));

    // 将折线添加到图表中
    svg.append('path')
      .attr('class', 'line')
      .attr('fill', 'none')
      .attr('stroke', '#144E2E');

    this.updateChart(); // 初始化图表
  },
    updateChart() {
      const svg = d3.select('.d3LineChart').select('svg').select('g');

      // 更新 x 轴和 y 轴的比例尺的域（domain）
      const xScale = d3.scaleLinear().domain([0, this.trafficData.length -
1]).range([0, 350]);
      const yScale = d3.scaleLinear().domain([0, d3.max(this.trafficData)]).range([200,
0]);

      // 选择 x 轴和 y 轴，并更新它们的比例尺
      svg.select('.x-axis').call(d3.axisBottom(xScale));
      svg.select('.y-axis').call(d3.axisLeft(yScale));

      // 更新折线的路径
      svg.select('.line').datum(this.trafficData).attr('d', d3.line().x((d, i) =>
xScale(i)).y(d => yScale(d)));
    },
```

手机端适配

在每一个view中对设备类型进行判断

```
created:function(){
  // 进行设备类型检查
  var u = navigator.userAgent
  var isAndroid = u.indexOf('Android') > -1 || u.indexOf('Adr') > -1 //android
  var isiOS = !!u.match(/\(i[^;]+;( U;)? CPU.+Mac OS X/) //ios
  if (isiOS || isAndroid) {
    this.mobileFlag=1;
  } else {
    this.mobileFlag=0;
  }  //set mobile
},
```

然后将判断结果通过属性传递到每一个组件中，在每一个组件里对不同的设备进行对应的手机端效果设计。