****

**实**

**验**

**报**

**告**

**课 程 可视语言与信息可视化**

**题 目 基于Vue框架的d3可视化demo**

**分 数**

**学院名称 软件工程学院**

**专 业 软件工程**

**学生姓名 杨宇鑫**

**学 号 3022207128**

**年 级 大学三年级**

**班 级 软件工程4班**

**目录**

[一、 实验目的 1](#_Toc18466)

[二、 实验内容 1](#_Toc19363)

[三、 实验步骤 1](#_Toc5215)

[1. 安装Vue CLI 1](#_Toc6433)

[2. 创建新项目 1](#_Toc9195)

[3. 安装d3依赖 2](#_Toc22152)

[4. 创建并修改组件App.vue 3](#_Toc11058)

[5. 创建并修改力导向图组件ForceGraph.vue 4](#_Toc30565)

[6. 创建并修改柱状图组件BarChart.vue 5](#_Toc17709)

[7. 运行项目 7](#_Toc20960)

[四、 实验结果 8](#_Toc25108)

[五、 实验结论 9](#_Toc1344)

[六、 源代码 9](#_Toc31833)

## 实验目的

使用vue框架搭建一个前端页面 , 应用d3.js技术展示所给数据 , 网页分为左右两个部分 , 右侧的部分展示所有数据的力导向图 , 左侧的数据是鼠标在右侧拖动左键框选选中的节点的生成的柱形图

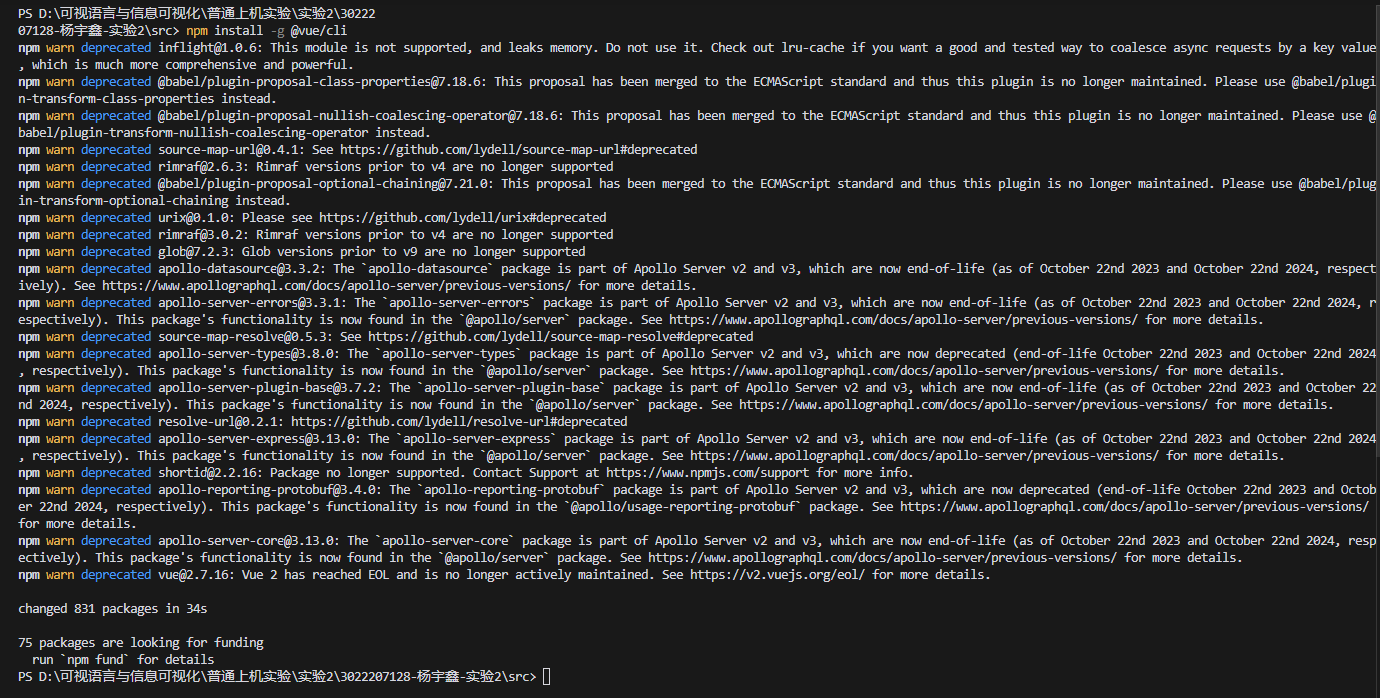
## 实验内容

使用vue框架搭建一个前端页面 , 应用d3.js技术展示所给数据 , 网页分为左右两个部分 , 右侧的部分展示所有数据的力导向图 , 左侧的数据是鼠标在右侧拖动左键框选选中的节点的生成的柱形图

## 实验步骤

1. 安装Vue CLI

执行命令npm install -g @Vue/cli

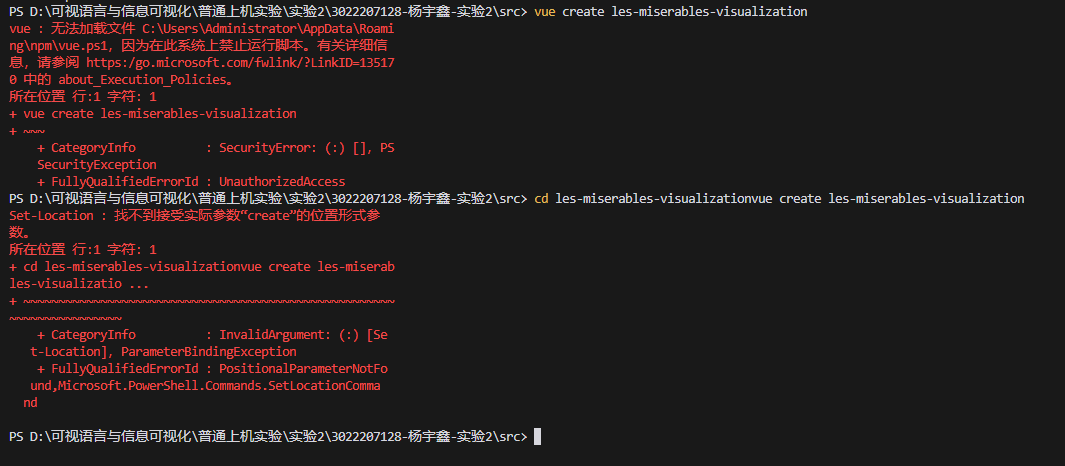


1. 创建新项目

vue create les-miserables-visualization

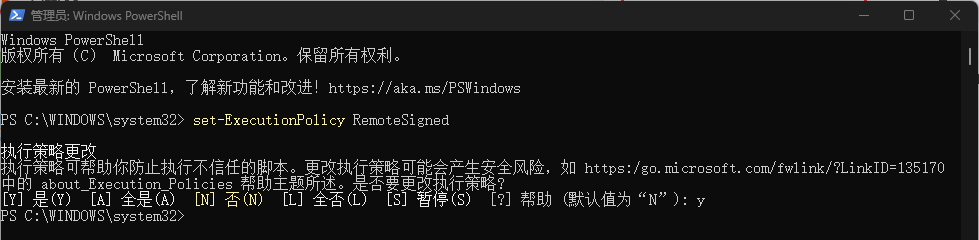
cd les-miserables-visualization

在vscode中运行报错了

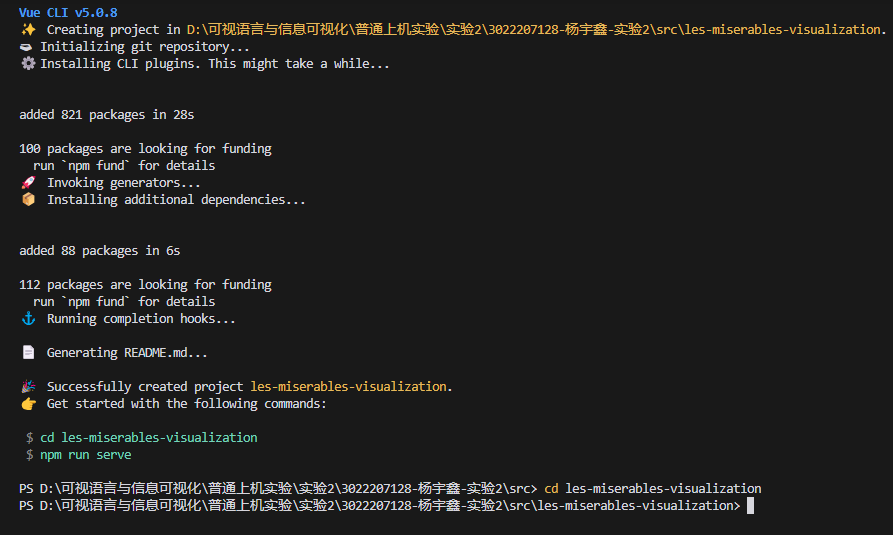


解决方案:

以管理员的身份启动PowerShell , 执行命令

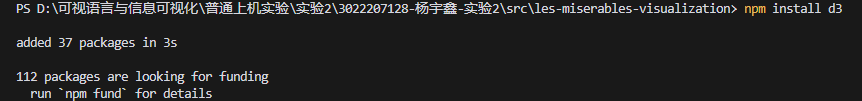


再次执行vue初始化的命令 , 成功 :

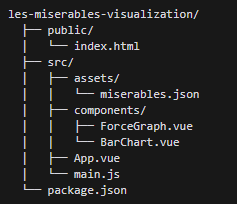


1. 安装d3依赖

执行命令npm install d3



此时项目的结构:



1. 创建并修改组件App.vue

文件结构分析

创建的Vue 单文件组件包含三个主要部分：

<template> - 视图结构

<script> - 组件逻辑

<style> - 样式定义

功能分析

页面被分为左右两个面板

实现了力导向图和柱状图的联动展示

组件结构

App.vue

├── ForceGraph (右侧面板)

└── BarChart (左侧面板)

数据流

父组件 App.vue 维护 selectedNodes 状态

ForceGraph 组件通过事件 @nodesSelected 向上传递选中的节点

BarChart 组件通过 props :selectedNodes 接收选中的节点数据

主要方法

methods: {

handleNodesSelected(nodes) {

  this.selectedNodes = nodes

}

}

布局设计

采用 flex 布局

左侧面板占40%宽度，用于显示柱状图

右侧面板占60%宽度，用于显示力导向图

两个面板都可以滚动(overflow: auto)

1. 创建并修改力导向图组件ForceGraph.vue

组件初始化数据:

data() {

    return {

      simulation: null,    // 存储力导向图模拟实例

      svg: null,          // 存储SVG元素

      brush: null,        // 存储刷选工具实例

      selectedNodes: new Set()  // 存储选中的节点

    }

}

initializeGraph() 方法:

获取容器尺寸并创建SVG

创建力导向图模拟，包含三个力：

  link力：连接相关节点

  charge力：节点间的排斥力，strength(-50)表示排斥强度

  center力：将整个图形拉向中心

绘制连接线：

  使用line元素

  线条宽度根据value值计算

  设置透明度0.6

绘制节点：

  使用circle元素

  半径为5

  颜色根据group值从d3.schemeCategory10获取

添加刷选功能

设置tick事件更新节点和连接线位置

drag() 方法:

  dragstarted：拖拽开始

  设置模拟器的alphaTarget使其"热身"

  固定节点的初始位置

  dragged：拖拽中

  更新节点的固定位置(fx, fy)

  dragended：拖拽结束

  停止模拟器的"热身"

  释放节点的固定位置

brushended() 方法:

  处理刷选结束事件

计算选中区域内的节点：

  通过比较节点坐标是否在选择框范围内

  使用d3.select过滤符合条件的节点

触发nodesSelected事件通知父组件

tick 更新逻辑:

 在每一帧更新：

 连接线：更新起点(x1,y1)和终点(x2,y2)坐标

 节点：更新圆心(cx,cy)坐标

实现细节：1. 使用了 Vue 的生命周期钩子 mounted 在组件挂载后初始化图形

使用 ref 获取容器DOM元素

利用D3的数据绑定机制处理节点和连接线

通过事件委托处理交互（拖拽和刷选）

1. 创建并修改柱状图组件BarChart.vue

组件结构

<template>

  <div class="bar-chart" ref="chartContainer"></div>

</template>

组件属性和监听

props: {

  selectedNodes: {

    type: Array,

    default: () => []

  }

},

watch: {

  selectedNodes: {

    handler() {

      this.updateChart()

    },

    deep: true

  }

}

接收一个 selectedNodes 数组作为属性

使用深度监听来观察数组变化，当数据变化时重新渲染图表

核心方法 updateChart()

a. 初始化设置

const container = this.$refs.chartContainer

d3.select(container).selectAll('\*').remove()  // 清除现有图表

if (this.selectedNodes.length === 0) return   // 无数据则返回

const width = container.clientWidth

const height = container.clientHeight

const margin = { top: 20, right: 20, bottom: 100, left: 60 }

获取容器引用并清除旧内容

设置图表尺寸和边距

b. 创建 SVG 容器

const svg = d3.select(container)

  .append('svg')

  .attr('width', width)

  .attr('height', height)

c. 设置比例尺

const x = d3.scaleBand()  // 横轴（离散型）

  .domain(this.selectedNodes.map(d => d.id))

  .range([margin.left, width - margin.right])

  .padding(0.1)

const y = d3.scaleLinear()  // 纵轴（连续型）

  .domain([0, d3.max(this.selectedNodes, d => d.group)])

  .nice()

  .range([height - margin.bottom, margin.top])

x轴使用 scaleBand 处理离散数据

y轴使用 scaleLinear 处理连续数值

d. 绘制坐标轴

// X轴

svg.append('g')

  .attr('transform', `translate(0,${height - margin.bottom})`)

  .call(d3.axisBottom(x))

  .selectAll('text')

  .attr('transform', 'translate(-10,10) rotate(-45)')  // 文本旋转45度

  .style('text-anchor', 'end')

  .style('font-size', '12px')

// Y轴

svg.append('g')

  .attr('transform', `translate(${margin.left},0)`)

  .call(d3.axisLeft(y))

e. 绘制柱状图及交互效果

svg.append('g')

  .selectAll('rect')

  .data(this.selectedNodes)

  .join('rect')

  .attr('x', d => x(d.id))

  .attr('y', d => y(d.group))

  .attr('height', d => y(0) - y(d.group))

  .attr('width', x.bandwidth())

  .attr('fill', d => d3.schemeCategory10[d.group])

交互效果包括：

鼠标悬停效果

柱子透明度变化

显示提示框

显示数值信息

提示框实现

  // 添加提示框背景

   svg.append('rect')

     .attr('class', 'tooltip-bg')

     // ... 设置背景样式

   // 添加提示文本

   svg.append('text')

     .attr('class', 'tooltip')

     // ... 设置文本内容和样式

鼠标移出效果

恢复柱子透明度

移除提示框

生命周期钩子

mounted() {

  this.updateChart()

}

组件挂载后初始化图表

样式设置

.bar-chart {

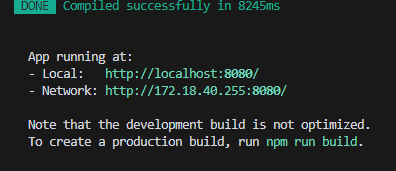
  width: 100%;

  height: 100%;

}

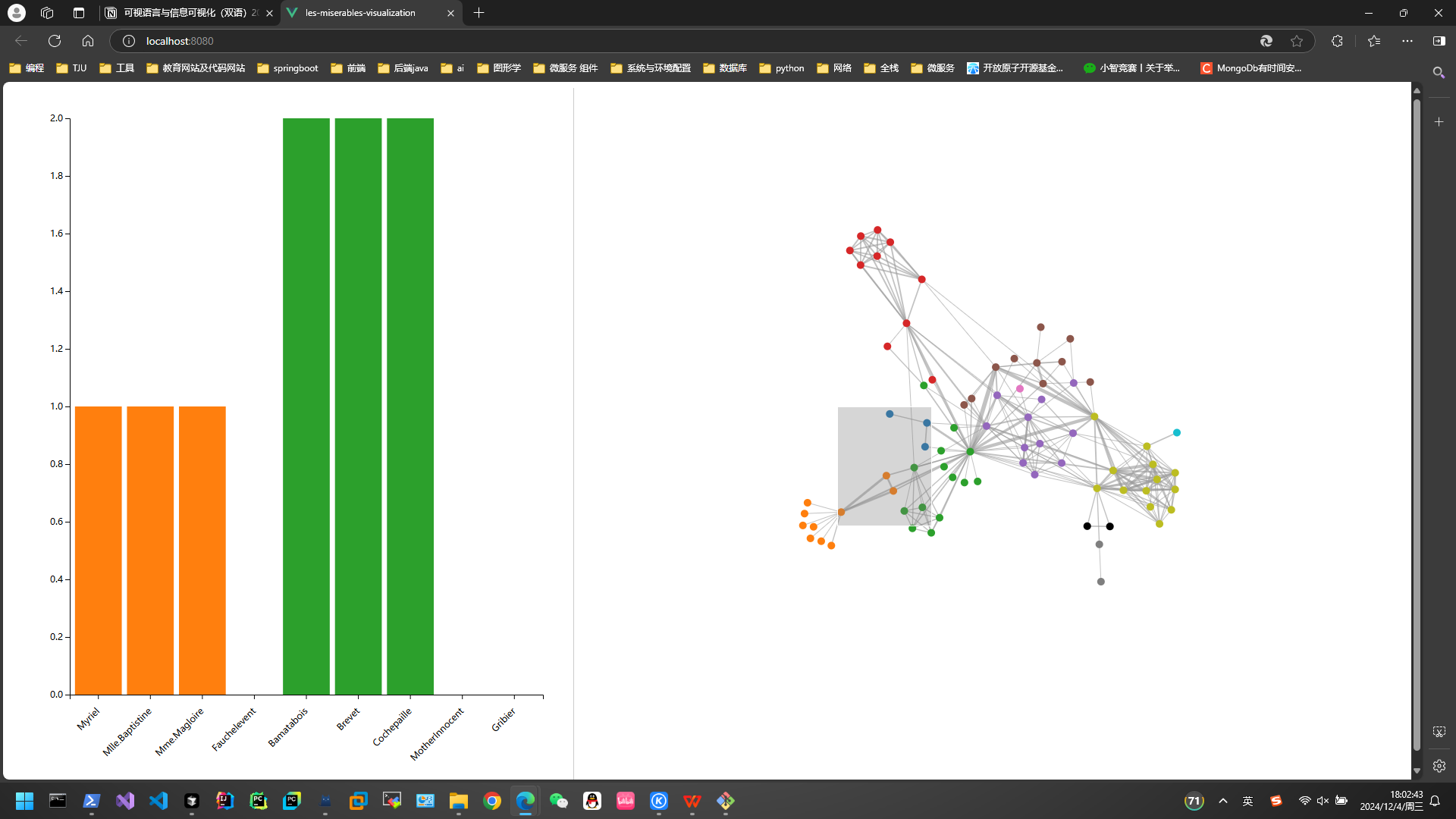
1. 运行项目

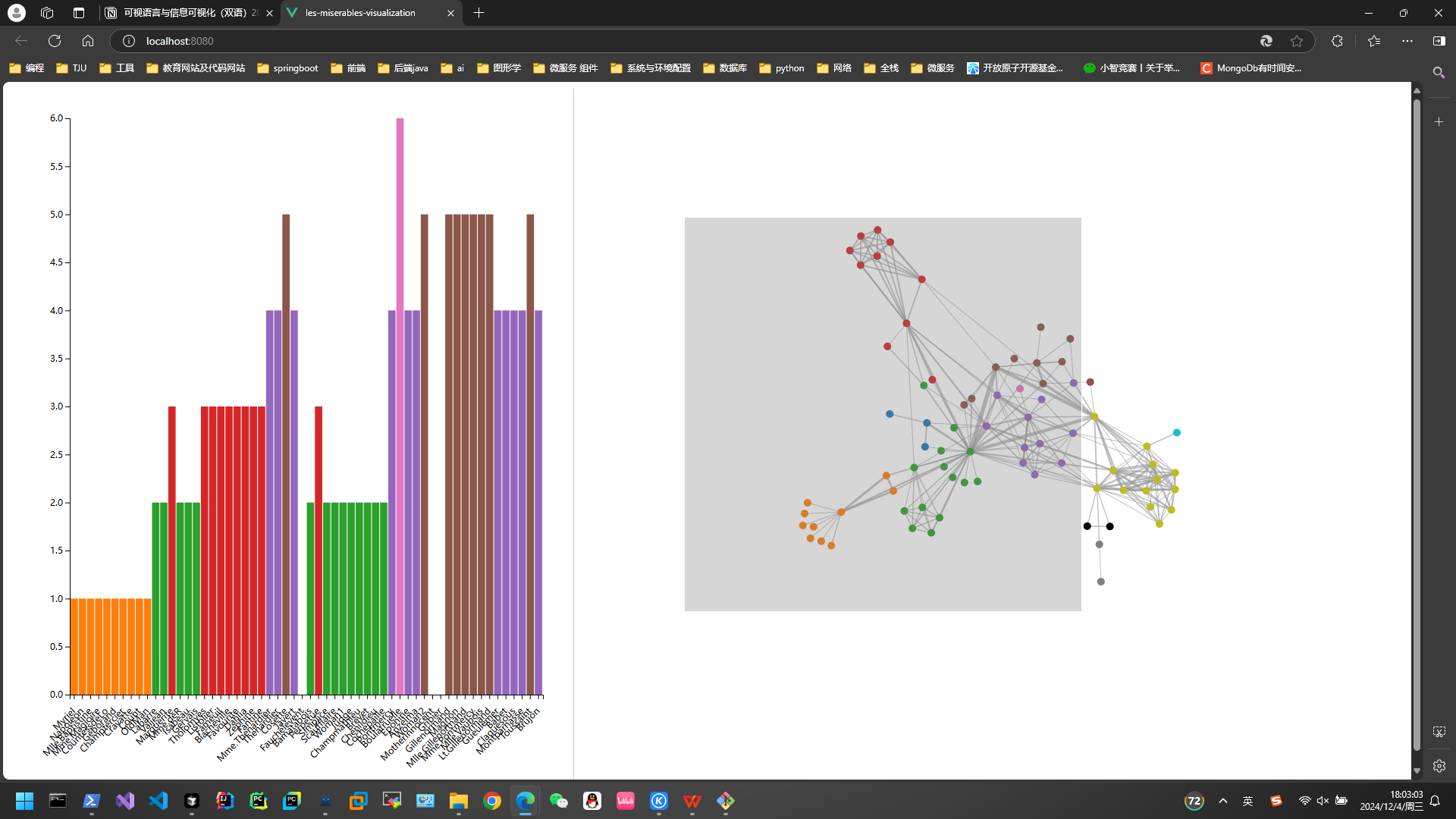
执行命令npm run serve

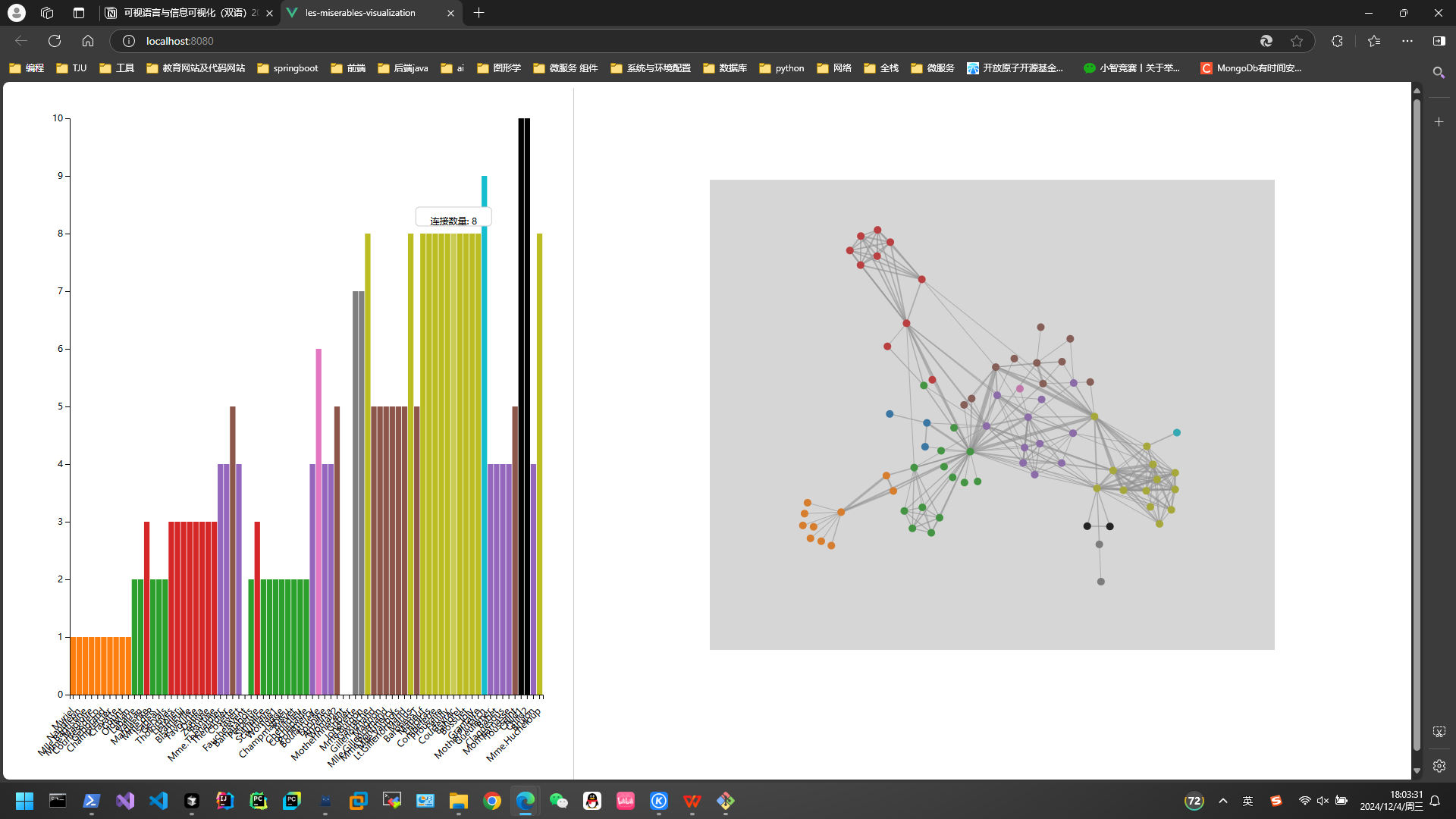


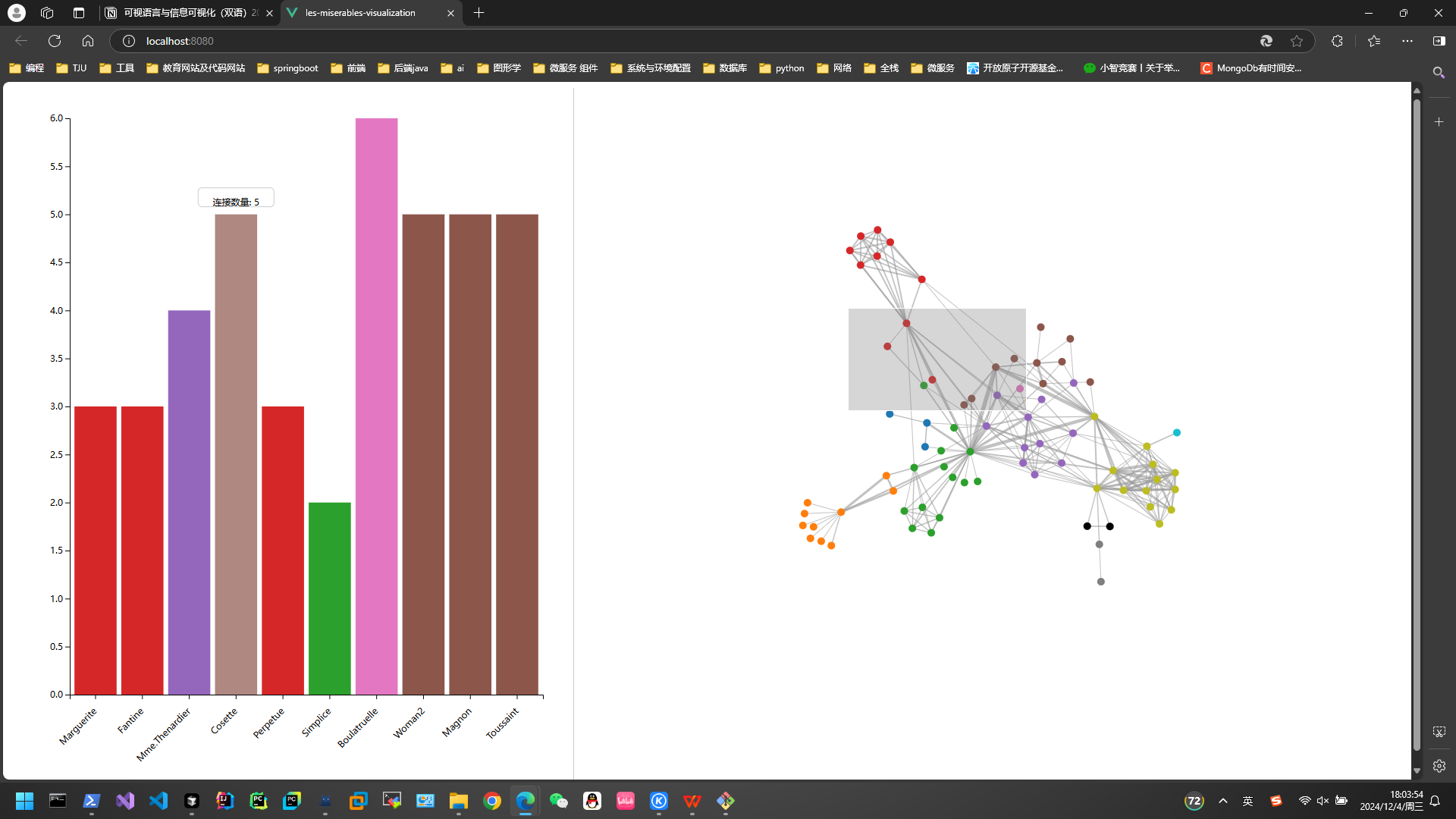
成功运行

## 实验结果









## 实验结论

## 源代码

源代码打包在文件中