目录

[电商平台数据可视化实时监控系统 2](#_Toc26203)

[技术使用： 2](#_Toc16974)

[1、 Html, css, js 2](#_Toc11900)

[2、 ECharts 2](#_Toc31248)

[3、 Vue, VueX, Router, Webpack 2](#_Toc30317)

[4、 WebSocke 2](#_Toc30139)

[项目主要计算实现方式 2](#_Toc9617)

[项目灵感 2](#_Toc11269)

[可视化的实现方式 2](#_Toc14647)

[ECharts 3](#_Toc18597)

[Echarts特点 3](#_Toc27452)

[1、 丰富的可视化类型 3](#_Toc28418)

[2、 多种数据格式的支持（key-value数据格式 二维表 TypedArray格式） 3](#_Toc18048)

[3、 流数据的支持（流数据的动态渲染，增量渲染技术） 3](#_Toc26439)

[4、 移动端优化 3](#_Toc24611)

[5、 跨平台使用 3](#_Toc12586)

[6、 绚丽的特效 3](#_Toc25523)

[7、 三维可视化 3](#_Toc4591)

[Koa2的特点 4](#_Toc20903)

[1、 Koa2是基于Node.Js平台的web开发框架 4](#_Toc8038)

[2、 支持async/await 洋葱模式的中间件 4](#_Toc2786)

[后台项目主要实现对的功能 4](#_Toc6135)

[1、 计算服务器处理请求的总耗时 4](#_Toc22797)

[2、 在响应头上加响应内容的mime类型 4](#_Toc6140)

[3、 根据URL读取指定目录下的文件内容 4](#_Toc25355)

[后台项目对的实现步骤 4](#_Toc32503)

[1、 项目准备 4](#_Toc13787)

[2、 总耗时中间件 4](#_Toc11108)

[3、 响应头中间件 4](#_Toc3482)

[4、 业务逻辑中间件 4](#_Toc10809)

[5、 允许跨域 4](#_Toc10117)

[前端项目准备 5](#_Toc8023)

[项目准备的表格： 6](#_Toc27952)

[1、 商家销售系统（横向柱状图） 6](#_Toc5741)

[2、 销售趋势图表（折线图） 6](#_Toc2633)

[3、 商家发布模块（地图+散点图） 6](#_Toc10280)

[4、 销售排行模块（柱状图） 6](#_Toc18851)

[5、 热销商品占比模块（饼图） 6](#_Toc32158)

[6、 库存与销量模块（圆环饼图） 6](#_Toc15192)

[一、 商家销售系统（横向柱状图）的实现 6](#_Toc19543)

[实现效果： 19](#_Toc30022)

[数据库表格： 19](#_Toc30565)

[感谢观看 25](#_Toc2102)

电商平台数据可视化实时监控系统

技术使用：

1. Html, css, js
2. ECharts
3. Vue, VueX, Router, Webpack
4. WebSocke

# 项目主要计算实现方式

该项目为电商平台可视化实时监控系统，可以保证实时获取数据进行分析，可以支持大屏显示，能自适应分辨率，支持联动效果，一端操作，多端联动展示。该项目只要使用ECharts用来完成整个图表的绘制，然后结合Vue，Vue Router，VueX，Webpack等一系列的技术，然后使用Axios来完成前后端数据的交互，以及使用WebSocket来完成前后端数据的推送。后端通过koa2这个框架来进行后端服务器的开发。

# 项目灵感

前端就是为了数据可视化，那么怎样才能把数据以更直观的方式来显示出来呢，那就是以图表的方式把数据展现出来，正所谓文不如表，表不如图，人们对图的敏感程要比苍白无力的文字好的不是一清半点，数据可视化的好处就是能有效的传达数据里面的信息，能跟清楚的看到隐藏在数据中的信息（将数据以图表的形式呈现，更有效的传递数据中的信息）

# 可视化的实现方式

1、报表类（简单，但是非常的不灵活，图表一旦生成，就固定不变了，如果数据发生改变，就要重新生成报表）

Excle

水晶报表

1. 商业智能BI（相比报表，可以根据数据分析一些决策依据，可以帮助企业做出明智的业务经营的决策，）

Microsoft BI

Power-Bi

1. 编码类

ECharts.js

D3.js

# ECharts

ECharts，一个使用 JavaScript 实现的开源可视化库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖轻量级的矢量图形库 [ZRender](https://github.com/ecomfe/zrender" \t "_blank)，提供直观，交互丰富，可高度个性化定制的数据可视化图表。  
ECharts 提供了常规的折线图、柱状图、散点图、饼图、K线图，用于统计的盒形图，用于地理数据可视化的地图、热力图、线图，用于关系数据可视化的关系图、旭日图，多维数据可视化的平行坐标，还有用于 BI 的漏斗图，仪表盘，并且支持图与图之间的混搭。  
2018年3月全球著名开源社区Apache宣布百度ECharts进入Apache孵化器。

Echarts特点

1. 丰富的可视化类型
2. 多种数据格式的支持（key-value数据格式 二维表 TypedArray格式）
3. 流数据的支持（流数据的动态渲染，增量渲染技术）
4. 移动端优化
5. 跨平台使用
6. 绚丽的特效
7. 三维可视化

要为前端开发内容必须先搭建后台，所以第一步先是搭建后台，后台使用koa2进行后台的开发，koa2是一个web服务器的框架，因为图表的数据都要从服务器获取过来。

# Koa2的特点

1. Koa2是基于Node.Js平台的web开发框架
2. 支持async/await 洋葱模式的中间件

****koa2****是由 Express 原班人马打造的，致力于成为一个更小、更富有表现力、更健壮的 Web 框架。 使用 koa 编写 web 应用，可以免除重复繁琐的回调函数嵌套， 并极大地提升错误处理的效率。koa 不在内核方法中绑定任何中间件， 它仅仅提供了一个轻量优雅的函数库，使得编写 Web 应用变得得心应手。开发思路和express差不多，最大的特点就是可以避免异步嵌套。

****koa2****利用ES7的async/await特性，极大的解决了我们在做nodejs开发的时候异步给我们带来的烦恼。

k2a命令

npm init -y

npm install koa

# 后台项目主要实现对的功能

1. 计算服务器处理请求的总耗时
2. 在响应头上加响应内容的mime类型
3. 根据URL读取指定目录下的文件内容

# 后台项目对的实现步骤

1. 项目准备
2. 总耗时中间件
3. 响应头中间件
4. 业务逻辑中间件
5. 允许跨域

我这里就直接借用gethub里一位大神的后端接口

git clone <https://gitee.com/penggang-home/koa_server.git>

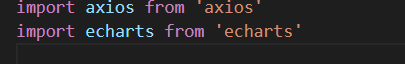
# 前端项目准备

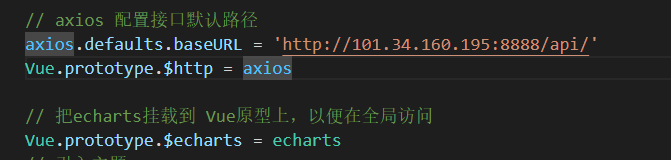
vue create vision创建vue项目

因为主要用到的是echarts，所以第一步现在项目中全局挂载echarts对象

在public目录下的index.html引入echarts然后在min.js将全局的eccharts对象挂载到vue的原型对象上

然后还要使用axios，所以随便吧axios一起挂载



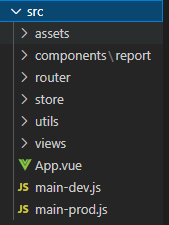


项目准备的表格：

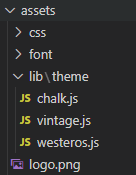
1. 商家销售系统（横向柱状图）
2. 销售趋势图表（折线图）
3. 商家发布模块（地图+散点图）
4. 销售排行模块（柱状图）
5. 热销商品占比模块（饼图）
6. 库存与销量模块（圆环饼图）

# 商家销售系统（横向柱状图）的实现

首先创建文件和目录结构



assets主要存放样式类文件字体图标还有echarts主题文件



components主要存放共用的组件

router路由

store vuex状态管理

utils WebSocket

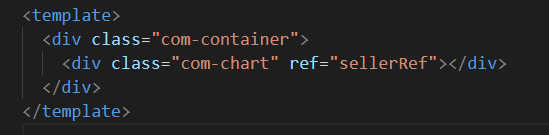
views 主要视图

项目目录搭建完毕后开始封装商家销售系统要使用的组件

在components目录下创建report文件夹然后在report文件夹中创建Seller.vue

以下是Seller.vue的代码

先给一个盒子



然后是要使用的代码

export default {

  // 商家销售统计

  name: 'Seller',

  data() {

    return {

      // echarts 实例对象

      chartInstance: null,

      // 服务器返回的数据

      allData: null,

      // 当前显示的页数

      curretnPage: 1,

      // 总页数

      totalPage: 0,

      // 定时器标识

      timerId: null,

      // 当鼠标移入axis(坐标轴)时展示 底层的背景色

      PointerColor: this.axisPointerColor,

    }

  },

  created() {

    this.$socket.registerCallBack('sellerData', this.getData)

  },

  computed: {

    ...mapState(['theme']),

    axisPointerColor() {

      return getThemeValue(this.theme).sellerAxisPointerColor

    },

  },

  watch: {

    theme() {

      console.log('主题切换了')

      // 销毁当前的图表

      this.chartInstance.dispose()

      // 以最新主题初始化图表对象

      this.initChart()

      // 屏幕适配

      this.screenAdapter()

      // 渲染数据

      this.updateChart()

    },

  },

  mounted() {

    // 由于初始化 使用到了DOM元素，因此需要在 mounted生命周期内调用

    this.initChart()

    // this.getData();

    this.$socket.send({

      action: 'getData',

      socketType: 'sellerData',

      chartName: 'seller',

      value: '',

    })

    // 在界面加载完成时，主动对屏幕进行适配

    this.screenAdapter()

    window.addEventListener('resize', this.screenAdapter)

  },

  // 实例销毁后触发

  destroyed() {

    clearInterval(this.timeID)

    // 在组件销毁的时候，把监听器取消掉

    window.removeEventListener('resize', this.screenAdapter)

    this.$socket.unRegisterCallBack('sellerData')

  },

  methods: {

    // 初始化 echartsInstance 对象

    initChart() {

      this.chartInstance = this.$echarts.init(this.$refs.sellerRef, this.theme)

      // 对图表初始化的配置

      const initOption = {

        title: {

          text: '▎商家销售统计',

          left: 20,

          top: 20,

        },

        grid: {

          top: '20%',

          left: '3%',

          right: '6%',

          bottom: '3%',

          // 默认grid不包含坐标轴文字，改为true

          containLabel: true,

        },

        xAxis: {

          type: 'value',

        },

        yAxis: {

          type: 'category',

        },

        tooltip: {

          // 当鼠标移入axis(坐标轴)时展示 底层的背景色

          trigger: 'axis',

          axisPointer: {

            // 展示的类型是线条类型

            type: 'line',

            lineStyle: {

              color: this.axisPointerColor,

            },

            // 相等于 z-index 将层级调低

            z: 0,

          },

        },

        series: [

          {

            type: 'bar',

            label: {

              show: true,

              position: 'right',

              textStyle: {

                color: 'white',

              },

            },

            // 每一个柱的样式

            itemStyle: {

              // new 出 echarts 全局对象的一个线性渐变方法

              // 指明方向(第四象限坐标轴),不同百分比之下颜色的值

              color: new this.$echarts.graphic.LinearGradient(0, 0, 1, 0, [

                // 0% 状态时的颜色

                { offset: 0, color: '#5052EE' },

                // 100% 的颜色

                { offset: 1, color: '#AB6EE5' },

              ]),

            },

          },

        ],

      }

      this.chartInstance.setOption(initOption)

      // 对图表进行鼠标事件的监听

      this.chartInstance.on('mouseover', () => {

        this.timerId && clearInterval(this.timerId)

      })

      this.chartInstance.on('mouseout', () => {

        this.startInterval()

      })

    },

    // 获取服务器数据

    getData(res) {

      // http://101.34.160.195:8888/api/seller

      // const { data: res } = await this.$http.get('seller')

      // console.log('res: ', res)

      this.allData = res

      // 对数组排序 从小到大进行排序

      this.allData.sort((a, b) => b.value - a.value)

      // 每五个元素显示一页 计算出总页数

      this.totalPage = Math.ceil(this.allData.length / 5)

      // 开始第一次渲染

      this.updateChart()

      // 开启定时器 开始动态渲染

      this.startInterval()

    },

    // 更新图表

    updateChart() {

      // 动态从数组中取出5条数据

      const start = (this.curretnPage - 1) \* 5

      const end = this.curretnPage \* 5

      const showData = this.allData.slice(start, end)

      console.log('showData: ', showData)

      // y轴上的数据

      const sellerNames = showData.map(item => item.name)

      // x 轴上的数据

      const sellerValues = showData.map(item => item.value)

      // 当拿到数据后，准备数据的配置项

      const dataOption = {

        yAxis: {

          data: sellerNames,

        },

        series: [

          {

            data: sellerValues,

          },

        ],

      }

      // 设置数据

      this.chartInstance.setOption(dataOption)

    },

    // 开启动态渲染的定时器

    startInterval() {

      // 一般使用定时器都有一个保险操作,先关闭再开启

      this.timerId && clearInterval(this.timerId)

      this.timerId = setInterval(() => {

        this.curretnPage++

        // 当超出最大页数时,回滚到第一页

        if (this.curretnPage > this.totalPage) this.curretnPage = 1

        this.updateChart()

      }, 3000)

    },

    // 当浏览器窗口大小发生变化，完成屏幕适配

    screenAdapter() {

      const titleFontSize = (this.$refs.sellerRef.offsetWidth / 100) \* 3.6

      // 浏览器分辨率大小相关的配置项

      const adapterOption = {

        title: {

          textStyle: {

            fontSize: titleFontSize,

          },

        },

        tooltip: {

          axisPointer: {

            lineStyle: {

              width: titleFontSize,

            },

          },

        },

        series: [

          {

            barWidth: titleFontSize,

            itemStyle: {

              barBorderRadius: [0, titleFontSize / 2, titleFontSize / 2, 0],

            },

          },

        ],

      }

      this.chartInstance.setOption(adapterOption)

      // 手动调用图表的 resize 才能产生效果

      this.chartInstance.resize()

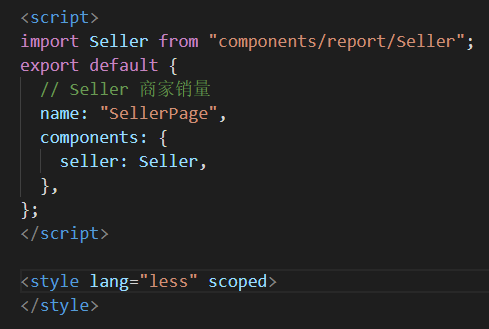
    },

  },

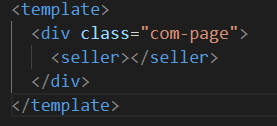
}

然后在views目录下创建SellerPage.vue的视图

引入组件



使用组件

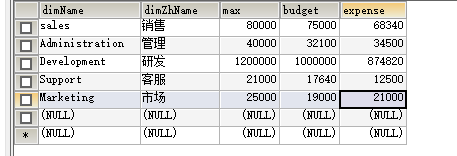


实现效果：



数据库表格：

Budget表



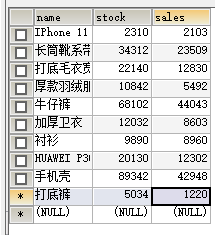
Rank表

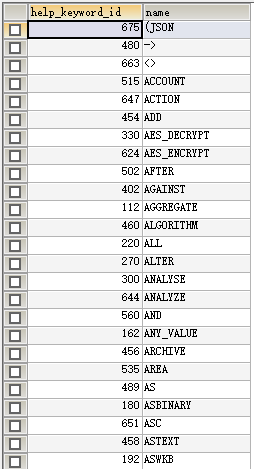


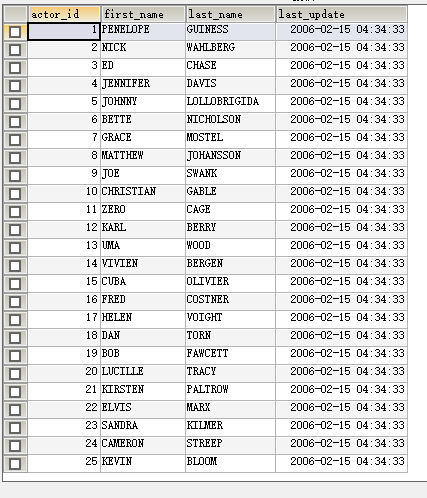
Seller表

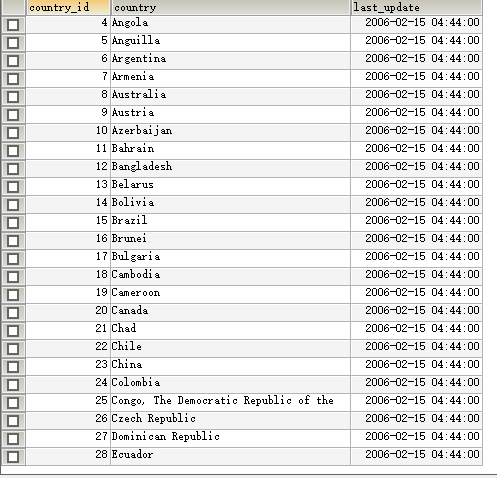


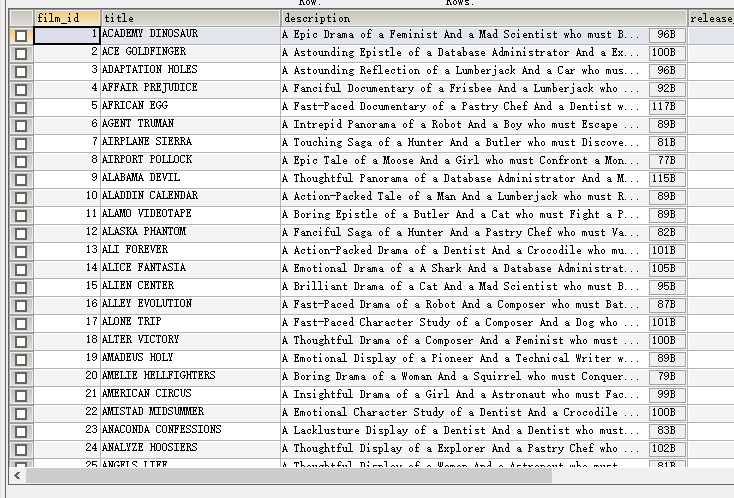
Stock表

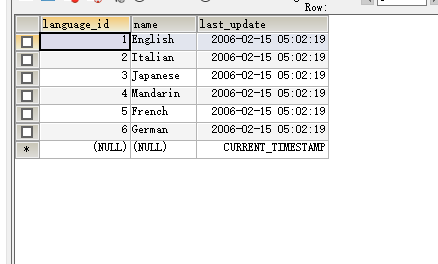


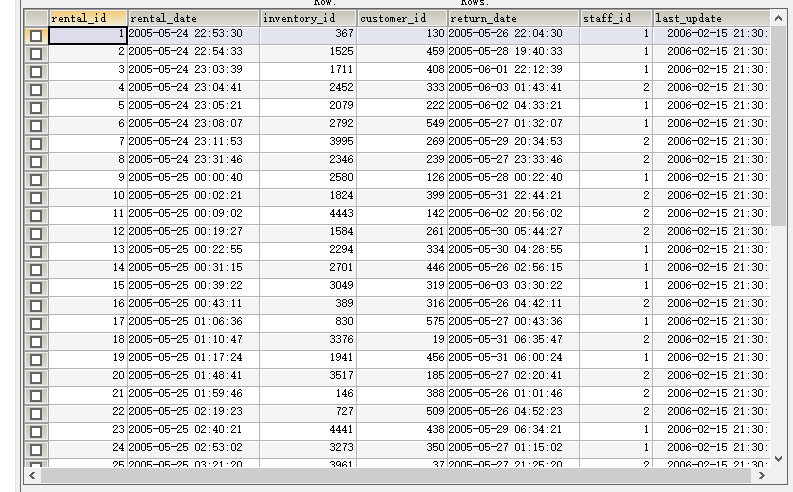












# 感谢观看