项目最终的效果如图所示:

最终效果涉及到6个图表, 5种图表类型,它们分别是折线图,柱状图,地图,散点图,饼图

每个图表的数据都是从后端推送到前端来的, 不过在项目的初期,我们会先使用`ajax`由前端主动获取数据, 后续会使用`WebSocket`进行改造.

整个项目的架构是基于`Vue`的, 所以我们需要创建`Vue`项目, 然后在`Vue`项目中开发各个图表组件.

1.前端项目的准备

1.1.`vue-cli`脚手架创建项目

1.1.1 脚手架环境的安装

- 在全局环境中安装`vue-cli`脚手架

```
npm install -g @vue/cli
```

1.1.2. 工程的创建

- 使用命令行执行

```
vue create vision
```

- 具体的配置项如下:
  - 手动选择特性

  - 集成`Router`,`Vuex`,`CSS Pre-processors`

- 是否选用历史模式的路由

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors, Linter
? Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n) n
```

- 选择`Less`作为`CSS`的预处理器

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors, Linter
? Use history mode for router? (Requires proper server setup for index fallback in production) No
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default):
  Sass/SCSS (with dart-sass)
  Sass/SCSS (with node-sass)
> Less
  Stylus
```

- 选择`ESLint`的配置

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors, Linter
? Use history mode for router? (Requires proper server setup for index fallback in production) No
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Less
? Pick a linter / formatter config:
  ESLint with error prevention only
  ESLint + Airbnb config
> ESLint + Standard config
  ESLint + Prettier
```

- 什么时候进行`Lint`提示

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors, Linter
? Use history mode for router? (Requires proper server setup for index fallback in production) No
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Less
? Pick a linter / formatter config: Standard
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)
>(*) Lint on save
 ( ) Lint and fix on commit
```

- 如何存放`Babel`，`ESLint`等配置文件

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors, Linter
? Use history mode for router? (Requires proper server setup for index fallback in production) No
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Less
? Pick a linter / formatter config: Standard
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json
```

- 是否保存以上配置以便下次创建项目时使用

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors, Linter
? Use history mode for router? (Requires proper server setup for index fallback in production) No
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Less
? Pick a linter / formatter config: Standard
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) n
```

- 配置选择完之后，就开始创建项目了，这个过程需要一些时间：

```
✨  Creating project in C:\Users\itheima\vision.
🗃  Initializing git repository...
⚙  Installing CLI plugins. This might take a while...

[                    ] \ fetchMetadata: sill pacote version manifest for @webassemblyjs/ast@1.9.0 fetched in 53ms
```

- 当项目就创建完成了，会看到这个提示

```
$ cd vision
$ npm run serve

 WARN  Skipped git commit due to missing username and email in git config.
You will need to perform the initial commit yourself.
```

- 运行默认的项目

```
cd vision
npm run serve
```

- 将目录使用`vscode`打开

1.1.3.删除无关代码

- 修改 `App.vue` 中的代码,将布局和样式删除, 变成如下代码:

```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>

<style lang="less">
</style>
```

- 删除 `components/HelloWorld.vue` 这个文件
- 删除 `views/About.vue` 和 `views/Home.vue` 这两个文件
- 修改 `router/index.js` 中的代码,去除路由配置和 `Home` 组件导入的代码

```
import Vue from 'vue'
import VueRouter from 'vue-router'

Vue.use(VueRouter)

const routes = []

const router = new VueRouter({
  routes
})

export default router
```

1.2. 项目的基本配置

- 在项目根目录下创建 `vue.config.js` 文件
- 在文件中增加代码:

```
// 使用vue-cli创建出来的vue工程, webpack的配置是被隐藏起来了的
// 如果想覆盖webpack中的默认配置,需要在项目的根路径下增加vue.config.js文件
module.exports = {
  devServer: {
    port: 8999, // 端口号的配置
    open: true // 自动打开浏览器
  }
}
```

1.3.全局 `echarts` 对象

1.3.1.引入 `echarts` 包

- 将资料文件夹中的 `static` 目录复制到 `public` 目录之下
- 在 `public/index.html` 文件中引入 `echarts.min.js` 文件

```
<body>
  <noscript>
    <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScr
  </noscript>
  <div id="app"></div>
  <!-- built files will be auto injected -->
  <script src="static/lib/echarts.min.js"></script>
</body>
```

1.3.2.挂载到Vue原型上

- 在src/main.js文件中挂载

  由于在index.html中已经通过script标签引入了echarts.js文件夹, 故在window全局对象中是存在echarts全局对象, 将其挂载到Vue的原型对象上

```
......
// 将全局echarts对象挂载到Vue的原型对象上
Vue.prototype.$echarts = window.echarts
......
```

1.3.3.使用全局echarts对象

- 在其他组件中使用

```
this.$echarts
```

1.4.axios的处理

1.4.1.安装axios包

```
npm install axios
```

1.4.2.封装axios对象

- 在src/main.js文件中配置axios并且挂载到Vue的原型对象上

```
......
import axios from 'axios'
axios.defaults.baseURL = 'http://127.0.0.1:8888/api/'
// 将axios挂载到Vue的原型对象上
Vue.prototype.$http = axios
......
```

1.4.3.使用axios对象

- 在其他组件中使用

```
this.$http
```

2.单独图表组件的开发

在项目的初期, 我们会每个图表单独的进行开发, 最后再将所有的图表合并到一个界面中.

在单独开发每个图表的时候, 一个图表会用一个单独的路径进行全屏展示, 他们分别是:

- 商家销售统计

`http://127.0.0.1:8999/#/sellerpage`

- 销量趋势分析

  `http://127.0.0.1:8999/#/trendpage`

- 商家地图分布

  `http://127.0.0.1:8999/#/mappage`

- 地区销量排行

  `http://127.0.0.1:8999/#/rankpage`

- 热销商品占比

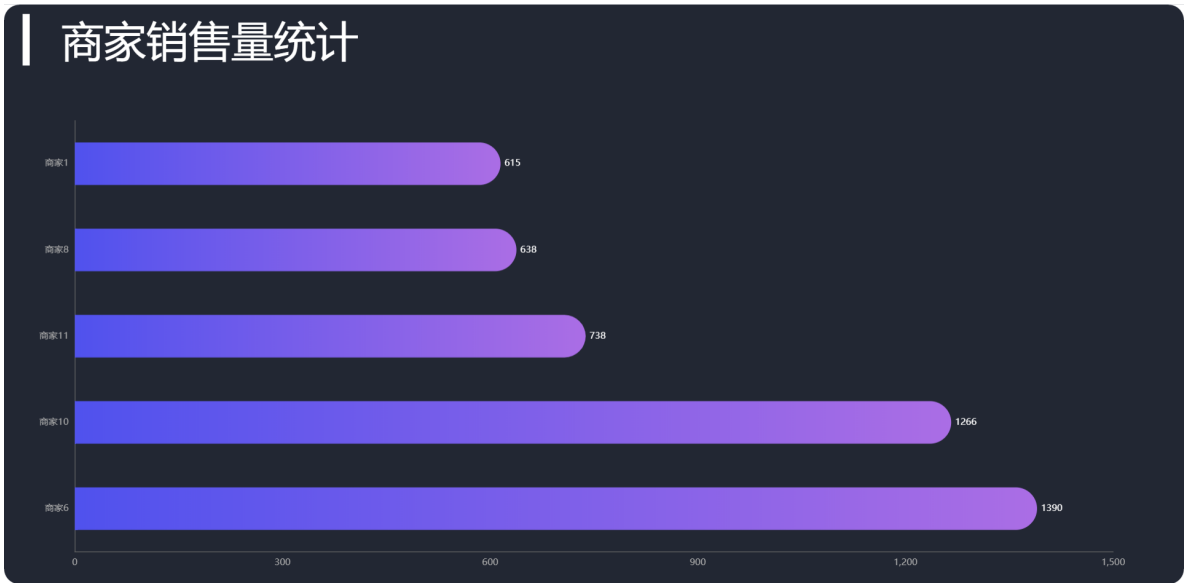
  `http://127.0.0.1:8999/#/hotpage`

- 库存销量分析

  `http://127.0.0.1:8999/#/stockpage`

2.1.商家销量排行

最终的效果如下图所示:



2.1.1.组件结构设计

- 在`src/components/`目录下建立`Seller.vue`,这个组件是真实展示图表的组件
  - 给外层div增加类样式`com-container`
  - 建立一个显示图表的div元素
  - 给新增的这个div增加类样式`com-chart`

```
<!-- 显示商家销量统计的图表 -->
<template>
  <div class="com-container">
    Seller组件
    <div class="com-chart"></div>
  </div>
</template>

<script>
export default {
  data () {
    return {}
```

```
  },
  methods: {}
}
</script>

<style lang="less" scoped>
</style>
```

- 在`src/views/`目录下建立`SellerPage.vue`,这个组件是对应于路由 /seller 而展示的
  - 给外层div元素增加样式`com-page`
  - 在`SellerPage`中引入`Seller`组件,并且注册和使用

```
<!-- 这个组件是对应于路由规则中 /seller 这条路径的
     在这个组件中,需要展示Seller.vue这个组件
     Seller.vue才是真正显示图表的组件
 -->
<template>
  <div class="com-page">
    <seller></seller>
  </div>
</template>

<script>
import Seller from '@/components/Seller'
export default {
  data () {
    return {}
  },
  methods: {},
  components: {
    seller:Seller
  }
}
</script>

<style lang='less' scoped>
</style>
```

- 增加路由规则, 在`src/router/index.js`文件中修改

```
......
import SellerPage from '@/views/SellerPage'
......
const routes = [
  {
    path: '/sellerpage',
    component: SellerPage
  }
]
```

- 新建`src/assets/css/global.less`增加宽高样式

  原则就是将所有的容器的宽度和高度设置为占满父容器

```
html,
body,
#app {
  width: 100%;
  height: 100%;
  padding: 0;
  margin: 0;
  overflow: hidden;
}
.com-page {
  width: 100%;
  height: 100%;
  overflow: hidden;
}
.com-container {
  width: 100%;
  height: 100%;
  overflow: hidden;
}
.com-chart {
  width: 100%;
  height: 100%;
  overflow: hidden;
}
```

在`main.js`中引入样式

```
import './assets/css/global.less'
```

- 打开浏览器, 输入`http://127.0.0.1:8999/#/sellerpage`看Seller组件是否能够显示

2.1.2.图表`Seller.vue`基本功能的实现

- 1.在mounted生命周期中初始化`echartsInstance`对象
- 2.在mounted中获取服务器的数据
- 3.将获取到的数据设置到图表上

```
<script>
export default {
  data () {
    return {
      chartInstance: null, // echarts实例对象
      allData: [] // 服务器获取的所有数据
    }
  },
  mounted () {
    // 由于初始化echarts实例对象需要使用到dom元素,因此必须要放到mounted中, 而不是created
    this.initChart()
    this.getData()
  },
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.seller_ref) // 初始化
echarts实例对象
    },
```

```
    async getData () {
      const { data: res } = await this.$http.get('seller') // 获取数据
      this.allData = res
      // 对allData进行从大到小的排序
      this.allData.sort((a, b) => {
        return a.value - b.value
      })
      this.updateChart()
    },
    updateChart () {
      // 处理数据并且更新界面图表
      const sellerNames = this.allData.map((item) => {
        return item.name
      })
      const sellerValues = this.allData.map((item) => {
        return item.value
      })
      const option = {
        xAxis: {
          type: 'value'
        },
        yAxis: {
          type: 'category',
          data: sellerNames
        },
        series: [
          {
            type: 'bar',
            data: sellerValues
          }
        ]
      }
      this.chartInstance.setOption(option)
    }
  }
}
</script>
```

- 4.拆分配置项`option`
  - 初始化配置项

```
initChart () {
  this.chartInstance = this.$echarts.init(this.$refs.seller_ref) // 初始化echarts实例对象
  // 初始化的图表配置项, 和数据无关
  const initOption = {
    xAxis: {
      type: 'value'
    },
    yAxis: {
      type: 'category'
    },
    series: [
      {
        type: 'bar'
      }
    ]
  }
  this.chartInstance.setOption(initOption)
},
```

  - 拥有数据之后的配置项

```
updateChart () {
  // 处理数据并且更新界面图表
  const sellerNames = this.allData.map((item) => {
    return item.name
  })
  const sellerValues = this.allData.map((item) => {
    return item.value
  })
  // 拥有数据之后的配置项
  const dataOption = {
    yAxis: {
      data: sellerNames
    },
    series: [
      {
        data: sellerValues
      }
    ]
  }
  this.chartInstance.setOption(dataOption)
}
```

2.1.3.分页动画的实现

- 数据的处理, 每5个元素显示一页
  - 数据的处理

```
data () {
  return {
    chartInstance: null, // echarts实例对象
    allData: [], // 服务器获取的所有数据
    currentPage: 1, // 当前页数
    totalPage: 0, // 总页数,需要进行计算,每页显示5个条目
  }
},
```

```
async getData () {
  const { data: res } = await this.$http.get('seller') // 获取数据
  this.allData = res
  // 对allData进行从大到小的排序
  this.allData.sort((a, b) => {
    return a.value - b.value
  })
  // 计算总页数
  this.totalPage = this.allData.length % 5 === 0 ? parseInt(this.allData.length / 5) : parseInt
  (this.allData.length / 5) + 1
  this.updateChart()
},
```

```
updateChart () {
  const start = (this.currentPage - 1) * 5
  const end = this.currentPage * 5
  const showData = this.allData.slice(start, end)
  // 处理数据并且更新界面图表
  const sellerNames = showData.map((item) => {
    return item.name
  })
  const sellerValues = showData.map((item) => {
    return item.value
  })
```

  - 动画的启动和停止

```
data () {
  return {
    chartInstance: null, // echarts实例对象
    allData: [], // 服务器获取的所有数据
    currentPage: 1, // 当前页数
    totalPage: 0, // 总页数,需要进行计算,每页显示5个条目
    timerId: null // 定时器标识
  }
},
```

```
async getData () {
  const { data: res } = await this.$http.get('seller') // 获取数据
  this.allData = res
  // 对allData进行从大到小的排序
  this.allData.sort((a, b) => {
    return a.value - b.value
  })
  // 计算总页数
  this.totalPage = this.allData.length % 5 === 0 ? parseInt(this.allData.length / 5) : parseInt
  (this.allData.length / 5) + 1
  this.updateChart()
  this.startInterval() // 开启动画效果
},
```

```
startInterval () {
  if (this.timerId) {
    clearInterval(this.timerId)
  }
  this.timerId = setInterval(() => {
    this.currentPage++
    if (this.currentPage > this.totalPage) {
      this.currentPage = 1
    }
    this.updateChart()
  }, 3000)
}
},
```

- 鼠标事件的处理

```
initChart () {
  this.chartInstance = this.$echarts.init(this.$refs.seller_ref) // 初始化echarts实例对象
  this.chartInstance.on('mouseover', () => {
    clearInterval(this.timerId)
  })
  this.chartInstance.on('mouseout', () => {
    this.startInterval()
  })
},
```

- 细节的处理
  - 固定x轴最大值

```
const option = {
  xAxis: {
    type: 'value',
    max: this.allData[this.allData.length - 1].value
  },
```

2.1.4. UI效果调整

- 主题文件的导入

  public/index.html中引入

```
    <div id="app"></div>
    <!-- built files will be auto injected -->
    <!-- 引入echarts.js文件
         引入该文件后，该文件就会在window对象中增加一个属性echarts
         所以此时使用全局echarts对象的方式是:window.echarts
    -->
    <script src="static/lib/echarts.min.js"></script>
    <!-- 引入主题文件 -->
    <script src="static/theme/chalk.js"></script>
    <script src="static/theme/vintage.js"></script>
  </body>
</html>
```

- 主题的指定,在初始化echarts实例对象的时候指定

  src/components/Seller.vue

```
initChart () {
  this.chartInstance = this.$echarts.init(this.$refs.seller_ref, 'chalk') // 初始化echarts实例对象
  this.chartInstance.on('mouseover', () => {
    clearInterval(this.timerId)
  })
  this.chartInstance.on('mouseout', () => {
    this.startInterval()
  })
},
```

- 边框圆角的设置

  `src/assets/css/global.less`

```
canvas {
  border-radius: 20px;
}
```

- 其他图表样式的配置
  - 标题的位置和颜色

```
const initOption = {
    title: {
        text: '▌ 商家销量排行',
        left: 20,
        top: 20,
        textStyle: {
            textStyle: {
                "color": "#fff"
            }
        }
    },
```

  - 坐标轴的大小

```
const initOption = {
    ......
    grid: {
        top: '20%',
        left: '3%',
        right: '6%',
        bottom: '3%',
        containLabel: true
    },
```

  - 工具提示和背景

```
const initOption = {
    ......
    tooltip: {
        trigger: 'axis',
        axisPointer: {
            type: 'line',
            z: 0,
            lineStyle: {
                width: 66,
                color: '#2D3443'
            }
        }
    },
```

  - 文字显示和位置

```
 const initOption = {
    ......
    series: [
         {
            ......
            label: {
                show: true,
                position: 'right',
                textStyle: {
                    color: '#fff'
                }
           },
```

- 柱宽度和柱圆角的实现

```
const initOption = {
    ......
    series: [
        {
            ......
            barWidth: 66,
            itemStyle: {
                barBorderRadius: [0, 33, 33, 0]
            }
        }
    ]
}
```

- 柱颜色渐变的实现

  线性渐变可以通过`LinearGradient`进行实现

  `LinearGradient`需要传递5个参数, 前四个代表两个点的相对位置,第五个参数代表颜色变化的范围

  `0, 0, 1, 0`代表的是从左往右的方向

```
const initOption = {
    series: [
          {
            ......
            itemStyle: {
              barBorderRadius: [0, 33, 33, 0],
              color: new this.$echarts.graphic.LinearGradient(0, 0, 1,
0, [
                {
                  offset: 0,
                  color: '#5052EE'
                },
                {
                  offset: 1,
                  color: '#AB6EE5'
                }
              ])
            }
          }
        ]
```

```
        }
```

2.1.5.分辨率适配

- 对窗口大小变化的事件进行监听

  `mounted`时候监听

  ```
  mounted () {
      this.initChart()
      this.getData()
      window.addEventListener('resize', this.screenAdapter)
  }
  ```

  组件销毁时取消监听

  ```
  destroyed () {
      clearInterval(this.timerId)
      // 在组件销毁的时候，需要将监听器取消掉
      window.removeEventListener('resize', this.screenAdapter)
  },
  ```

- 获取图表容器的宽度计算字体大小

  ```
  // 当浏览器的大小发生变化的时候，会调用的方法，来完成屏幕的适配
  screenAdapter () {
      // console.log(this.$refs.seller_ref.offsetWidth)
      const titleFontSize = this.$refs.seller_ref.offsetWidth / 100 * 3.6
  ```

- 将字体大小的值设置给图表的某些区域
  - 标题大小
  - 背景大小
  - 柱宽度
  - 圆角大小

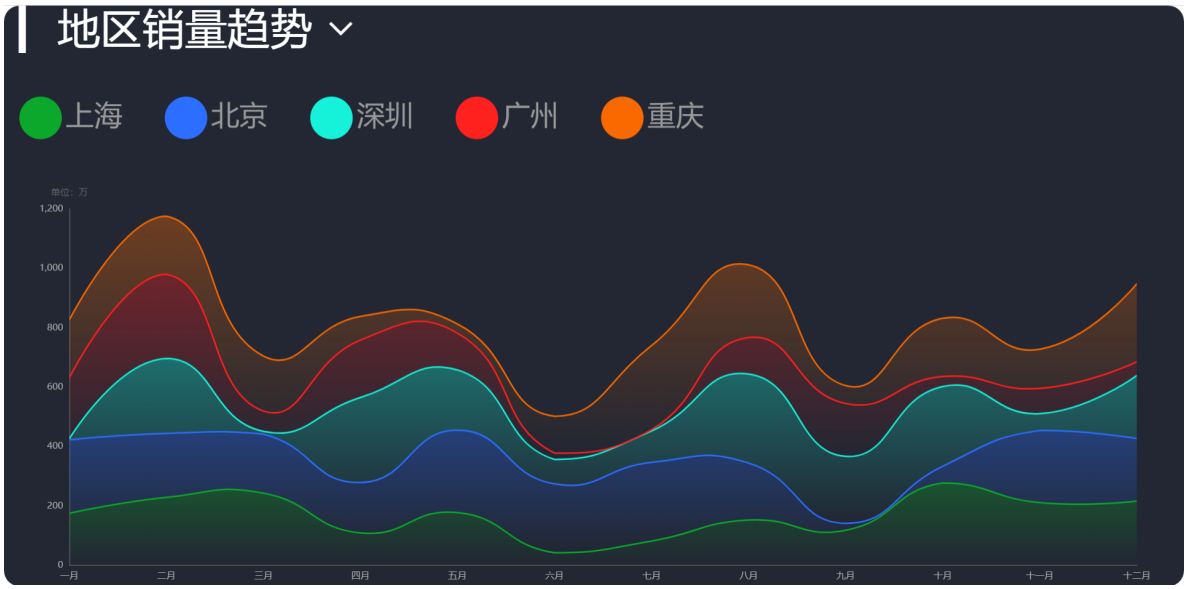
  ```
  // 当浏览器的大小发生变化的时候，会调用的方法，来完成屏幕的适配
  screenAdapter () {
      // console.log(this.$refs.seller_ref.offsetWidth)
      const titleFontSize = this.$refs.seller_ref.offsetWidth / 100 * 3.6
      // 和分辨率大小相关的配置项
      const adapterOption = {
          title: {
              textStyle: {
                  fontSize: titleFontSize
              }
          },
          tooltip: {
              axisPointer: {
                  lineStyle: {
                      width: titleFontSize
                  }
              }
          },
          series: [
  ```

```
            {
                barWidth: titleFontSize,
                itemStyle: {
                    barBorderRadius: [0, titleFontSize / 2,
titleFontSize / 2, 0]
                }
            }
        ]
    }
    this.chartInstance.setOption(adapterOption)
    // 手动的调用图表对象的resize 才能产生效果
    this.chartInstance.resize()
}
```

- 删除`initOption`中 柱宽度\背景大小\圆角大小的配置

2.2.销量趋势分析

最终的效果如下:



2.2.1.代码环境的准备

`TrendPage.vue`

```
<!--
针对于 /trendpage 这条路径而显示出来的
在这个组件中，通过子组件注册的方式，要显示出Trend.vue这个组件
-->
<template>
  <div class="com-page">
    <trend></trend>
  </div>
</template>

<script>
import Trend from '@/components/Trend'
export default {
  data () {
    return {}
  },
```

```
  methods: {},
  components: {
    trend: Trend
  }
}
</script>

<style lang="less" scoped>
</style>
```

`Trend.vue`

```
<!-- 销量趋势图表 -->
<template>
  <div class='com-container'>
    <div class='com-chart' ref='trend_ref'></div>
  </div>
</template>

<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null
    }
  },
  mounted () {
    this.initChart()
    this.getData()
    window.addEventListener('resize', this.screenAdapter)
    this.screenAdapter()
  },
  destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
  },
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.trend_ref)
      const initOption = {}
      this.chartInstance.setOption(initOption)
    },
    async getData () {
      // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
      this.updateChart()
    },
    updateChart () {
      // 处理图表需要的数据
      const dataOption = {}
      this.chartInstance.setOption(dataOption)
    },
    screenAdapter () {
      const adapterOption = {}
      this.chartInstance.setOption(adapterOption)
      this.chartInstance.resize()
    }
```

```
  }
}
</script>

<style lang='less' scoped>
</style>
```

`router/index.js`

```
......
import TrendPage from '@/views/TrendPage'
......
const routes = [
  ......
  {
    path: '/trendpage',
    component: TrendPage
  }
]
......
```

2.2.2.图表基本功能的实现

- 数据的获取

```
async getData () {
    // 获取服务器的数据，对this.allData进行赋值之后，调用updateChart方法更新图表
    const { data: ret } = await this.$http.get('trend')
    this.allData = ret
    this.updateChart()
}
```

- 数据的处理

```
updateChart () {
    // x轴的数据
    const timeArrs = this.allData.common.month
    // y轴的数据，暂时先取出map这个节点的数据
    // map代表地区销量趋势
    // seller代表商家销量趋势
    // commodity代表商品销量趋势
    const valueArrs = this.allData.map.data
    // 图表数据，一个图表中显示5条折线图
    const seriesArr = valueArrs.map((item, index) => {
        return {
            type: 'line', // 折线图
            name: item.name,
            data: item.data,
        }
    })
    const dataOption = {
        xAxis: {
            data: timeArrs
        },
```

```
        legend: {
            data: legendArr
        },
        series: seriesArr
    }
    this.chartInstance.setOption(dataOption)
}
```

- 初始化配置

```
const initOption = {
    xAxis: {
        type: 'category',
        boundaryGap: false
    },
    yAxis: {
        type: 'value'
    }
}
```

- 堆叠图效果

  要实现堆叠图的效果, series下的每个对象都需要配置上相同的stack属性

```
updateChart () {
    const timeArrs = this.allData.common.month
    const valueArrs = this.allData.map.data
    const seriesArr = valueArrs.map((item, index) => {
        return {
            type: 'line',
            name: item.name,
            data: item.data,
            stack: 'map' // stack值相同, 可以形成堆叠图效果
        }
    })
    ......
}
```

- 图例效果

```
updateChart () {
    ......
    const valueArrs = this.allData.map.data
    const seriesArr = valueArrs.map((item, index) => {
        return {
          type: 'line',
          name: item.name,
          data: item.data,
          stack: 'map'
        }
      })
    // 准备图例数据, 它需要和series下的每个对象的name属性保持一致
    const legendArr = valueArrs.map(item => {
        return item.name
    })
    const dataOption = {
```

```
        ......
        legend: {
            data: legendArr
        }
        ......
    }
    this.chartInstance.setOption(dataOption)
}
```

2.2.3. UI效果的调整

- 主题的使用

```
initChart () {
    this.chartInstance = this.$echarts.init(this.$refs.trend_ref, 'chalk')
}
```

主题使用完之后, 发现折线图都变成了平滑折线图了, 这是因为在`chalk.js`主题文件中, 设置了`smooth:true`

- 坐标轴大小和位置

```
const initOption = {
    grid: {
        top: '35%',
        left: '3%',
        right: '4%',
        bottom: '1%',
        containLabel: true
    },
    tooltip: {
        trigger: 'axis'
    }
}
```

- 工具提示

```
const initOption = {
    ......
    tooltip: {
        trigger: 'axis'
    }
    ......
}
```

- 图例位置和形状

```
const initOption = {
    ......
    legend: {
        top: '15%',
        icon: 'circle',
        left: 20
    },
    ......
}
```

- 区域面积的设置

  区域面积只需要给series的每一个对象增加一个`areaStyle`即可

- 颜色渐变的设置

  颜色渐变可以通过`LinearGradient`进行设置, 颜色渐变的方向从上往下

```
updateChart () {
    // 半透明的颜色值
    const colorArr1 = [
        'rgba(11, 168, 44, 0.5)',
        'rgba(44, 110, 255, 0.5)',
        'rgba(22, 242, 217, 0.5)',
        'rgba(254, 33, 30, 0.5)',
        'rgba(250, 105, 0, 0.5)'
    ]
    // 全透明的颜色值
    const colorArr2 = [
        'rgba(11, 168, 44, 0)',
        'rgba(44, 110, 255, 0)',
        'rgba(22, 242, 217, 0)',
        'rgba(254, 33, 30, 0)',
        'rgba(250, 105, 0, 0)'
    ]
    ......
    const seriesArr = valueArrs.map((item, index) => {
        return {
            ......
            areaStyle: {
                color: new this.$echarts.graphic.LinearGradient(0, 0, 0, 1,
[
                    {
                        offset: 0,
                        color: colorArr1[index]
                    },
                    {
                        offset: 1,
                        color: colorArr2[index]
                    }
                ])
            },
            stack: 'map'
        }
    })
    ......
```

2.2.4.切换图表

- 布局的实现

  增加类样式为`title`的容器

```
<template>
  <div class='com-container'>
    <div class="title">
      <span>我是标题</span>
      <span class="iconfont title-icon">&#xe6eb;</span>
      <div class="select-con">
        <div class="select-item">
          标题选择1
        </div>
        <div class="select-item">
          标题选择2
        </div>
        <div class="select-item">
          标题选择3
        </div>
      </div>
    </div>
    <div class='com-chart' ref='trend_ref'></div>
  </div>
</template>
```

  字体文件的引入

  将资料文件夹下的字体文件夹中的`font`复制到`asset`目录下, 然后在`main.js`中引入字体样式文件

```
// 引入字体文件
import './assets/font/iconfont.css'
```

  在`Trend.vue`中的style标签中增加一些样式

```
<style lang='less' scoped>
.title {
  position: absolute;
  left: 20px;
  top: 20px;
  z-index: 10;
  color: white;
  .title-icon {
    margin-left: 10px;
    cursor: pointer;
  }
  .select-item {
    cursor: pointer;
  }
}
</style>
```

- 数据动态渲染

```
<!-- 销量趋势图表 -->
<template>
  <div class='com-container'>
    <div class="title">
      <span>{{ title }}</span>
      <span class="iconfont title-icon">&#xe6eb;</span>
      <div class="select-con">
        <div class="select-item" v-for="item in selectTypes"
:key="item.key">
          {{ item.text }}
        </div>
      </div>
    </div>
    <div class='com-chart' ref='trend_ref'></div>
  </div>
</template>
```

使用计算属性`title`控制标题的内容和标题的可选择项

```
<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null,
      dataType: 'map' // 这项数据代表目前选择的数据类型, 可选值有map seller
commodity
    }
  },
  computed: {
    selectTypes () {
      if (!this.allData || ! this.allData.type) {
        return []
      } else {
        return this.allData.type.filter(item => {
          return item.key !== this.dataType
        })
      }
    },
    title () {
      if (!this.allData) {
        return ''
      } else {
        return this.allData[this.dataType].title
      }
    }
  },
 ......
```

- 点击三角控制显示隐藏

  增加一项变量控制可选容器的显示与隐藏

```
export default {
  data () {
    return {
      showChoice: false // 控制可选面板的显示或者隐藏
    }
  },
```

使用指令`v-if`和点击事件的监听

```
<template>
  <div class='com-container'>
    <div class="title">
      <span>{{ title }}</span>
      <span class="iconfont title-icon" @click="showChoice =
!showChoice">&#xe6eb;</span>
      <div class="select-con" v-if="showChoice">
        <div class="select-item" v-for="item in selectTypes"
:key="item.key">
          {{ item.text }}
        </div>
      </div>
    </div>
    <div class='com-chart' ref='trend_ref'></div>
  </div>
</template>
```

- 点击可选条目的控制

```
<template>
  <div class='com-container'>
    <div class="title">
      <span>{{ title }}</span>
      <span class="iconfont title-icon" @click="showChoice =
!showChoice">&#xe6eb;</span>
      <div class="select-con" v-if="showChoice">
        <div class="select-item" v-for="item in selectTypes" :key="item.key"
@click="handleSelect(item.key)">
          {{ item.text }}
        </div>
      </div>
    </div>
    <div class='com-chart' ref='trend_ref'></div>
  </div>
</template>
<script>
export default {
  ......
  methods: {
    handleSelect (key) {
      this.dataType = key
      this.updateChart()
      this.showChoice = false
    }
  }
}
</script>
```

将`updateChart`中, 之前写死的map变成`dataType`

```
const valueArrs = this.allData[this.dataType].data
const seriesArr = valueArrs.map((item, index) => {
    return {
        ......
        stack: this.dataType
    }
})
```

2.2.5.分辨率适配

分辨率适配主要就是在`screenAdapter`方法中进行, 需要获取图表容器的宽度,计算出标题字体大小,

将字体的大小赋值给`titleFontSize`

```
<script>
export default {
  data () {
    return {
      titleFontSize: 0
    }
  },
......
screenAdapter () {
    this.titleFontSize = this.$refs.trend_ref.offsetWidth / 100 * 3.6
}
```

通过`titleFontSize`从而设置给标题文字的大小和图例的大小

- 标题文字的大小

  增加计算属性`comStyle`并设置给对应的`div`,如下:

```
<!-- 销量趋势图表 -->
<template>
  <div class='com-container'>
    <div class="title" :style="comStyle">
      <span>{{ title }}</span>
      <span class="iconfont title-icon" @click="showChoice = !showChoice"
:style="comStyle">&#xe6eb;</span>
......
<script>
export default {
  ......
  computed: {
    ......
    comStyle () {
      return {
        fontSize: this.titleFontSize + 'px'
      }
    }
  },
```

- 图例的大小

```
screenAdapter () {
    this.titleFontSize = this.$refs.trend_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        legend: {
            itemWidth: this.titleFontSize,
            itemHeight: this.titleFontSize,
            itemGap: this.titleFontSize,
            textStyle: {
                fontSize: this.titleFontSize / 2
            }
        }
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
},
```

2.2.6.细节调整

- 可选条目的背景色

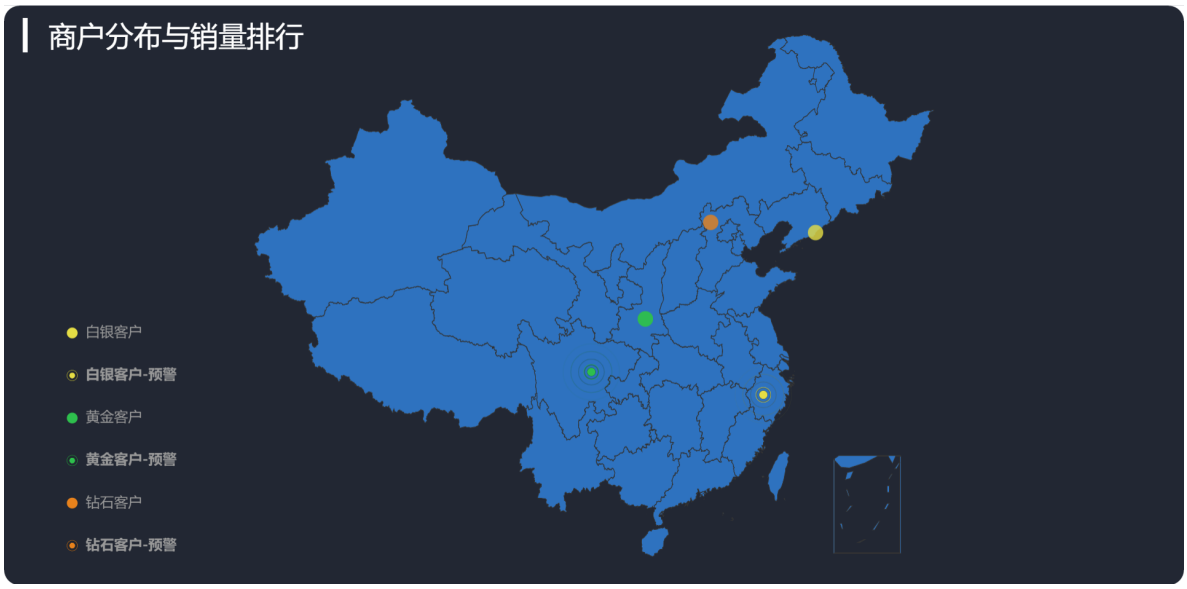
```
<style lang='less' scoped>
.title {
  ......
  .select-con {
    background-color: #222733;
  }
  .select-item {
    cursor: pointer;
  }
}
</style>
```

- 增加标题左侧的小竖杆

```
<template>
  <div class='com-container'>
    <div class="title" :style="comStyle">
      <span>{{'▌ ' + title }}</span>
      <span class="iconfont title-icon" @click="showChoice = !showChoice"
:style="comStyle">&#xe6eb;</span>
      <div class="select-con" v-if="showChoice" :style="marginStyle">
......
<script>
export default {
  ......
  computed: {
    marginStyle () {
      return {
        marginLeft: this.titleFontSize + 'px'
      }
    }
  },
```

2.3.商家地图分布

最终的效果如下:



2.3.1.代码环境的准备

`MapPage.vue`

```
<!--
针对于 /mappage 这条路径而显示出来的
在这个组件中，通过子组件注册的方式，要显示出Map.vue这个组件
-->
<template>
  <div class="com-page">
    <single-map></single-map>
  </div>
</template>

<script>
import Map from '@/components/Map'
export default {
  data () {
    return {}
  },
  methods: {},
  components: {
    'single-map': Map
  }
}
</script>

<style lang="less" scoped>
</style>
```

`Map.vue`

```
<!-- 商家分布图表 -->
<template>
  <div class='com-container'>
    <div class='com-chart' ref='map_ref'></div>
  </div>
```

```
</template>

<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null
    }
  },
  mounted () {
    this.initChart()
    this.getData()
    window.addEventListener('resize', this.screenAdapter)
    this.screenAdapter()
  },
  destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
  },
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.map_ref)
      const initOption = {}
      this.chartInstance.setOption(initOption)
    },
    async getData () {
      // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
      this.updateChart()
    },
    updateChart () {
      // 处理图表需要的数据
      const dataOption = {}
      this.chartInstance.setOption(dataOption)
    },
    screenAdapter () {
      const adapterOption = {}
      this.chartInstance.setOption(adapterOption)
      this.chartInstance.resize()
    }
  }
}
</script>

<style lang='less' scoped>
</style>
```

`router/index.js`

```
......
import MapPage from '@/views/MapPage'
......
const routes = [
  ......
  {
    path: '/mappage',
    component: MapPage
  }
]
......
```

2.3.2.显示地图

- 获取中国地图矢量数据
- 注册地图数据到`全局echarts对象`中
- 配置`geo`

```
<script>
// 获取的是Vue环境之下的数据，而不是我们后台的数据
import axios from 'axios'
export default {
  ......
  methods: {
    async initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.map_ref)
      const { data: mapData } = await
axios.get('http://127.0.0.1:8999/static/map/china.json')
      this.$echarts.registerMap('china', mapData)
      const initOption = {
        geo: {
          type: 'map',
          map: 'china'
        }
      }
      this.chartInstance.setOption(initOption)
    },
```

2.3.3.显示散点图

- 获取散点数据

```
async getScatterData () {
    // 获取服务器的数据，对this.allData进行赋值之后，调用updateChart方法更新图表
    const { data: ret} = await this.$http.get('map')
    this.allData = ret
    this.updateChart()
}
```

- 处理数据并且更新图表

```
updateChart () {
    // 处理图表需要的数据
```

```
    // 图例数据
    const legendData = this.allData.map(item => {
        return item.name
    })
    // 散点数据
    const seriesArr = this.allData.map(item => {
        return {
            type: 'effectScatter',
            coordinateSystem: 'geo',
            name: item.name,
            data: item.children
        }
    })
    const dataOption = {
        legend: {
            data: legendData
        },
        series: seriesArr
    }
    this.chartInstance.setOption(dataOption)
},
```

2.3.4. UI 效果的调整

- 主题的使用

```
methods: {
    async initChart () {
        this.chartInstance = this.$echarts.init(this.$refs.map_ref, 'chalk')
```

- 标题显示

```
const initOption = {
    title: {
        text: '▌ 商家分布',
        left: 20,
        top: 20
    },
```

- 地图位置和颜色

```
const initOption = {
    ......
    geo: {
        type: 'map',
        map: 'china',
        top: '5%',
        bottom: '5%',
        itemStyle: {
            areaColor: '#2E72BF',
            borderColor: '#333'
        }
    }
}
```

- 图例控制

```
const initOption = {
    ......
    legend: {
        left: '5%',
        bottom: '5%',
        orient: 'vertical'
    }
}
```

- 涟漪效果

```
updateChart () {
    ......
    const seriesArr = this.allData.map(item => {
        return {
            type: 'effectScatter',
            rippleEffect: {
                scale: 5,
                brushType: 'stroke'
            },
            ......
        }
    })
```

2.3.5.分辨率适配

- 计算`titleFontSize`

```
screenAdapter () {
    const titleFontSize = this.$refs.map_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
}
```

- 将`titleFontSize`设置给图表的某些区域
  - 标题的大小
  - 图例大小

```
screenAdapter () {
    const titleFontSize = this.$refs.map_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        title: {
            textStyle: {
                fontSize: titleFontSize
            }
        },
        legend: {
            itemWidth: titleFontSize / 2,
            itemHeight: titleFontSize / 2,
            itemGap: titleFontSize / 2,
            textStyle: {
```

```
                fontSize: titleFontSize / 2
            }
        }
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
}
```

2.3.6.地图点击事件

- 响应图表的点击事件, 并获取点击项相关的数据

```
async initChart () {
    ......
    this.chartInstance.on('click', arg => {
        // arg.name 就是所点击的省份名称, 是中文
    })
```

- 将资料中的`map_utils.js`复制到`src/utils/`目录之下
- 得到地图所点击项的拼音和地图矢量数据的路径

```
<script>
// 获取的是Vue环境之下的数据, 而不是我们后台的数据
import axios from 'axios'
import { getProvinceMapInfo } from '@/utils/map_utils'
export default {
  ......
  methods: {
    async initChart () {
      ......
      this.chartInstance.setOption(initOption)
      this.chartInstance.on('click', async arg => {
        // arg.name 就是所点击的省份名称, 是中文
        const provinceInfo = getProvinceMapInfo(arg.name)
        const { data: ret } = await axios.get('http://127.0.0.1:8999' +
provinceInfo.path)
        this.$echarts.registerMap(provinceInfo.key, ret)
        this.chartInstance.setOption({
          geo: {
            map: provinceInfo.key
          }
        })
      })
      this.getScatterData()
    }
  }
}
</script>
```

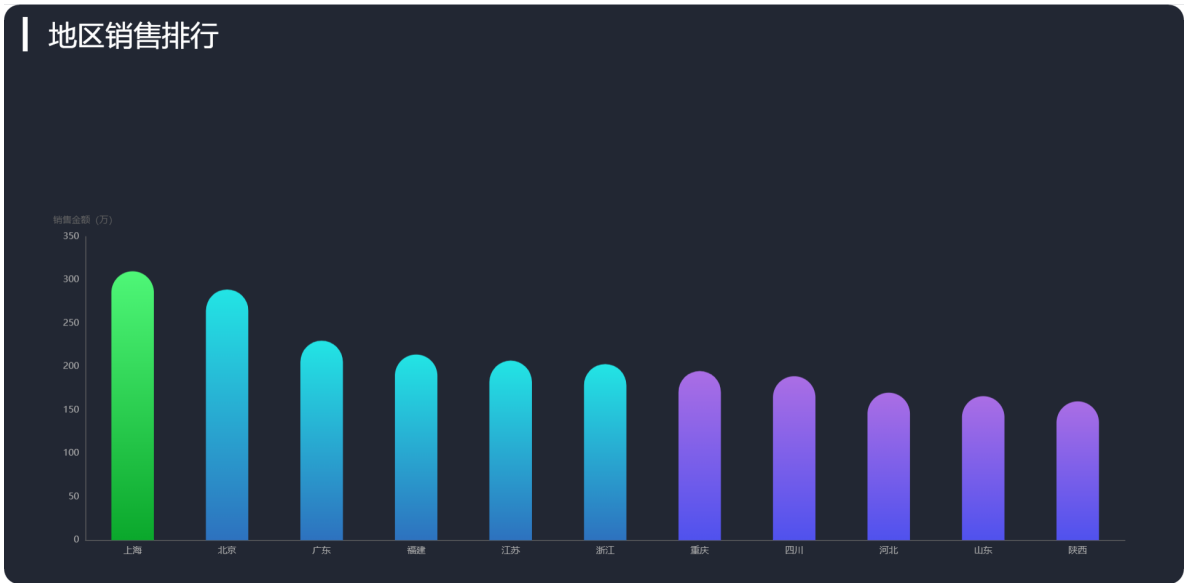
- 回到中国地图

```
<template>
  <div class='com-container' @dblclick="revertMap">
    <div class='com-chart' ref='map_ref'></div>
  </div>
```

```
</template>
<script>
export default {
  ......
  methods: {
    ......
    revertMap () {
      this.chartInstance.setOption({
        geo: {
          map: 'china'
        }
      })
    }
  }
}
</script>
```

2.4.地区销量排行

最终的效果如下:



2.4.1.代码环境的准备

`RankPage.vue`

```
<!--
针对于 /rankpage 这条路径而显示出来的
在这个组件中，通过子组件注册的方式，要显示出Rank.vue这个组件
-->
<template>
  <div class="com-page">
    <rank></rank>
  </div>
</template>

<script>
import Rank from '@/components/Rank'
export default {
  data () {
```

```
    return {}
  },
  methods: {},
  components: {
    rank: Rank
  }
}
</script>

<style lang="less" scoped>
</style>
```

Rank.vue

```
<!-- 地区销售排行 -->
<template>
  <div class='com-container'>
    <div class='com-chart' ref='rank_ref'></div>
  </div>
</template>

<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null
    }
  },
  mounted () {
    this.initChart()
    this.getData()
    window.addEventListener('resize', this.screenAdapter)
    this.screenAdapter()
  },
  destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
  },
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.rank_ref)
      const initOption = {}
      this.chartInstance.setOption(initOption)
    },
    async getData () {
      // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
      this.updateChart()
    },
    updateChart () {
      // 处理图表需要的数据
      const dataOption = {}
      this.chartInstance.setOption(dataOption)
    },
    screenAdapter () {
      const adapterOption = {}
      this.chartInstance.setOption(adapterOption)
      this.chartInstance.resize()
```

```
    }
  }
}
</script>

<style lang='less' scoped>
</style>
```

`router/index.js`

```
......
import RankPage from '@/views/RankPage'
......
const routes = [
  ......
  {
    path: '/rankpage',
    component: RankPage
  }
]
......
```

2.4.2.图表基本功能的实现

- 数据的获取

```
async getData () {
    // 获取服务器的数据，对this.allData进行赋值之后，调用updateChart方法更新图表
    const { data: ret } = await this.$http.get('rank')
    this.allData = ret
    // 对数据进行排序，从大到小排序
    this.allData.sort((a, b) => {
      return b.value - a.value
    })
    this.updateChart()
},
```

- 数据的处理

```
updateChart () {
    // 处理图表需要的数据
    const provinceArr = this.allData.map(item => {
        return item.name
    })
    const valueArr = this.allData.map(item => {
        return item.value
    })
    const dataOption = {
        xAxis: {
            data: provinceArr
        },
        series: [
            {
                data: valueArr
            }
        ]
```

```
    }
    this.chartInstance.setOption(dataOption)
},
```

- 初始化配置

```
initChart () {
    this.chartInstance = this.$echarts.init(this.$refs.rank_ref)
    const initOption = {
        xAxis: {
            type: 'category'
        },
        yAxis: {
            type: 'value'
        },
        series: [
            {
                type: 'bar'
            }
        ]
    }
    this.chartInstance.setOption(initOption)
}
```

2.4.3. UI 效果调整

- 主题的使用

```
initChart () {
    this.chartInstance = this.$echarts.init(this.$refs.rank_ref, 'chalk')
```

- 标题的设置

```
initChart () {
    this.chartInstance = this.$echarts.init(this.$refs.rank_ref, 'chalk')
    const initOption = {
        title: {
            text: '▌ 地区销售排行',
            left: 20,
            top: 20
        }
```

- 坐标轴大小和位置

```
initChart () {
    this.chartInstance = this.$echarts.init(this.$refs.rank_ref, 'chalk')
    const initOption = {
        ......
        grid: {
            top: '40%',
            left: '5%',
            bottom: '5%',
            right: '5%',
            containLabel: true
        }
```

- 工具提示

```
tooltip: {
    show: true
}
```

- 颜色的设置
  - 不同柱显示不同颜色
  - 渐变的控制

```
updateChart () {
    // 处理图表需要的数据
    const colorArr = [
        ['#0BA82C', '#4FF778'],
        ['#2E72BF', '#23E5E5'],
        ['#5052EE', '#AB6EE5']
    ]
    ......
    const dataOption = {
        xAxis: {
            data: provinceArr
        },
        series: [
            {
                data: valueArr,
                itemStyle: {
                    color: arg => {
                        let targetColorArr = colorArr[0]
                        if (arg.vaule >= 300) {
                            targetColorArr = colorArr[0]
                        } else if (arg.value >= 200) {
                            targetColorArr = colorArr[1]
                        } else {
                            targetColorArr = colorArr[2]
                        }
                        return new this.$echarts.graphic.LinearGradient(0,
1, 0, 0, [
                            {
                                offset: 0,
                                color: targetColorArr[0]
                            },
                            {
                                offset: 1,
```

```
                                color: targetColorArr[1]
                            }
                        ])
                    }
                }
            }
        ]
    }
    this.chartInstance.setOption(dataOption)
},
```

2.4.4.平移动画的实现

平移动画可以使用 `dataZoom` 中的 `startValue` 和 `endValue` 来实现

- 定义数据

```
<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null,
      startValue: 0,
      endValue: 9
    }
  },
```

- 将 `startValue` 和 `endValue` 应用在 `dataZoom` 上, 并隐藏 `dataZoom` 的显示

```
updateChart () {
    ......
    const dataOption = {
        xAxis: {
            data: provinceArr
        },
        dataZoom: {
            show: false,
            startValue: this.startValue,
            endValue: this.endValue
        },
```

- 启动和停止定时器

  增加 `timerId` 的变量, 并且增加一个方法 `startInterval` ,来控制 `startValue` 和 `endValue` 的值

```
<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null,
      startValue: 0,
      endValue: 9,
      timerId: null
    }
```

```
  },
......
  methods: {
    ......
    startInterval () {
      if (this.timerId) {
        clearInterval(this.timerId)
      }
      this.timerId = setInterval(() => {
        this.startValue++
        this.endValue++
        if (this.endValue > this.allData.length - 1) {
          this.startValue = 0
          this.endValue = 9
        }
        this.updateChart()
      }, 3000)
    }
  }
}
```

- 获取数据之后启动

```
async getData () {
    ......
    this.updateChart()
    this.startInterval()
},
```

- 组件销毁停止

```
destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
    clearInterval(this.timerId)
},
```

- 鼠标移入停止

```
methods: {
    initChart () {
      ......
      this.chartInstance.setOption(initOption)
      this.chartInstance.on('mouseover', () => {
        clearInterval(this.timerId)
      })
```

- 鼠标离开启动

```
methods: {
    initChart () {
        ......
        this.chartInstance.on('mouseout', () => {
            this.startInterval()
        })
    },
```

2.4.5.分辨率适配

- 计算`titleFontSize`

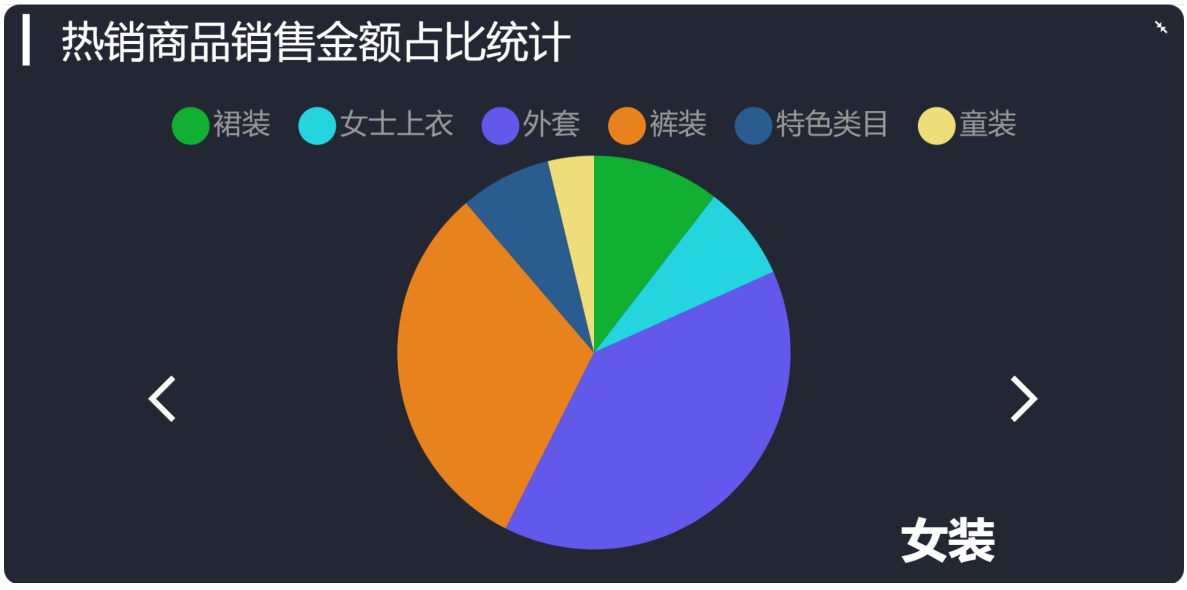
```
screenAdapter () {
    const titleFontSize = this.$refs.rank_ref.offsetWidth / 100 * 3.6
```

- 将`titleFontSize`设置给图表的某些区域

```
screenAdapter () {
    const titleFontSize = this.$refs.rank_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        title: {
            textStyle: {
                fontSize: titleFontSize
            }
        },
        series: [
            {
                barWidth: titleFontSize,
                itemStyle: {
                    barBorderRadius: [0.5 * titleFontSize, 0.5 *
titleFontSize, 0, 0]
                }
            }
        ]
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
},
```

2.5.热销商品占比

最终的效果如下:



2.5.1.代码环境的准备

`HotPage.vue`

```
<!--
针对于 /hotpage 这条路径而显示出来的
在这个组件中，通过子组件注册的方式，要显示出Hot.vue这个组件
-->
<template>
  <div class="com-page">
    <hot></hot>
  </div>
</template>

<script>
import Hot from '@/components/Hot'
export default {
  data () {
    return {}
  },
  methods: {},
  components: {
    hot: Hot
  }
}
</script>

<style lang="less" scoped>
</style>
```

Hot.vue

```
<!-- 热销商品图表 -->
<template>
  <div class='com-container'>
    <div class='com-chart' ref='hot_ref'></div>
  </div>
</template>

<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null
    }
  },
  mounted () {
    this.initChart()
    this.getData()
    window.addEventListener('resize', this.screenAdapter)
    this.screenAdapter()
  },
  destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
  },
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.hot_ref)
      const initOption = {}
```

```
      this.chartInstance.setOption(initOption)
    },
    async getData () {
      // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
      this.updateChart()
    },
    updateChart () {
      // 处理图表需要的数据
      const dataOption = {}
      this.chartInstance.setOption(dataOption)
    },
    screenAdapter () {
      const adapterOption = {}
      this.chartInstance.setOption(adapterOption)
      this.chartInstance.resize()
    }
  }
}
</script>

<style lang='less' scoped>
</style>
```

`router/index.js`

```
......
import HotPage from '@/views/HotPage'
......
const routes = [
  ......
  {
    path: '/hotpage',
    component: HotPage
  }
]
......
```

2.5.2.图表基本功能的实现

- 数据的获取

```
async getData () {
    // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
    const { data: ret } = await this.$http.get('hotproduct')
    this.allData = ret
    this.updateChart()
},
```

- 数据的处理

  增加 `currentIndex` 索引代表当前显示的数据索引, 后期通过左右箭头改变 `currentIndex` 的值

```
<script>
export default {
  data () {
```

```
    return {
      chartInstance: null,
      allData: null,
      currentIndex: 0
    }
  },
  ......
  updateChart () {
      // 处理图表需要的数据
      // 饼图数据
      const seriesData = this.allData[this.currentIndex].children.map(item
=> {
          return {
              value: item.value,
              name: item.name
          }
      })
      // 图例数据
      const legendData = this.allData[this.currentIndex].children.map(item
=> {
          return item.name
      })
      const dataOption = {
          legend: {
              data: legendData
          },
          series: [
              {
                  data: seriesData
              }
          ]
      }
      this.chartInstance.setOption(dataOption)
  },
```

初始化配置

```
methods: {
    initChart () {
        this.chartInstance = this.$echarts.init(this.$refs.hot_ref)
        const initOption = {
            title: {
              text: '▌ 热销商品销售金额占比统计',
              left: 20,
              top: 20
            },
            series: [
                {
                    type: 'pie'
                }
            ]
        }
        this.chartInstance.setOption(initOption)
    },
```

2.5.3.切换数据的实现

- 布局

```
<!-- 热销商品图表 -->
<template>
  <div class='com-container'>
    <div class='com-chart' ref='hot_ref'></div>
    <span class="iconfont arr_left">&#xe6ef;</span>
    <span class="iconfont arr_right">&#xe6ed;</span>
  </div>
</template>
```

- 样式

```
<style lang='less' scoped>
.arr_left {
  position: absolute;
  left: 10%;
  top: 50%;
  transform: translateY(-50%);
  cursor: pointer;
}
.arr_right {
  position: absolute;
  right: 10%;
  top: 50%;
  transform: translateY(-50%);
  cursor: pointer;
}
</style>
```

- 点击事件

```
<span class="iconfont arr_left" @click="toLeft">&#xe6ef;</span>
<span class="iconfont arr_right" @click="toRight">&#xe6ed;</span>
 methods: {
    toLeft () {
      this.currentIndex--
      if (this.currentIndex < 0) {
        this.currentIndex = this.allData.length - 1
      }
      this.updateChart()
    },
    toRight () {
      this.currentIndex++
      if (this.currentIndex > this.allData.length - 1) {
        this.currentIndex = 0
      }
      this.updateChart()
    }
  }
```

- 分类名称的显示
  - 布局和样式

```
<!-- 热销商品图表 -->
```

```
<template>
  <div class='com-container'>
    ......
    <span class="cat_name">分类名称</span>
  </div>
</template>

<style lang='less' scoped>
.cat_name {
  position: absolute;
  left: 80%;
  bottom: 20px;
  font-weight: bold;
}
</style>
```

- 名称的改变

  增加计算属性`catTitle`

```
<script>
export default {
  ......
  computed: {
    catTitle () {
      if (!this.allData) {
        return ''
      }
      return this.allData[this.currentIndex].name
    }
  },
```

  布局中使用计算属性

```
<!-- 热销商品图表 -->
<template>
  <div class='com-container'>
    ......
    <span class="cat_name">{{ catTitle }}</span>
  </div>
</template>
```

2.5.4. `UI`效果的调整

- 主题的使用

```
methods: {
    initChart () {
        this.chartInstance = this.$echarts.init(this.$refs.hot_ref, 'chalk')
```

- 分类名称和箭头的颜色

```
<style lang='less' scoped>
```

```
.arr_left {
  ......
  color: white;
}
.arr_right {
  ......
  color: white;
}
.cat_name {
  .....
  color: white;
}
</style>
```

- 默认隐藏文字, 高亮显示文字

```
   methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.hot_ref, 'chalk')
      const initOption = {
        ......
        series: [
          {
            type: 'pie',
            label: { // 隐藏文字
              show: false
            },
            labelLine: { // 隐藏线
              show: false
            },
            emphasis: {
              label: { // 高亮显示文字
                show: true
              }
            }
          }
        ]
      }
      this.chartInstance.setOption(initOption)
    },
```

- 图例形状和位置

```
methods: {
    initChart () {
        this.chartInstance = this.$echarts.init(this.$refs.hot_ref, 'chalk')
        const initOption = {
            legend: {
                top: '5%',
                icon: 'circle'
            },
```

- 工具提示

  当鼠标移入某个扇区的时候, 需要将该二级分类之下的三级分类数据进行展示

  增加`series`下饼图每一个扇区的数据

```
updateChart () {
    // 处理图表需要的数据
    const seriesData = this.allData[this.currentIndex].children.map(item =>
{
        return {
            ......
            children: item.children
        }
    })
```

显示`tooltip`,并控制显示内容

```
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.hot_ref, 'chalk')
      const initOption = {
        ......
        tooltip: {
          trigger: 'item',
          formatter: function(params) {
            let tipArray = []
            params.data.children.forEach(function(item) {
              let childStr = `
              ${item.name}   
              ${parseInt((item.value / params.value) * 100) + '%'}
            `
              tipArray.push(childStr)
            })
            return tipArray.join('<br/>')
          }
        },
```

2.5.5.分辨率适配

分辨率适配主要就是在`screenAdapter`方法中进行, 需要获取图表容器的宽度,计算出标题字体大小,

将字体的大小赋值给`titleFontSize`

```
<script>
export default {
  data () {
    return {
      titleFontSize: 0
    }
  },
......
screenAdapter () {
    this.titleFontSize = this.$refs.hot_ref.offsetWidth / 100 * 3.6
}
```

通过`titleFontSize`从而设置给图表某些区域

- 标题大小

```
screenAdapter () {
    this.titleFontSize = this.$refs.hot_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        title: {
            textStyle: {
                fontSize: this.titleFontSize
            }
        }
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
},
```

- 饼图大小和位置

```
screenAdapter () {
    this.titleFontSize = this.$refs.hot_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        ......
        series: [
            {
                radius: this.titleFontSize * 4.5,
                center: ['50%', '60%'],
            }
        ]
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
},
```

- 图例大小

```
screenAdapter () {
    this.titleFontSize = this.$refs.hot_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        ......
        legend: {
            itemWidth: this.titleFontSize / 2,
            itemHeight: this.titleFontSize / 2,
            itemGap: this.titleFontSize / 2,
            textStyle: {
                fontSize: this.titleFontSize / 2
            }
        },
        ......
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
},
```

- 箭头大小和分类名称

  定义计算属性`comStyle`

```
computed: {
    ......
    comStyle () {
        return {
            fontSize: this.titleFontSize + 'px'
        }
    }
},
```

将`comStyle`通过`:style`的方式作用到箭头和分类上

```
<template>
  <div class='com-container'>
    <div class='com-chart' ref='hot_ref'></div>
    <span class="iconfont arr_left" @click="toLeft"
:style="comStyle">&#xe6ef;</span>
    <span class="iconfont arr_right" @click="toRight"
:style="comStyle">&#xe6ed;</span>
    <span class="cat_name" :style="comStyle">{{ catTitle }}</span>
  </div>
</template>
```

2.6.库存销量分析

最终的效果如下:

2.6.1.代码环境的准备

`StockPage.vue`

```
<!--
针对于 /stockpage 这条路径而显示出来的
在这个组件中，通过子组件注册的方式，要显示出Stock.vue这个组件
-->
<template>
  <div class="com-page">
    <stock></stock>
  </div>
</template>
```

```
<script>
import Stock from '@/components/Stock'
export default {
  data () {
    return {}
  },
  methods: {},
  components: {
    stock: Stock
  }
}
</script>

<style lang="less" scoped>
</style>
```

`Stock.vue`

```
<!-- 库存销量分析 -->
<template>
  <div class='com-container'>
    <div class='com-chart' ref='stock_ref'></div>
  </div>
</template>

<script>
export default {
  data () {
    return {
      chartInstance: null,
      allData: null
    }
  },
  mounted () {
    this.initChart()
    this.getData()
    window.addEventListener('resize', this.screenAdapter)
    this.screenAdapter()
  },
  destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
  },
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.stock_ref)
      const initOption = {}
      this.chartInstance.setOption(initOption)
    },
    async getData () {
        // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
      this.updateChart()
    },
    updateChart () {
      // 处理图表需要的数据
      const dataOption = {}
```

```
      this.chartInstance.setOption(dataOption)
    },
    screenAdapter () {
      const adapterOption = {}
      this.chartInstance.setOption(adapterOption)
      this.chartInstance.resize()
    }
  }
}
</script>

<style lang='less' scoped>
</style>
```

`router/index.js`

```
......
import StockPage from '@/views/StockPage'
......
const routes = [
  ......
  {
    path: '/stockpage',
    component: StockPage
  }
]
......
```

2.6.2.图表基本功能的实现

- 数据的获取

```
async getData () {
    // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
    const { data: ret } = await this.$http.get('stock')
    this.allData = ret
    this.updateChart()
},
```

- 数据的处理, 要显示5个圆环的实现

```
updateChart () {
    // 处理图表需要的数据
    // 5个圆环对应的圆心点
    const centerPointers = [
        ['18%', '40%'],
        ['50%', '40%'],
        ['82%', '40%'],
        ['34%', '75%'],
        ['66%', '75%']
    ]
    // 先显示前5条数据
    const showData = this.allData.slice(0, 5)
    const seriesArr = showData.map((item, index) => {
        return {
```

```
            type: 'pie',
            center: centerPointers[index],
            radius: [110, 100],
            data: [
                {
                    value: item.sales
                },
                {
                    value: item.stock,
                }
            ]
        }
    })
    const dataOption = {
        series: seriesArr
    }
    this.chartInstance.setOption(dataOption)
},
```

- 标题的配置

```
initChart () {
    this.chartInstance = this.$echarts.init(this.$refs.stock_ref)
    const initOption = {
        title: {
            text: '▌ 库存销售量',
            left: 20,
            top: 20
        }
    }
    this.chartInstance.setOption(initOption)
},
```

2.6.3. UI 效果的调整

- 主题的使用

```
methods: {
    initChart () {
        this.chartInstance = this.$echarts.init(this.$refs.stock_ref,
'chalk')
```

- 鼠标移入动画的移除

```
updateChart () {
    ......
    const seriesArr = showData.map((item, index) => {
        return {
            ......
            hoverAnimation: false,
```

- 指示线的移除

```
updateChart () {
    ......
    const seriesArr = showData.map((item, index) => {
        return {
            ......
            labelLine: {
                show: false
            },
```

- 圆环内文字的显示

  给饼图第一部分数据增加`name`属性

```
updateChart () {
    ......
    const seriesArr = showData.map((item, index) => {
        return {
            ......
            data: [
                {
                    name: item.name + '\n\n' + item.sales,
                    value: item.sales,
                    ......
```

  设置`label`的显示位置

```
updateChart () {
    ......
    const seriesArr = showData.map((item, index) => {
        return {
            .....
            label: {
                show: true,
                position: 'center'
            },
```

- 颜色的处理

```
    updateChart () {
      // 增加5个圆环的渐变颜色范围
      const colorArrs = [
        ['#4FF778', '#0BA82C'],
        ['#E5DD45', '#E8B11C'],
        ['#E8821C', '#E55445'],
        ['#5052EE', '#AB6EE5'],
        ['#23E5E5', '#2E72BF']
      ]
      ......
      const seriesArr = showData.map((item, index) => {
        return {
          ......
          label: {
            show: true,
            position: 'center',
            color: colorArrs[index][0]
          },
```

```
          data: [
            {
              name: item.name + '\n\n' + item.sales,
              value: item.sales,
              itemStyle: {
                color: new this.$echarts.graphic.LinearGradient(0, 1, 0, 0,
[
                    {
                      offset: 0,
                      color: colorArrs[index][0]
                    },
                    {
                      offset: 1,
                      color: colorArrs[index][1]
                    }
                ])
              }
            },
            {
              value: item.stock,
              itemStyle: {
                color: '#333843'
              }
            }
          ]
        }
      })
      const dataOption = {
        series: seriesArr
      }
      this.chartInstance.setOption(dataOption)
    },
```

2.6.4.切换动画

- 增加数据`currentIndex`,标识当前的页数

```
<script>
export default {
  data () {
    return {
      ......
      currentIndex: 0,
      timerId: null
    }
  },
```

- 根据`currentIndex`决定展示的数据

```
updateChart () {
    ......
    const start = this.currentIndex * 5
    const end = (this.currentIndex + 1) * 5
    const showData = this.allData.slice(start, end)
    const seriesArr = showData.map((item, index) => {
    ......
```

- 数据获取成功之后启动动画

```
async getData () {
    // 获取服务器的数据, 对this.allData进行赋值之后, 调用updateChart方法更新图表
    const { data: ret } = await this.$http.get('stock')
    this.allData = ret
    this.updateChart()
    this.startInterval()
},
......
startInterval () {
    if (this.timerId) {
        clearInterval(this.timerId)
    }
    this.timerId = setInterval(() => {
        this.currentIndex++
        if (this.currentIndex > 1) {
            this.currentIndex = 0
        }
        this.updateChart()
    }, 3000)
}
```

- 组件销毁时停止动画

```
destroyed () {
    window.removeEventListener('resize', this.screenAdapter)
    clearInterval(this.timerId)
},
```

- 鼠标事件的处理

```
methods: {
    initChart () {
        ......
        this.chartInstance.on('mouseover', () => {
            clearInterval(this.timerId)
        })
        this.chartInstance.on('mouseout', () => {
            this.startInterval()
        })
    },
```

2.6.5.分辨率适配

分辨率适配主要就是在`screenAdapter`方法中进行, 需要获取图表容器的宽度,计算出标题字体大小,

将字体的大小赋值给`titleFontSize`

```
<script>
export default {
  data () {
    return {
      ......
      titleFontSize: 0
    }
  },
......
screenAdapter () {
    this.titleFontSize = this.$refs.stock_ref.offsetWidth / 100 * 3.6
}
```

通过`titleFontSize`从而设置给图表某些区域

- 标题大小

```
screenAdapter () {
    this.titleFontSize = this.$refs.stock_ref.offsetWidth / 100 * 3.6
    const adapterOption = {
        title: {
            textStyle: {
                fontSize: this.titleFontSize
            }
        }
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
},
```

- 圆环半径和圆环文字

```
screenAdapter () {
    this.titleFontSize = this.$refs.stock_ref.offsetWidth / 100 * 3.6
    const innerRadius = this.titleFontSize * 2
    const outterRadius = innerRadius * 1.125
    const adapterOption = {
        title: {
            textStyle: {
                fontSize: this.titleFontSize
            }
        },
        series: [
            {
                radius: [outterRadius, innerRadius],
                label: {
                    fontSize: this.titleFontSize / 2
                }
            },
            {
                radius: [outterRadius, innerRadius],
                label: {
                    fontSize: this.titleFontSize / 2
                }
            },
            {
```

```
                radius: [outterRadius, innerRadius],
                label: {
                    fontSize: this.titleFontSize / 2
                }
            },
            {
                radius: [outterRadius, innerRadius],
                label: {
                    fontSize: this.titleFontSize / 2
                }
            },
            {
                radius: [outterRadius, innerRadius],
                label: {
                    fontSize: this.titleFontSize / 2
                }
            }
        ]
    }
    this.chartInstance.setOption(adapterOption)
    this.chartInstance.resize()
}
```

- 调整 `initOption` 和 `dataOption`

  `initOption`

```
methods: {
    initChart () {
        this.chartInstance = this.$echarts.init(this.$refs.stock_ref,
'chalk')
        const centerPointers = [
            ['18%', '40%'],
            ['50%', '40%'],
            ['82%', '40%'],
            ['34%', '75%'],
            ['66%', '75%']
        ]
        const initOption = {
            title: {
                text: '▌ 库存销售量',
                left: 20,
                top: 20
            },
            series: [
                {
                    type: 'pie',
                    center: centerPointers[0],
                    hoverAnimation: false,
                    label: {
                        show: true,
                        position: 'center'
                    },
                    labelLine: {
                        show: false
                    }
                },
```

```
                {
                    type: 'pie',
                    center: centerPointers[1],
                    hoverAnimation: false,
                    label: {
                        show: true,
                        position: 'center'
                    },
                    labelLine: {
                        show: false
                    }
                },
                {
                    type: 'pie',
                    center: centerPointers[2],
                    hoverAnimation: false,
                    label: {
                        show: true,
                        position: 'center'
                    },
                    labelLine: {
                        show: false
                    }
                },
                {
                    type: 'pie',
                    center: centerPointers[3],
                    hoverAnimation: false,
                    label: {
                        show: true,
                        position: 'center'
                    },
                    labelLine: {
                        show: false
                    }
                },
                {
                    type: 'pie',
                    center: centerPointers[4],
                    hoverAnimation: false,
                    label: {
                        show: true,
                        position: 'center'
                    },
                    labelLine: {
                        show: false
                    }
                }
            ]
        }
        this.chartInstance.setOption(initOption)
        this.chartInstance.on('mouseover', () => {
            clearInterval(this.timerId)
        })
        this.chartInstance.on('mouseout', () => {
            this.startInterval()
        })
    },
```

`dataOption`

```
updateChart () {
    // 处理图表需要的数据
    const colorArrs = [
        ['#4FF778', '#0BA82C'],
        ['#E5DD45', '#E8B11C'],
        ['#E8821C', '#E55445'],
        ['#5052EE', '#AB6EE5'],
        ['#23E5E5', '#2E72BF'],
    ]
    const start = this.currentIndex * 5
    const end = (this.currentIndex + 1) * 5
    const showData = this.allData.slice(start, end)
    const seriesArr = showData.map((item, index) => {
        return {
            label: {
                color: colorArrs[index][0]
            },
            data: [
                {
                    name: item.name + '\n\n' + item.sales,
                    value: item.sales,
                    itemStyle: {
                        color: new this.$echarts.graphic.LinearGradient(0,
1, 0, 0, [
                            {
                                offset: 0,
                                color: colorArrs[index][0]
                            },
                            {
                                offset: 1,
                                color: colorArrs[index][1]
                            }
                        ])
                    }
                },
                {
                    value: item.stock,
                    itemStyle: {
                        color: '#333843'
                    }
                }
            ]
        }
    })
    const dataOption = {
        series: seriesArr
    }
    this.chartInstance.setOption(dataOption)
}
```

3.WebSocket的引入

`WebSocket`可以保持着浏览器和客户端之间的长连接， 通过`WebSocket`可以实现数据由后端推送到前端，保证了数据传输的实时性.`WebSocket`涉及到前端代码和后端代码的改造

3.1.后端代码的改造

3.1.1.`WebSocket`的使用

- 安装`WebSocket`包

```
npm i ws -S
```

- 创建`WebSocket`实例对象

```
const WebSocket = require("ws")
// 创建出WebSocket实例对象
const wss = new WebSocket.Server({
  port: 9998
})
```

- 监听事件

```
wss.on("connection", client => {
    console.log("有客户端连接...")
    client.on("message", msg => {
        console.log("客户端发送数据过来了")
        // 发送数据给客户端
        client.send('hello socket')
    })
})
```

- 前端的测试代码如下:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <button id="connect">连接</button>
  <button id="send" disabled="true">发送数据</button> <br>
  从服务器接收的数据如下:<br>
  <span id="content"></span>
  <script>
    var connect = document.querySelector('#connect')
    var send = document.querySelector('#send')
    var content = document.querySelector('#content')
    var ws = null
    connect.onclick = function() {
      ws = new WebSocket('ws://localhost:9998')
      ws.onopen = () => {
        console.log('连接服务器成功')
        send.disabled = false
```

```
      }
      ws.onmessage = msg => {
        console.log('从服务器接收到了数据')
        content.innerHTML = msg.data
      }
      ws.onclose = e => {
        console.log('服务器关闭了连接')
        send.disabled = true
      }
    }
    send.onclick = function(){
      ws.send('hello websocket from frontend')
    }
  </script>
</body>
</html>
```

3.1.2.改造后端程序`Koa_Server`

- 1.创建`service\web_socket_service.js`文件

```
const WebSocket = require("ws")
// 创建出WebSocket实例对象
const wss = new WebSocket.Server({
  port: 9998
})

module.exports.listen = function() {
  wss.on("connection", ws => {
    console.log("有客户端连接...")
    ws.on("message", msg => {
      console.log("客户端发送数据过来了")
    })
  })
}
```

- 2.在`app.js`中引入`web_socket_service.js`这个文件, 并调用`listen`方法

```
const webSocketService = require('./service/web_socket_service')
webSocketService.listen()
```

- 3.约定好和客户端之间数据交互的格式和含义
  - 客户端和服务端之间的数据交互采用`JSON`格式
  - 客户端发送数据给服务端的字段如下:

```
{
    "action": "getData",
    "socketType": "trendData",
    "chartName": "trend",
    "value": ""
}
或者
```

```
{
    "action": "fullScreen",
    "socketType": "fullScreen",
    "chartName": "trend",
    "value": true
}
或者
{
    "action": "themeChange",
    "socketType": "themeChange",
    "chartName": "",
    "value": "chalk"
}
```

其中:

- `action`: 代表某项行为,可选值有
  - `getData` 代表获取图表数据
  - `fullScreen` 代表产生了全屏事件
  - `themeChange` 代表产生了主题切换的事件
- `socketType`: 代表业务模块类型, 这个值代表前端注册数据回调函数的标识, 可选值有:
  - `trendData`
  - `sellerData`
  - `mapData`
  - `rankData`
  - `hotData`
  - `stockData`
  - `fullScreen`
  - `themeChange`
- `chartName`: 代表图表名称, 如果是主题切换事件, 可不传此值, 可选值有:
  - `trend`
  - `seller`
  - `map`
  - `rank`
  - `hot`
  - `stock`
- `value`: 代表 具体的数据值, 在获取图表数据时, 可不传此值, 可选值有
  - 如果是全屏事件, `true`代表全屏, `false`代表非全屏
  - 如果是主题切换事件, 可选值有`chalk`或者`vintage`

- 服务端发送给客户端的数据如下:

```
{
    "action": "getData",
    "socketType": "trendData",
    "chartName": "trend",
    "value": "",
    "data": "从文件读取出来的json文件的内容"
}
或者
{
    "action": "fullScreen",
    "socketType": "fullScreen",
```

```
    "chartName": "trend",
    "value": true
}
或者
{
    "action": "themeChange",
    "socketType": "themeChange",
    "chartName": "",
    "value": "chalk"
}
```

注意, 除了`action`为`getData`时, 服务器会在客户端发过来数据的基础之上, 增加`data`字段, 其他的情况, 服务器会原封不动的将从某一个客户端发过来的数据转发给每一个处于连接状态的客户端

- 4.代码实现

```
const path = require('path')
const fileUtils = require('../utils/file_utils')
const WebSocket = require("ws")
// 创建出WebSocket实例对象
const wss = new WebSocket.Server({
  port: 9998
})
module.exports.listen = function() {
  wss.on("connection", client => {
    console.log("有客户端连接...")
    client.on("message", async msg => {
      let payload = JSON.parse(msg)
      if (payload.action === 'getData') {
        // 返回每个模块的数据
        let filePath = '../data/' + payload.chartName + '.json'  //
../data/seller.json
        filePath = path.join(__dirname, filePath)
        const ret = await fileUtils.getFileJsonData(filePath)
        payload.data = ret // 增加data字段
        client.send(JSON.stringify(payload))
      }  else {
        // 主题切换, 全屏切换, 进行每个客户端的同步, 收到什么数据就发送什么数据
        wss.clients.forEach(client => {
            console.log("******************************************")
            client.send(message)
        })
      }
    })
  })
}
```

3.2.前端代码的改造

3.2.1.定义单例,创建`WebSocket`实例对象

- 创建`src/utils/socket_service.js`文件
- 定义单例

```
export default class SocketService {
  /**
   * 单例
   */
  static instance = null
  static get Instance () {
    if (!this.instance) {
      this.instance = new SocketService()
    }
    return this.instance
  }
}
```

3.2.2.监听`webSocket`事件

- 定义`connect`函数,将创建的`webSocket`赋值给实例属性

```
export default class SocketService {
   ......
  // 实例属性
  ws = null
  // 初始化连接websocket
  connect () {
    if (!window.WebSocket) {
      return console.log('您的浏览器不支持 WebSocket!')
    }
    this.ws = new WebSocket('ws://localhost:9998')
  }
}
```

- 监听事件

```
connect () {
    if (!window.WebSocket) {
        return console.log('您的浏览器不支持 WebSocket!')
    }
    this.ws = new WebSocket('ws://localhost:9998')
    // 监听连接成功
    this.ws.onopen = () => {
        console.log('WebSocket 连接成功')
    }
    // 1.服务器连接不成功  2.服务器关闭了连接
    this.ws.onclose = e => {
        console.log('服务器关闭了连接')
    }
    // 监听接收消息
    this.ws.onmessage = msg => {
        console.log('WebSocket 接收到数据')
    }
}
```

3.2.3.定义注册函数

记录一下当得到数据时, 应该调用的函数回调

```
export default class SocketService {

  // 业务类型和回调函数的对于关系
  callBackMapping = {}
  /**
   * socketType
   *    trendData sellerData mapData rankData hotData stockData
   *    fullScreen
   *    themeChange
   * callBack
   *    回调函数
   */
  registerCallBack (socketType, callBack) {
     // 往 callBackMap中存放回调函数
    this.callBackMapping[socketType] = callBack
  }

  unRegisterCallBack (socketType) {
    this.callBackMapping[socketType] = null
  }
}
```

3.2.4.连接服务端

- 在`main.js`中连接服务器端

  ```
  import SocketService from '@/utils/socket_service'
  SocketService.Instance.connect()
  ```

- 将`SocketService`实例对象挂载到`Vue`的原型对象上

  ```
  Vue.prototype.$socket = SocketService.Instance
  ```

3.2.5.发送数据给服务端

- 在`socket_service.js`中定义发送数据的方法

  ```
  export default class SocketService {
    ......
    send (data) {
      console.log('发送数据给服务器:')
      this.ws.send(JSON.stringify(data))
    }
  }
  ```

- 先对修改`Trend.vue`进行代码改造
  - `created`中注册回调函数

```
created () {
    // 当socket来数据的时候, 会调用getData这个函数
    this.$socket.registerCallBack('trendData', this.getData)
}
```

- `destroyed`中取消注册

```
destroyed () {
    this.$socket.unRegisterCallBack('trendData')
}
```

- `mounted`中往`socket`发送数据, 目的是想让服务端传输销量趋势这个模块的数据

```
mounted () {
    this.initChart()
    // this.getData() 先将getData的调用注释起来
    this.$socket.send({
        action: 'getData',
        socketType: 'trendData',
        chartName: 'trend'
    })
    ......
},
```

- 运行代码, 发现数据发不出去

  因为在刷新界面之后, 客户端和服务端的连接并不会立马连接成功, 在处于连接状态下就调用`send`是发送不成功的, 因此需要修改`service_socket.js`中的`send`方法进行容错处理

```
// 是否已经连接成功
connected = false
sendRetryCount = 0

send (data) {
    console.log('发送数据给服务器:')
    if (this.connected) {
      this.sendRetryCount = 0
      this.ws.send(JSON.stringify(data))
    } else {
      setTimeout(() => {
        this.sendRetryCount++
        this.send(data)
      }, 200 * this.sendRetryCount) // 发送数据尝试的次数越大, 则下一次连接的延迟也就越长
    }
  }
```

  在`onopen`时设置`connected`的值

```
connect () {
    ......
    this.ws.onopen = () => {
        console.log('WebSocket 连接成功')
        this.connected = true
    }
}
```

- 在`socket_service.js`中修改接收到消息的代码处理

```
connect () {
    // 监听接收消息
    this.ws.onmessage = msg => {
        console.log('WebSocket 接收到数据')
        const recvData = JSON.parse(msg.data) // 取出服务端传递的数据
        const socketType = recvData.socketType // 取出业务类型,要根据业务类
型,得到回调函数
        // 先判断有没有回调函数
        if (this.callBackMapping[socketType]) {
            if (recvData.action === 'getData') {
                const realData = recvData.data // 得到该图表的数据
                this.callBackMapping[socketType].call(this,
JSON.parse(realData))
            }
        }
    }
}
```

3.2.6.断开重连机制

如果初始化连接服务端不成功, 或者连接成功了, 后来服务器关闭了, 这两种情况都会触发`onclose`事件, 我们需要在这个事件中,进行重连

```
connectRetryCount = 0 // 重连次数, 重连次数越大, 下一次再发起重连的延时也就越长

connect () {
    this.ws.onopen = () => {
      ......
      this.connectRetryCount = 0 // 连接成功之后, 重置重连次数
    }
    ......
    // 1.服务器连接不成功  2.服务器关闭了连接
    this.ws.onclose = e => {
        console.log('服务器关闭了连接')
        setTimeout(() => {
            this.connectRetryCount++
            this.connect()
        }, 200 * this.connectRetryCount)
    }
}
```

3.2.7.其他模块的修改

参照`Trend.vue`, 其他图表要得到数据, 需要做这么几件事情

- `created`中注册回调函数

```
created () {
    this.$socket.registerCallBack('xxx', this.getData)
    // xxx的可选值有: trendData,sellerData,mapData,rankData,hotData,stockData
}
```

- `destroyed`中取消注册

```
destroyed () {
    this.$socket.unRegisterCallBack('xxx')
    // xxx的值与注册时对应即可
}
```

- `mounted`中注释`getData`的调用,改成往`socket`发送数据

```
mounted () {
    this.initChart()
    // this.getData() 先将getData的调用注释起来
    this.$socket.send({
        action: 'getData',
        socketType: 'trendData',
        chartName: 'trend'
    })
    ......
},
// action的值不变,都是getData
// socketType的可选值
有:trendData,sellerData,mapData,rankData,hotData,stockData
// chartName的可选值有: trend,seller,map,rank,hot,stock
```

4.其他细节

4.1.合并组件

4.1.1.创建`ScreenPage.vue`文件,并配置路由规则

`router/index.js`

```
import ScreenPage from '@/views/ScreenPage'
Vue.use(VueRouter)
const routes = [
  {
    path: '/',
    redirect: '/screen'
  },
  {
    path: '/screen',
    component: ScreenPage
  }
  ......
]
```

4.1.1.创建布局结构和样式

复制资料中的相关图片到`public/static/img`目录之下

ScreenPage.vue

```
<template>
  <div class="screen-container">
    <header class="screen-header">
      <div>
        <img src="/static/img/header_border_dark.png" alt="">
      </div>
      <span class="logo">
        <img src="/static/img/logo_dark.png" alt="" />
      </span>
      <span class="title">电商平台实时监控系统</span>
      <div class="title-right">
        <img src="/static/img/qiehuan_dark.png" class="qiehuan">
        <span class="datetime">2049-01-01 00:00:00</span>
      </div>
    </header>
    <div class="screen-body">
      <section class="screen-left">
        <div id="left-top">
          <!-- 销量趋势图表 -->
        </div>
        <div id="left-bottom">
          <!-- 商家销售金额图表 -->
        </div>
      </section>
      <section class="screen-middle">
        <div id="middle-top">
          <!-- 商家分布图表 -->
        </div>
        <div id="middle-bottom">
          <!-- 地区销量排行图表 -->
        </div>
      </section>
      <section class="screen-right">
        <div id="right-top">
          <!-- 热销商品占比图表 -->
        </div>
        <div id="right-bottom">
          <!-- 库存销量分析图表 -->
        </div>
      </section>
    </div>
  </div>
</template>

<script>
export default {}
</script>
<style lang="less" scoped>
.screen-container {
  width: 100%;
  height: 100%;
  padding: 0 20px;
  background-color: #161522;
  color: #fff;
  box-sizing: border-box;
```

```
}
.screen-header {
  width: 100%;
  height: 64px;
  font-size: 20px;
  position: relative;
  > div {
      img {
        width: 100%;
      }
    }
  .title {
    position: absolute;
    left: 50%;
    top: 50%;
    font-size: 20px;
    transform: translate(-50%, -50%);
  }
  .title-right {
    display: flex;
    align-items: center;
    position:absolute;
    right: 0px;
    top: 50%;
    transform: translateY(-80%);
  }
  .qiehuan {
    width: 28px;
    height: 21px;
    cursor: pointer;
  }
  .datetime {
    font-size: 15px;
    margin-left: 10px;
  }
  .logo {
    position: absolute;
    left: 0px;
    top: 50%;
    transform: translateY(-80%);
    img {
      height: 35px;
      width: 128px;
    }
  }
}
.screen-body {
  width: 100%;
  height: 100%;
  display: flex;
  margin-top: 10px;
  .screen-left {
    height: 100%;
    width: 27.6%;
    #left-top {
      height: 53%;
    }
    #left-bottom {
```

```
      height: 31%;
      margin-top: 25px;
    }
  }
  .screen-middle {
    height: 100%;
    width: 41.5%;
    margin-left: 1.6%;
    margin-right: 1.6%;
    #middle-top {
      width: 100%;
      height: 56%;
    }
    #middle-bottom {
      margin-top: 25px;
      width: 100%;
      height: 28%;
    }
  }
  .screen-right {
    height: 100%;
    width: 27.6%;
    #right-top {
      height: 46%;
    }
    #right-bottom {
      height: 38%;
      margin-top: 25px;
    }
  }
}
</style>
```

4.1.2.注册组件, 并将其置于合适的容器中

- 组件的引入和注册

```
import Trend from '@/components/Trend'
import Seller from '@/components/Seller'
import Map from '@/components/Map'
import Rank from '@/components/Rank'
import Hot from '@/components/Hot'
import Stock from '@/components/Stock'

export default {
  components: {
    Seller,
    Stock,
    Trend,
    Map,
    Hot,
    Rank
  }
}
```

- 布局如下:

给图表每一个组件都增加上`ref`属性

```
<div class="screen-body">
      <section class="screen-left">
        <div id="left-top">
          <!-- 销量趋势图表 -->
          <Trend ref="trend"></Trend>
        </div>
        <div id="left-bottom">
          <!-- 商家销售金额图表 -->
          <Seller ref="seller"></Seller>
        </div>
      </section>
      <section class="screen-middle">
        <div id="middle-top">
          <!-- 商家分布图表 -->
          <Map ref="map"></Map>
        </div>
        <div id="middle-bottom">
          <!-- 地区销量排行图表 -->
          <Rank ref="rank"></Rank>
        </div>
      </section>
      <section class="screen-right">
        <div id="right-top">
          <!-- 热销商品占比图表 -->
          <Hot ref="hot"></Hot>
        </div>
        <div id="right-bottom">
          <!-- 库存销量分析图表 -->
          <Stock ref="stock"></Stock>
        </div>
      </section>
    </div>
```

- 增加全局样式,如下:

`global.less`

```
.com-container {
  position: relative;
}
```

这个样式主要是给`Trend.vue`和`Hot.vue`这两个组件使用的,因为这两个组件中有元素HTML标签, 需要进行定位

- 调整样式

调整一下`Hot.vue`中图例的大小

```
screenAdapter () {
    const adapterOption = {
        legend: {
            itemWidth: this.titleFontSize,
            itemHeight: this.titleFontSize
        }
    }
},
```

调整一下`Stock.vue`中圆环的大小

```
screenAdapter () {
    const innerRadius = titleFontSize * 2.8
```

4.2.全屏切换

4.2.1.布局调整

- 以销量趋势图表为例, 布局修改如下:

  ```
  <div id="left-top">
      <Trend ref="trend"></Trend>
      <div class="resize">
          <span class="iconfont icon-compress-alt"></span>
      </div>
  </div>
  ```

  其中, `span`标签的`class`的值为`iconfont icon-expand-alt`代表展开图标, `class`值为`iconfont icon-compress-alt`为收缩图标

- 设置resize样式

  ```
  .resize {
    position: absolute;
    top: 20px;
    right: 20px;
    cursor: pointer;
  }
  ```

- 修改各个容器样式, 增加`position`为相对布局

  ```
  <style lang="less" scoped>
  .screen-body {
    .screen-left {
      #left-top {
        position: relative;
      }
      #left-bottom {
        position: relative;
      }
    }
    .screen-middle {
      #middle-top {
        position: relative;
  ```

```
    }
    #middle-bottom {
      position: relative;
    }
  }
  .screen-right {
    #right-top {
      position: relative;
    }
    #right-bottom {
      position: relative;
    }
  }
}
</style>
```

4.2.2.全屏实现

- 全屏状态数据的定义

```
export default {
  data () {
    return {
      fullScreenStatus: {
        trend: false,
        seller: false,
        map: false,
        rank: false,
        hot: false,
        stock: false
      }
    }
  }
}
```

- 全屏状态样式的定义

```
.fullscreen {
  position: fixed!important;
  top: 0 !important;
  left: 0 !important;
  width: 100% !important;
  height: 100% !important;
  margin: 0 !important;
  z-index: 100;
}
```

- `class`值的处理

```
<div id="left-top" :class="[fullScreenStatus.trend ? 'fullscreen' : '']">
    <Trend ref="trend"></Trend>
    <div class="resize">
        <span
              :class="['iconfont', fullScreenStatus.trend ? 'icon-compress-
alt' : 'icon-expand-alt']"
              @click="changeSize('trend')">
        </span>
    </div>
</div>
```

- 点击事件的处理

```
export default {
  methods: {
    changeSize (chartName) {
      // 先得到目标状态
      const targetValue = !this.fullScreenStatus[chartName]
      // 将所有的图表设置为非全屏
      Object.keys(this.fullScreenStatus).forEach(item => {
        this.fullScreenStatus[item] = false
      })
      // 将目标图表设置为目标状态
      this.fullScreenStatus[chartName] = targetValue
      this.$nextTick(() => {
        this.$refs[chartName].screenAdapter()
      })
    }
}
```

- 其他图表的样式和点击事件的处理类似

```
<div class="screen-body">
    <section class="screen-left">
        <div id="left-top" :class="[fullScreenStatus.trend ? 'fullscreen' :
'']">
            <Trend ref="trend"></Trend>
            <div class="resize">
                <span
                      :class="['iconfont', fullScreenStatus.trend ? 'icon-
compress-alt' : 'icon-expand-alt']"
                      @click="changeSize('trend')">
                </span>
            </div>
        </div>
        <div id="left-bottom" :class="[fullScreenStatus.seller ?
'fullscreen' : '']">
            <Seller ref="seller"></Seller>
            <div class="resize">
                <span
                      :class="['iconfont', fullScreenStatus.seller ? 'icon-
compress-alt' : 'icon-expand-alt']"
                      @click="changeSize('seller')">
                </span>
            </div>
        </div>
```

```
    </section>
    <section class="screen-middle">
        <div id="middle-top" :class="[fullScreenStatus.map ? 'fullscreen' :
''']">
            <Map ref="map"></Map>
            <div class="resize">
                <span
                      :class="['iconfont', fullScreenStatus.map ? 'icon-
compress-alt' : 'icon-expand-alt']"
                      @click="changeSize('map')">
                </span>
            </div>
        </div>
        <div id="middle-bottom" :class="[fullScreenStatus.rank ?
'fullscreen' : '']">
            <Rank ref="rank"></Rank>
            <div class="resize">
                <span
                      :class="['iconfont', fullScreenStatus.rank ? 'icon-
compress-alt' : 'icon-expand-alt']"
                      @click="changeSize('rank')">
                </span>
            </div>
        </div>
    </section>
    <section class="screen-right">
        <div id="right-top" :class="[fullScreenStatus.hot ? 'fullscreen' :
''']">
            <Hot ref="hot"></Hot>
            <div class="resize">
                <span
                      :class="['iconfont', fullScreenStatus.hot ? 'icon-
compress-alt' : 'icon-expand-alt']"
                      @click="changeSize('hot')">
                </span>
            </div>
        </div>
        <div id="right-bottom" :class="[fullScreenStatus.stock ?
'fullscreen' : '']">
            <Stock ref="stock"></Stock>
            <div class="resize">
                <span
                      :class="['iconfont', fullScreenStatus.stock ? 'icon-
compress-alt' : 'icon-expand-alt']"
                      @click="changeSize('stock')">
                </span>
            </div>
        </div>
    </section>
</div>
```

4.2.3.全屏事件的数据发送

- 点击按钮发送数据

```
export default {
  methods: {
```

```
    changeSize (chartName) {
      // 先得到目标状态
      const targetValue = !this.fullScreenStatus[chartName]
      // 将所有的图表设置为非全屏
      // Object.keys(this.fullScreenStatus).forEach(item => {
      //   this.fullScreenStatus[item] = false
      // })
      // 将目标图表设置为目标状态
      // this.fullScreenStatus[chartName] = targetValue
      // this.$nextTick(() => {
      //   this.$refs[chartName].screenAdapter()
      // })
      this.$socket.send({
        action: 'fullScreen',
        socketType: 'fullScreen',
        chartName: chartName,
        value: targetValue
      })
    }
  }
}
```

- `created`时注册回调函数

```
export default {
  created () {
    this.$socket.registerCallBack('fullScreen', this.recvData)
  },
}
```

- `destroyed`时取消回调函数

```
export default {
  destroyed () {
    this.$socket.unRegisterCallBack('fullScreen')
  },
}
```

- 得到数据的处理

```
export default {
  methods: {
    recvData (data) {
      // 将所有的图表设置为非全屏
      Object.keys(this.fullScreenStatus).forEach(item => {
        this.fullScreenStatus[item] = false
      })
      // 将目标图表设置为目标状态
      this.fullScreenStatus[data.chartName] = data.value
      // 更新所有图表
      Object.keys(this.fullScreenStatus).forEach(item => {
        this.$nextTick(() => {
          this.$refs[item].screenAdapter()
        })
      })
    }
```

```
  }
}
```

- `socket_service.js`代码的修改

```
if (recvData.action === 'getData') {
    const realData = recvData.data // 得到该图表的数据
    this.callBackMapping[socketType].call(this, JSON.parse(realData))
} else if (action === 'fullScreen') {
    this.callBackMapping[socketType].call(this, recvData)
}
```

4.3.主题切换

4.3.1.当前主题数据的存储

当前主题的数据, 会在多个组件中使用, 因此设置在`vuex`中是最合适的, 增加仓库数据`theme`, 并增加一个`mutation`用来修改`theme`

`store/index.js`

```
export default new Vuex.Store({
  state: {
    theme: 'chalk'
  },
  mutations: {
    changeTheme (state) {
      if (state.theme === 'chalk') {
        state.theme = 'vintage'
      } else {
        state.theme = 'chalk'
      }
    }
  },
  actions: {
  },
  modules: {
  }
})
```

4.3.2.点击切换主题按钮

- 点击事件的响应

```
<div class="title-right">
    <img src="/static/img/qiehuan_dark.png" class="qiehuan"
@click="changeTheme">
    <span class="datetime">2049-01-01 00:00:00</span>
</div>
```

- 点击事件的处理

```
export default {
  methods: {
    changeTheme () {
      this.$store.commit('changeTheme')
    }
  }
}
```

4.3.3.监听主题的变化

以`seller.vue`为例, 进行主题数据变化的监听

- 映射`store`中的`theme`作为当前组件的计算属性

```
<script>
import { mapState } from 'vuex'
export default {
  computed: {
    ...mapState(['theme'])
  }
}
```

- 监听`theme`的变化

```
export default {
  watch: {
    theme () {
      this.chartInstance.dispose()
      this.initChart()
      this.screenAdapter()
      this.updateChart()
    }
  }
}
```

- 主题的切换

  将之前写死的`chalk`改为`this.theme`

```
export default {
  methods: {
    initChart () {
      this.chartInstance = this.$echarts.init(this.$refs.seller_ref,
this.theme)
    }
}
```

  通过这个步骤就可以实现每一个图表组件切换主题了.不过有一些原生`HTML`中的样式需要调整

4.3.4.原生HTML主题样式适配

- 创建`utils/theme_utils.js`文件

  定义两个主题下, 需要进行样式切换的样式数据, 并对外导出一个函数, 用于方便的通过主题名称得到对应主题的某些配置项

```
const theme = {
  chalk: {
    // 背景色
    backgroundColor: '#161522',
    // ScreenPage组件中标题的颜色
    titleColor: '#fff',
    // 页面左上角logo图标
    logoSrc: 'logo_dark.png',
    // 页面顶部头部边框图片
    headerBorderSrc: 'header_border_dark.png',
    // 页面右上角切换按钮的图标
    themeSrc: 'qiehuan_dark.png'
  },
  vintage: {
    backgroundColor: '#eeeeee',
    titleColor: '#000',
    logoSrc: 'logo_light2.png',
    headerBorderSrc: 'header_border_light.png',
    themeSrc: 'qiehuan_light.png'
  }
}

export function getThemeValue (arg) {
  return theme[arg]
}
```

- `Screen.vue`的调整
  - 映射`vuex`中的`theme`数据作为该组件的计算属性

```
import { mapState } from 'vuex'
export default {
  computed: {
    ...mapState(['theme'])
  }
```

  - 定义一些控制样式的计算属性

```
import { mapState } from 'vuex'
import { getThemeValue } from '@/utils/theme_utils'

export default {
  computed: {
    ...mapState(['theme']),
    borderSrc () {
      return '/static/img/' + getThemeValue(this.theme).headerBorderSrc
    },
    logoSrc () {
      return '/static/img/' + getThemeValue(this.theme).logoSrc
    },
    themeSrc () {
      return '/static/img/' + getThemeValue(this.theme).themeSrc
    },
    containerStyle () {
      return {
        backgroundColor: getThemeValue(this.theme).backgroundColor,
```

```
        color: getThemeValue(this.theme).titleColor
      }
    }
  }
}
```

- 将计算属性应用到布局中

```
<template>
  <div class="screen-container" :style="containerStyle">
    <header class="screen-header">
      <div>
        <img :src="borderSrc" alt="">
      </div>
      <span class="logo">
        <img :src="logoSrc" alt="" />
      </span>
      <span class="title">电商平台实时监控系统</span>
      <div class="title-right">
        <img :src="themeSrc" class="qiehuan" @click="changeTheme">
        <span class="datetime">2049-01-01 00:00:00</span>
      </div>
    </header>
  </div>
</template>
```

- `Trend.vue`
  - 修改计算属性`comStyle`和`marginStyle`

```
import { mapState } from 'vuex'
import { getThemeValue } from '@/utils/theme_utils'

export default {
  computed: {
    comStyle () {
      return {
        fontSize: this.titleFontSize + 'px',
        color: getThemeValue(this.theme).titleColor
      }
    },
    marginStyle () {
      return {
        marginLeft: this.titleFontSize + 'px',
        backgroundColor: getThemeValue(this.theme).backgroundColor,
        color: getThemeValue(this.theme).titleColor
      }
    },
    ...mapState(['theme'])
  },
}
```

- `Hot.vue`
  - 修改计算属性`comStyle`

```
import { mapState } from 'vuex'
import { getThemeValue } from '@/utils/theme_utils'

export default {
  computed: {
    comStyle () {
      return {
        fontSize: this.titleFontSize + 'px',
        color: getThemeValue(this.theme).titleColor
      }
    },
    ...mapState(['theme'])
  }
}
```