今日目标

1.完成商品添加
2.完成订单列表
3.完成数据统计展示

1.添加商品

A.完成图片上传

使用upload组件完成图片上传
在element.js中引入upload组件，并注册
因为upload组件进行图片上传的时候并不是使用axios发送请求
所以，我们需要手动为上传图片的请求添加token，即为upload组件添加headers属性

```
//在页面中添加upload组件，并设置对应的事件和属性
<el-tab-pane label="商品图片" name="3">
  <!-- 商品图片上传
  action:指定图片上传api接口
  :on-preview ： 当点击图片时会触发该事件进行预览操作,处理图片预览
  :on-remove ： 当用户点击图片右上角的x号时触发执行
  :on-success：当用户点击上传图片并成功上传时触发
  list-type ： 设置预览图片的方式
  :headers ： 设置上传图片的请求头 -->
  <el-upload :action="uploadURL" :on-preview="handlePreview" :on-
remove="handleRemove" :on-success="handleSuccess" list-type="picture"
:headers="headerObj">
    <el-button size="small" type="primary">点击上传</el-button>
  </el-upload>
</el-tab-pane>
//在el-card卡片视图下面添加对话框用来预览图片
<!-- 预览图片对话框 -->
<el-dialog title="图片预览" :visible.sync="previewVisible" width="50%">
  <img :src="previewPath" class="previewImg" />
</el-dialog>

//在data中添加数据
data(){
  return {
    ......
    //添加商品的表单数据对象
    addForm: {
      goods_name: '',
      goods_price: 0,
      goods_weight: 0,
      goods_number: 0,
      goods_cat: [],
      //上传图片数组
      pics: []
    },
    //上传图片的url地址
    uploadURL: 'http://127.0.0.1:8888/api/private/v1/upload',
    //图片上传组件的headers请求头对象
```

```
    headerObj: { Authorization: window.sessionStorage.getItem('token') },
    //保存预览图片的url地址
    previewPath: '',
    //控制预览图片对话框的显示和隐藏
    previewVisible:false
  }
},
//在methods中添加事件处理函数
methods:{
  .......
  handlePreview(file) {
    //当用户点击图片进行预览时执行，处理图片预览
    //形参file就是用户预览的那个文件
    this.previewPath = file.response.data.url
    //显示预览图片对话框
    this.previewVisible = true
  },
  handleRemove(file) {
    //当用户点击X号删除时执行
    //形参file就是用户点击删除的文件
    //获取用户点击删除的那个图片的临时路径
    const filePath = file.response.data.tmp_path
    //使用findIndex来查找符合条件的索引
    const index = this.addForm.pics.findIndex(item => item.pic === filePath)
    //移除索引对应的图片
    this.addForm.pics.splice(index, 1)
  },
  handleSuccess(response) {
    //当上传成功时触发执行
    //形参response就是上传成功之后服务器返回的结果
    //将服务器返回的临时路径保存到addForm表单的pics数组中
    this.addForm.pics.push({ pic: response.data.tmp_path })
  }
}
```

B.使用富文本插件

想要使用富文本插件vue-quill-editor，就必须先从依赖安装该插件
引入并注册vue-quill-editor，打开main.js，编写如下代码

```
//导入vue-quill-editor（富文本编辑器）
import VueQuillEditor from 'vue-quill-editor'
//导入vue-quill-editor的样式
import 'quill/dist/quill.core.css'
import 'quill/dist/quill.snow.css'
import 'quill/dist/quill.bubble.css'
......
//全局注册组件
Vue.component('tree-table', TreeTable)
//全局注册富文本组件
Vue.use(VueQuillEditor)
```

使用富文本插件vue-quill-editor

```
<!-- 富文本编辑器组件 -->
<el-tab-pane label="商品内容" name="4">
```

```
  <!-- 富文本编辑器组件 -->
  <quill-editor v-model="addForm.goods_introduce"></quill-editor>
  <!-- 添加商品按钮 -->
  <el-button type="primary" class="btnAdd">添加商品</el-button>
</el-tab-pane>

//在数据中添加goods_introduce
//添加商品的表单数据对象
addForm: {
  goods_name: '',
  goods_price: 0,
  goods_weight: 0,
  goods_number: 0,
  goods_cat: [],
  //上传图片数组
  pics: [],
  //商品的详情介绍
  goods_introduce:''
}
//在global.css样式中添加富文本编辑器的最小高度
.ql-editor{
    min-height: 300px;
}
//给添加商品按钮添加间距
.btnAdd{
  margin-top:15px;
}
```

C.添加商品

完成添加商品的操作
在添加商品之前，为了避免goods_cat数组转换字符串之后导致级联选择器报错
我们需要打开vue控制条，点击依赖，安装lodash，把addForm进行深拷贝

```
//打开Add.vue，导入lodash
<script>
//官方推荐将lodash导入为_
import _ from 'lodash'

//给添加商品按钮绑定点击事件
<!-- 添加商品按钮 -->
<el-button type="primary" class="btnAdd" @click="add">添加商品</el-button>
//编写点击事件完成商品添加
add(){
  this.$refs.addFormRef.validate(async valid=>{
    if(!valid) return this.$message.error("请填写必要的表单项!")

    //将addForm进行深拷贝，避免goods_cat数组转换字符串之后导致级联选择器报错
    const form = _.cloneDeep(this.addForm)
    //将goods_cat从数组转换为"1,2,3"字符串形式
    form.goods_cat = form.goods_cat.join(",")
    //处理attrs数组，数组中需要包含商品的动态参数和静态属性
    //将manyTableData（动态参数）处理添加到attrs
    this.manyTableData.forEach(item=>{
      form.attrs.push({ attr_id:item.attr_id, attr_value:item.attr_vals.join("
") })
    })
```

```
    //将onlyTableData（静态属性）处理添加到attrs
    this.onlyTableData.forEach(item=>{
      form.attrs.push({ attr_id:item.attr_id, attr_value:item.attr_vals })
    })

    //发送请求完成商品的添加,商品名称必须是唯一的
    const {data:res} = await this.$http.post('goods',form)
    if(res.meta.status !== 201){
      return this.$message.error('添加商品失败')
    }
    this.$message.success('添加商品成功')
    //编程式导航跳转到商品列表
    this.$router.push('/goods')
  })
}
</script>
```

D.推送代码

推送goods_list分支到码云
将代码添加到暂存区： git add .
将代码提交到本地仓库： git commit -m "完成商品功能开发"
将代码推送到码云： git push
切换到master主分支： git checkout master
将goods_list分支代码合并到master: git merge goods_list
将master推送到码云： git push

2.订单列表

A.创建分支

创建order子分支并推送到码云
创建order子分支: git checkout -b order
将order分支推送到码云： git push -u origin order

B.创建路由

创建订单列表路由组件并添加路由规则

```
//在components中新建order文件夹，新建Order.vue组件，组件中添加代码如下
<template>
    <div>
        <h3>订单列表</h3>
        <!-- 面包屑导航 -->
        <el-breadcrumb separator="/">
            <el-breadcrumb-item :to="{ path: '/home' }">首页</el-breadcrumb-item>
            <el-breadcrumb-item>订单管理</el-breadcrumb-item>
            <el-breadcrumb-item>订单列表</el-breadcrumb-item>
        </el-breadcrumb>
        <!-- 卡片视图区域 -->
        <el-card>
            <!-- 搜索栏 -->
            <el-row :gutter="20">
                <el-col :span="8">
                    <el-input placeholder="请输入内容" v-model="queryInfo.query"
clearable>
```

```
                        <el-button slot="append" icon="el-icon-search" ></el-
button>
                    </el-input>
                </el-col>
            </el-row>
        </el-card>
    </div>
</template>

<script>
export default {
  data() {
    return {
        //查询条件
        queryInfo:{
            query:'',
            pagenum:1,
            pagesize:10
        }
    }
  },
  created() {

  },
  methods: {

  }
}
</script>

<style lang="less" scoped>

</style>

//打开router.js导入Order.vue并添加规则
import Order from './components/order/Order.vue'

path: '/home', component: Home, redirect: '/welcome', children: [
  { path: "/welcome", component: Welcome },
  { path: "/users", component: Users },
  { path: "/rights", component: Rights },
  { path: "/roles", component: Roles  },
  { path: "/categories", component: Cate  },
  { path: "/params", component: Params  },
  { path: "/goods", component: GoodList  },
  { path: "/goods/add", component: GoodAdd  },
  { path: "/orders", component: Order  }
]
```

C.实现数据展示及分页

```
<!-- 卡片视图区域 -->
<el-card>
    <!-- 搜索栏 -->
    <el-row :gutter="20">
        <el-col :span="8">
```

```
            <el-input placeholder="请输入内容" v-model="queryInfo.query"
clearable>
                <el-button slot="append" icon="el-icon-search"></el-button>
            </el-input>
        </el-col>
    </el-row>

    <!-- 订单表格 -->
    <el-table :data="orderList" border stripe>
        <el-table-column type="index"></el-table-column>
        <el-table-column label="订单编号" prop="order_number"></el-table-column>
        <el-table-column label="订单价格" prop="order_price"></el-table-column>
        <el-table-column label="是否付款" prop="pay_status">
            <template slot-scope="scope">
                <el-tag type="success" v-if="scope.row.pay_status === '1'">已付款
</el-tag>
                <el-tag type="danger" v-else>未付款</el-tag>
            </template>
        </el-table-column>
        <el-table-column label="是否发货" prop="is_send"></el-table-column>
        <el-table-column label="下单时间" prop="create_time">
            <template slot-scope="scope">
                {{scope.row.create_time | dateFormat}}
            </template>
        </el-table-column>
        <el-table-column label="操作" width="125px">
            <template slot-scope="scope">
                <el-button size="mini" type="primary" icon="el-icon-edit"></el-
button>
                <el-button size="mini" type="success" icon="el-icon-location">
</el-button>
            </template>
        </el-table-column>
    </el-table>

    <!-- 分页 -->
    <el-pagination @size-change="handleSizeChange" @current-
change="handleCurrentChange" :current-page="queryInfo.pagenum" :page-sizes="[3,
5, 10, 15]" :page-size="queryInfo.pagesize" layout="total, sizes, prev, pager,
next, jumper" :total="total">
    </el-pagination>
</el-card>

<script>
export default {
  data() {
    return {
      //查询条件
      queryInfo: {
        query: '',
        pagenum: 1,
        pagesize: 10
      },
      //订单列表数据
      orderList: [],
      //数据总条数
      total: 0
    }
```

```
  },
  created() {
    this.getOrderList()
  },
  methods: {
    async getOrderList() {
      const { data: res } = await this.$http.get('orders', {
        params: this.queryInfo
      })

      if (res.meta.status !== 200) {
        return this.$message.error('获取订单列表数据失败!')
      }

      this.total = res.data.total
      this.orderList = res.data.goods
    },
    handleSizeChange(newSize){
        this.queryInfo.pagesize = newSize
        this.getOrderList()
    },
    handleCurrentChange(newPage){
        this.queryInfo.pagenum = newPage
        this.getOrderList()
    }
  }
}
</script>
```

D.制作省市区县联动

打开今天的资料，找到素材文件夹，复制citydata.js文件到components/order文件夹中
然后导入citydata.js文件

```
<script>
  import cityData from "./citydata.js"
</script>
```

具体代码如下：

```
//给修改地址按钮添加点击事件
<el-button size="mini" type="primary" icon="el-icon-edit"
@click="showEditAddress"></el-button>
//添加修改地址对话框,在卡片视图下方添加
<!-- 修改地址对话框 -->
<el-dialog title="修改收货地址" :visible.sync="addressVisible" width="50%"
@close="addressDialogClosed">
    <!-- 添加表单 -->
    <el-form :model="addressForm" :rules="addressFormRules" ref="addressFormRef"
label-width="100px">
        <el-form-item label="省市区县" prop="address1">
            <el-cascader :options="cityData" v-model="addressForm.address1">
</el-cascader>
        </el-form-item>
        <el-form-item label="详细地址" prop="address2">
            <el-input v-model="addressForm.address2"></el-input>
```

```
        </el-form-item>
    </el-form>
    <span slot="footer" class="dialog-footer">
        <el-button @click="addressVisible = false">取 消</el-button>
        <el-button type="primary" @click="addressVisible = false">确 定</el-
button>
    </span>
</el-dialog>

//js部分的代码
<script>
import cityData from "./citydata.js"
export default {
  data() {
    return {
      ......
      //控制修改地址对话框的显示和隐藏
      addressVisible:false,
      //修改收货地址的表单
      addressForm:{
          address1:[],
          address2:''
      },
      addressFormRules:{
          address1:[{ required: true, message: '请选择省市区县', trigger: 'blur'
}],
          address2:[{ required: true, message: '请输入详细地址', trigger: 'blur'
}],
      },
      //将导入的cityData数据保存起来
      cityData:cityData
      }
  },methods: {
    ......
    showEditAddress() {
      //当用户点击修改收货地址按钮时触发
      this.addressVisible = true;
    },
    addressDialogClosed(){
        this.$refs.addressFormRef.resetFields()
    }
  }
}
</script>
<style lang="less" scoped>
.el-cascader{
    width: 100%;
}
</style>
```

E.制作物流进度对话框

因为我们使用的是element-ui中提供的Timeline组件，所以需要导入并注册组件
打开element.js,编写代码会进行导入和注册

```
import {
    Timeline,TimelineItem
} from 'element-ui'

Vue.use(Timeline)
Vue.use(TimelineItem)
```

打开Order.vue文件，添加代码实现物流进度对话框

```
<!-- 物流信息进度对话框 -->
<el-dialog title="物流进度" :visible.sync="progressVisible" width="50%">
    <!-- 时间线组件  -->
    <el-timeline>
        <el-timeline-item v-for="(activity, index) in progressInfo"
        :key="index" :timestamp="activity.time">
            {{activity.context}}
        </el-timeline-item>
    </el-timeline>
</el-dialog>

<script>
import cityData from './citydata.js'
export default {
  data() {
    return {
      ......
      //控制物流进度对话框的显示和隐藏
      progressVisible: false,
      //保存物流信息
      progressInfo: []
      }
  },methods: {
    ......
    async showProgress() {
      //发送请求获取物流数据
      const { data: res } = await this.$http.get('/kuaidi/804909574412544580')

      if (res.meta.status !== 200) {
        return this.$message.error('获取物流进度失败!')
      }
      this.progressInfo = res.data
      //显示对话框
      this.progressVisible = true
    }
  }
}
</script>
```

F.推送代码

将order分支代码推送至码云
将代码添加到暂存区： git add .
将代码提交到本地仓库： git commit -m "完成订单列表功能开发"
将代码推送到码云： git push
切换到master主分支： git checkout master

将goods_list分支代码合并到master: git merge order
将master推送到码云： git push

3.数据统计

A.创建子分支

创建report子分支并推送到码云
创建report子分支: git checkout -b report
将report分支推送到码云： git push -u origin report

B.创建路由

创建数据统计路由组件并添加路由规则

```
//在components中新建report文件夹，新建Report.vue组件，组件中添加代码如下
<template>
    <div>
        <h3>数据报表</h3>
        <!-- 面包屑导航 -->
        <el-breadcrumb separator="/">
            <el-breadcrumb-item :to="{ path: '/home' }">首页</el-breadcrumb-item>
            <el-breadcrumb-item>数据统计</el-breadcrumb-item>
            <el-breadcrumb-item>数据报表</el-breadcrumb-item>
        </el-breadcrumb>
        <!-- 卡片视图区域 -->
        <el-card></el-card>
    </div>
</template>

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

    }
  },created(){

  },methods:{

  }
}
</script>

<style lang="less" scoped>

</style>
```

打开router.js，导入Report.vue并设置路由规则

```
import Report from './components/report/Report.vue'
path: '/home', component: Home, redirect: '/welcome', children: [
  { path: "/welcome", component: Welcome },
  { path: "/users", component: Users },
  { path: "/rights", component: Rights },
  { path: "/roles", component: Roles  },
  { path: "/categories", component: Cate  },
  { path: "/params", component: Params  },
  { path: "/goods", component: GoodList  },
  { path: "/goods/add", component: GoodAdd  },
  { path: "/orders", component: Order  },
  { path: "/reports", component: Report  }
]
```

C.导入ECharts并使用

```
<template>
    <div>
        <h3>数据报表</h3>
        <!-- 面包屑导航 -->
        <el-breadcrumb separator="/">
            <el-breadcrumb-item :to="{ path: '/home' }">首页</el-breadcrumb-item>
            <el-breadcrumb-item>数据统计</el-breadcrumb-item>
            <el-breadcrumb-item>数据报表</el-breadcrumb-item>
        </el-breadcrumb>
        <!-- 卡片视图区域 -->
        <el-card>
            <div id="main" style="width:750px;height:400px;"></div>
        </el-card>
    </div>
</template>

<script>
//导入echarts
import echarts from 'echarts'
//导入lodash
import _ from 'lodash'
export default {
  data() {
    return {
      //需要跟请求的折线图数据合并的options
      options: {
        title: {
          text: '用户来源'
        },
        tooltip: {
          trigger: 'axis',
          axisPointer: {
            type: 'cross',
            label: {
              backgroundColor: '#E9EEF3'
            }
          }
        },
        grid: {
          left: '3%',
```

```
          right: '4%',
          bottom: '3%',
          containLabel: true
        },
        xAxis: [
          {
            boundaryGap: false
          }
        ],
        yAxis: [
          {
            type: 'value'
          }
        ]
      }
    }
  },
  created() {},
  async mounted() {
    //在页面dom元素加载完毕之后执行的钩子函数mounted
    // 基于准备好的dom，初始化echarts实例
    var myChart = echarts.init(document.getElementById('main'))
    //准备数据和配置项
    //发送请求获取折线图数据
    const { data: res } = await this.$http.get('reports/type/1')

    if (res.meta.status !== 200) {
      return this.$message.error('获取折线图数据失败')
    }

    //合并res.data和this.options
    const result = _.merge(res.data,this.options)

    // 使用获取的数据展示图表
    myChart.setOption(result)
  },
  methods: {}
}
</script>

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

D.推送代码

推送report分支到码云
将代码添加到暂存区： git add .
将代码提交到本地仓库： git commit -m "完成数据报表功能开发"
将代码推送到码云： git push
切换到master主分支： git checkout master
将report分支代码合并到master: git merge report
将master推送到码云： git push