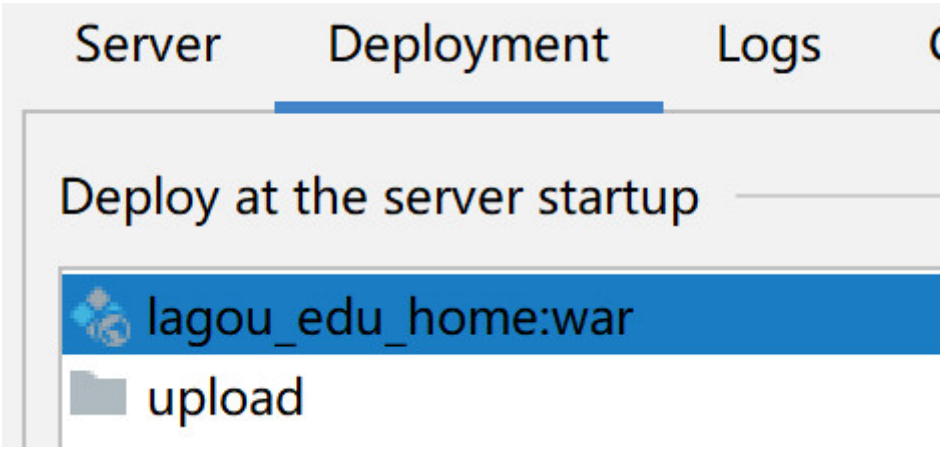
任务三 前后端项目接口联调

1.联调准备

1.1 运行后台项目

- **clean** 清空项目的编译文件
- **compile** 重新编译项目
- 将项目部署到 tomcat
  - 项目名为: **lagou_edu_home**
  - 端口号: **8080**
- 部署图片上传路径为 webapps目录下的 upload目录



1.2 运行前端项目

1. 首先导入前端项目到 VSCode
2. 运行项目.

2.课程管理首页

2.1 获取课程列表

JS部分

```
export default {
  name: "Courses",
  title: "课程管理",
  //定义数据部分
  data() {
    return {
      filter: { course_name: "", status: "" }, //查询对象
      courses: [], //课程信息集合
      loading: false //是否弹出加载
    };
  },
```

```
  //钩子函数
  created() {
    this.loadCourses();
  },

  methods: {
    //方法1: 获取课程列表
    loadCourses() {
      this.loading = true;

      //请求后台查询课程列表接口
      return axios
        .get("/course", {
          params: {
            methodName: "findCourseList"
          }
        })
        .then(resp => {
          console.log(resp);
          this.loading = false; //关闭加载
          this.courses = resp.data; //取出数据
        })
        .catch(error => {
          this.$message.error("数据获取失败! ! !");
        });
    },
  }
};
</script>
```

2.2 条件查询课程信息

JS部分

```
    //方法2: 条件查询课程信息
    filterQuery() {
      this.loading = true;

      //定义查询条件对象
      const search = {};

      //保存用户输入的条件
      if (this.filter.course_name) search.course_name = this.filter.course_name;
      if (this.filter.status) search.status = this.filter.status;

      //请求后台条件查询接口
      return axios
        .get("/course", {
          //准备参数
          params: {
            methodName: "findByCourseNameAndStatus",
            course_name: search.course_name,
            status: search.status
          }
        })
        .then(resp => {
          console.log(resp);
```

```
          this.loading = false;
          //将响应数据保存到 courses
          this.courses = resp.data;
        })
        .catch(error => {
          this.$message.error("数据获取失败！ ！ !");
        });
    },
```

2.3 跳转到添加课程

JS部分

```
    //方法3: 添加课程跳转方法
    addCourse() {
      //路由跳转到 CourseItem.vue组件
      this.$router.push({ name: "CourseItem", params: { courseId: "new" } });
    },
```

2.4 修改课程状态

JS部分

```
    //方法4: 修改课程状态
    updateStatus(item) {
      //item 表示选中的数据 = Course对象
      axios
        .get("/course", {
          params: {
            methodName: "updateCourseStatus",
            id: item.id
          }
        })
        .then(res => {
          console.log(res);
          //将返回的状态字段,封装到对象中
          Object.assign(item, res.data);

          //重新加载页面
          window.location.reload;
        });
    },
```

2.5 跳转课程营销或内容管理

JS部分

```
    //方法5: 根据路由名称, 导航到对应组件
    handleNavigate(name, id) {
      this.$router.push({ name, params: { courseId: id } });
    },
```

3.新建&修改课程

3.1 Course组件中的跳转方法

```
<!-- 营销信息按钮 -->
<el-button
     size="mini"
     @click="handleNavigate('CourseItem', scope.row.id)"
     >营销信息</el-button>
```

```
    //方法3: 添加课程跳转方法
    addCourse() {
      //路由跳转到 CourseItem.vue组件
      this.$router.push({ name: "CourseItem", params: { courseId: "new" } });
    },
```

3.2 router.js 路由

找到name为: CourseItem的路由

```
//添加课程的路由
{
    path: "/courses/:courseId", //路径,携带参数: 课程ID
    name: "CourseItem",
    //路由导航到的组件
    component: () =>
    import(/* webpackChunkName: 'courses' */ "../views/CourseItem.vue")
},
```

3.3 CourseItem组件

3.3.1 JS部分

```
  data() {
    //数据
    return {
      rules, //规则
      course: {}, //课程
      loading: false,
      params: {} //参数对象
    };
  },
  //钩子函数
  created() {

    //1.获取路由传递的参数
    const id = this.$route.params.courseId;

    //2.判断id是否有值,没有值跳转到 错误页面
    if (!id) return this.redirectToError();

    //3.判断 是new 还是具体的id
    if (id === "new") {
      //new 代表新增,添加一个新增课程标题
```

```
      this.course.title = "新增课程";
    } else {
      //否则就是修改,调用loadCourse方法进行回显
      this.loadCourse(id);
    }
  },
```

3.3.2 图片上传分析

页面部分

```
<!-- 上传图片部分 -->
<el-form-item label="分享小图" prop="share_image_title">
    <el-input v-model="course.share_image_title" type="text">
        <!-- :auto-upload="false",取消自动上传, :on-change="onchange" 调用onchange
进行处理 -->
        <el-upload
                    slot="prepend"
                    :auto-upload="false"
                    :on-change="onchange"
                    action
                    :limit="1"
                    >
            <el-button size="small" type="primary">点击上传</el-button>
        </el-upload>
    </el-input>
</el-form-item>
```

JS部分

FormData的主要用途有两个

1. 将form表单元素的name与value进行组合，实现表单数据的序列化，从而减少表单元素的拼接，提高工作效率。
2. 异步上传文件
3. 创建FormData对象

```
//通过FormData构造函数创建一个空对象
var formdata=new FormData();

//可以通过append()方法来追加数据
formdata.append("name","laotie");

//通过get方法对值进行读取
console.log(formdata.get("name"));//laotie

//通过set方法对值进行设置
formdata.set("name","laoliu");
console.log(formdata.get("name"));//laoliu
```

```
data中创建FormData
    //5.创建FormData对象,将图片与表单一同上传
    this.params = new FormData();

methods添加方法
    //文件上传
    onchange(file) {
      //判断文件不为空
      if (file != null) {
        //将文件信息 保存到params中
        this.params.append("file", file.raw, file.name);
      }
    },
```

3.3.3 新建课程信息

JS 部分

```
    //保存和修改课程信息
    handleSave() {
      //检查是否拿到了正确的需要验证的form
      this.$refs.form.validate(valid => {
        if (!valid) return false;

        //1.设置Content-Type为 多部件上传
        let config = {
          headers: {
            "Content-Type": "multipart/form-data"
          }
        };

        //2.获取表单中的数据,保存到params (params 就是 FromData对象)
        for (let key in this.course) {
          //debugger
          console.log(key + "---" + this.course[key]);
          this.params.append(key, this.course[key]);
        }

        //3.保存课程信息
        axios
          .post("/courseSalesInfo", this.params, config)
          .then(res => {
            //debugger
            if (res.data.status == 0) {
              //保存成功,跳转到首页
              this.$router.back();
            } else if (res.data.status == 1) {
              this.$message({
                type: "error",
                message: res.data.msg
              });
            }
          })
          .catch(err => {
            this.$message.error("保存失败! ! !");
          });
      });
```

```
    },
```

3.3.4 修改课程信息

```
    //方法2: 根据ID 回显课程信
    loadCourse(id) {
      this.loading = true;
      axios
        .get("/course", {
          params: {
            methodName: "findCourseById",
            id: id
          }
        })
        .then(res => {
          this.loading = false;
          this.course = res.data;
        })
        .catch(() => {
          this.$message.error("回显数据失败! ! !");
        });
    },
```

4.内容管理

4.1 Course组件中的跳转方法

```
<!-- 内容管理按钮 -->
<el-button
      size="mini"
      @click="handleNavigate('CourseTasks', scope.row.id)">内容管理</el-button>
```

```
//方法5: 根据路由名称, 导航到对应组件
handleNavigate(name, id) {
    this.$router.push({ name, params: { courseId: id } });
},
```

4.2 router.js 路由

```
//内容管理的路由
{
    path: "/courses/:courseId/tasks",
    name: "CourseTasks",
    meta: { requireAuth: true },
    component: () =>
    import(/* webpackChunkName: 'courses' */ "../views/CourseTasks.vue")
}
```

4.3 CourseTasks组件

4.3.1 树形控件测试

1. 打开我们之前编写的VueCLI项目
2. 在component目录下,添加一个组件, **TestTree.vue**

```
<template>

</template>

<script>
export default {

}
</script>

<style scoped>
</style>
```

3. 在Index.vue组件中的导航菜单位置添加一个 树形控件导航.

注意:要设置index的路径为 /tree

```
<el-submenu index="1">
    <template slot="title">
        <i class="el-icon-location"></i>
        <span>导航菜单</span>
    </template>
    <el-menu-item-group>
        <!-- 修改 index的路由地址 -->
        <el-menu-item index="/course">
            <i class="el-icon-menu"></i>课程管理
        </el-menu-item>

        <el-menu-item index="/tree">
            <i class="el-icon-menu"></i>树形控件
        </el-menu-item>
    </el-menu-item-group>
</el-submenu>
```

4. 在index.js 路由文件中进行配置,在布局路由中再添加一个子路由

```
//导入树形控件组件
import TestTree from "@/components/TestTree";

//布局路由
  {
    path: "/index",
    name: "index",
    component: Index,
    //添加子路由,使用 children属性 来表示子路由
    children: [
      //课程信息子路由
      {
        path: "/course",
        name: "course",
        component: Course,
```

```
      },
      //Tree控件测试路由
      {
        path: "/tree",
        name: "tree",
        component: TestTree,
      },
    ],
  },
```

5. 在ElementUI官网中查找树形控件

Tree 树形控件

用清晰的层级结构展示信息，可展开或折叠。

6. 先来查看基础用法,赋值代码到 TestTree.vue

```
<template>
  <el-tree :data="data" :props="defaultProps" ></el-tree>
</template>

<script>
export default {
  data() {
    return {
      data: [
        {
          label: "一级 1",
          children: [
            {
              label: "二级 1-1",
              children: [
                {
                  label: "三级 1-1-1"
                }
              ]
            }
          ]
        },
        {
          label: "一级 2",
          children: [
            {
              label: "二级 2-1",
              children: [
                {
                  label: "三级 2-1-1"
                }
              ]
            },
            {
              label: "二级 2-2",
              children: [
                {
                  label: "三级 2-2-1"
```

```
                }
              ]
            }
          ]
        },
        {
          label: "一级 3",
          children: [
            {
              label: "二级 3-1",
              children: [
                {
                  label: "三级 3-1-1"
                }
              ]
            },
            {
              label: "二级 3-2",
              children: [
                {
                  label: "三级 3-2-1"
                }
              ]
            }
          ]
        }
      ],
      defaultProps: {
        children: "children",
        label: "label"
      }
    };
  }
};
</script>
```

7. Tree组件属性分析

- **data**: 展示数据
- **props** : 配置树形结构
  - label : 设置节点名称
  - children: 指定生成子树的属性名称

8. 自定义树节点内容:

- data: 数据对象
- node: 节点对象

```
  <el-tree :data="data" :props="defaultProps">
    <span class="custom-tree-node" slot-scope="{ node, data }">
      <span>{{ data.label}}</span>
      <span>级别: {{node.level}}</span>
    </span>
  </el-tree>
```

9. 展示树形结构章节与课时

```
<template>
  <el-tree :data="data" :props="defaultProps">
    <span class="custom-tree-node" slot-scope="{ node, data }">
      <span>{{ data.section_name || data.theme}}</span>
      <span>级别: {{node.level}}</span>
    </span>
  </el-tree>
</template>

<script>
export default {
  data() {
    return {
      data: [
        {
          id: 5,
          course_id: 10,
          section_name: "麻式太极",
          description: "麻式太极拳,你动我试试",
          orderNum: 0,
          status: 2,
          create_time: "2019-07-11 10:55:10.0",
          update_time: "2019-10-09 12:43:01.0",
          isDel: 0,
          lessonList: [
            {
              id: 32,
              course_id: 10,
              section_id: 5,
              theme: "第一讲:如何给自己洗脑",
              duration: 10,
              is_free: 1,
              order_num: 1,
              status: 2,
              create_time: "2019-01-23 20:37:02.0",
              update_time: "2020-02-24 18:37:34.0",
              isDel: 0
            },
            {
              id: 33,
              course_id: 10,
              section_id: 5,
              theme: "第二讲:如何给别人洗脑",
              duration: 10,
              is_free: 1,
              order_num: 1,
              status: 2,
              create_time: "2019-01-23 20:37:02.0",
              update_time: "2020-02-24 18:37:34.0",
              isDel: 0
            }
          ]
        }
      ],
      defaultProps: {
        children: "lessonList",
        label: item => {
```

```
            return item.section_name || item.theme;
          }
        }
      };
    }
};
</script>
```

4.3.2 显示当前课程的名称

1. 显示当前课程名称

```
<el-page-header
        @back="() => this.$router.back()"
        :content="addSectionForm.course_name"
      />
```

2. data数据

```
data() {
    //定义章节信息
    const addSectionForm = {
      course_id: undefined,
      course_name: "",
      section_name: "",
      description: "",
      order_num: 0
    };

    //章节与课时信息,树形结构
    const treeProps = {
      label: item => {
        return item.section_name || item.theme;
      },
      children: "lessonList"
    };

    //定义章节状态信息
    const statusMapping = {
      0: "已隐藏",
      1: "待更新",
      2: "已更新"
    };

    const statusForm = {
      status: 0
    };

    return {
      addSectionForm,
      treeProps,
      sections: [],
      statusForm, //状态表单
      statusMapping,
```

```
      loading: false, //树形控件
      showAddSection: false, //添加或修改章节
      showStatusForm: false //状态修改
    };
  },
```

3. 加载课程信息

```
created() {
    //1.显示当前页面在网站中的位置
    this.$breadcrumbs = [
      { name: "Courses", text: "课程管理" },
      { text: "课程结构" }
    ];

    //2. 从路由中获取传递的参数, 课程id
    const id = this.$route.params.courseId;
    if (!id) return this.redirectToError();

    //3.加载课程信息
    this.loadCourse(id);

    //4.加载课程对应的章节与课时
    this.loadChildren(id);
  },

  methods: {
    //方法1: 加载课程信息
    loadCourse(id) {
      axios
        .get("/courseContent", {
          params: {
            methodName: "findCourseById",
            course_id: id
          }
        })
        .then(res => {
          console.log(res.data);
          //将数据保存到表单对象中
          this.addSectionForm.course_id = res.data.id;
          this.addSectionForm.course_name = res.data.course_name;
        })
        .catch(error => {
          this.$message.error("数据获取失败! ! !");
        });
    },
```

4.3.3 加载章节与课时信息

JS部分

```
    //方法2: 加载树(章节与课程)
    loadChildren(id) {
      this.loading = true;
      axios
        .get("/courseContent", {
```

```
          params: {
            methodName: "findSectionAndLessonByCourseId",
            course_id: id
          }
        })
        .then(resp => {
          //获取章节数据,保存到sections
          this.sections = resp.data;
          this.loading = false;
        })
        .catch(error => {
          this.loading = false;
          this.$message.error("数据获取失败! ! !");
        });
    },
```

HTML

- **el-tree** ElementUI树形控件
  - **:data** 列表数据
  - **:props** 配置选项

```
 <!-- 树形控件,展示课程对应的章节信息 -->
    <el-tree
      :data="sections"
      :props="treeProps"
      v-loading="loading"
      element-loading-text="数据加载中..."
    >
      <!-- slot-scope:代表当前树节点内容,有两个参数 data表示当前树节点, node表示当前节点状态 -->
      <div class="inner" slot-scope="{ data, node }">
        <span>{{ data.section_name || data.theme }}</span>
```

- **:props** 配置选项
  - **label** : 指定节点标签为节点对象的某个属性值
  - **children** : 指定子树为节点对象的某个属性值

```
    //章节与课时信息,树形结构
    const treeProps = {
      label: item => {
        return item.section_name || item.theme;
      },
      children: "lessonList"
    };

    //children: "lessonList" 就是返回的JSON数据中的lessonList集合
```

- 操作按钮显示
  - node.level 获取当前节点的级别
  - @click.stop 事件冒泡,点击哪个元素,就执行哪个元素绑定的事件

```

```

```
<span class="actions">
    <!-- 编辑章节  @click.stop 阻止事件冒泡 -->
    <el-button
               v-if="node.level == 1"
               size="small"
               @click.stop="handleEditSection(data)"
               >编辑</el-button
        >

    <!-- 修改章节状态 -->
    <el-button
               v-if="node.level == 1"
               size="small"
               @click.stop="handleShowToggleStatus(data)"
               >{{ statusMapping[data.status] }}</el-button
        >
</span>
```

4.3.4 回显信息

HTML

```
<el-button type="primary" icon="el-icon-plus" @click="handleShowAddSection">添加章节</el-button>
```

JS 部分

```
    //方法3: 显示添加章节表单,回显课程信息
    handleShowAddSection() {
      //显示表单
      this.showAddSection = true;
    },
```

4.3.5 添加章节

HTML

```
<el-button type="primary" @click="handleAddSection">确 定</el-button>
```

JS

```
    //方法4: 添加&修改章节操作
    handleAddSection() {
      axios
        .post("/courseContent", {
          methodName: "saveOrUpdateSection",
          section: this.addSectionForm
        })
        .then(resp => {
          //debugger;
          this.showAddSection = false;
          //重新加载列表
          return this.loadChildren(this.addSectionForm.course_id);
        })
        .then(() => {
```

```
          //reset 重置表单
          this.addSectionForm.section_name = "";
          this.addSectionForm.description = "";
          this.addSectionForm.order_num = 0;
        })
        .catch(error => {
          this.showAddSection = false;
          this.$message.error("操作执行失败! ! !");
        });
    },
```

4.3.6 后台接口问题解决

- BaseServlet中代码修改

```
if("application/json;charset=utf-8".equals(contentType))

修改为:
if("application/json;charset=utf-8".equalsIgnoreCase(contentType))
```

- CourseContentServlet中的saveOrUpdateSection方法修改

```
 //3.使用BeanUtils的 copyProperties方法,将map中的数据封装到section对象里
 BeanUtils.copyProperties(section,map.get("section"));
```

4.3.7 修改章节

HTML

```
<el-button v-if="node.level == 1" size="small"
@click.stop="handleEditSection(data)">编辑</el-button>
```

JS 回显示方法

```
  //方法5: 修改章节回显方法
    handleEditSection(section) {
      //对象拷贝
      Object.assign(this.addSectionForm, section);
      this.showAddSection = true;
    },
```

- 事件冒泡:当点击子元素的事件。如果父元素也有同样的事件的话。他就会一并的触发。
- 解决冒泡事件的方法: @click.stop

```
  <body>
    <!-- 事件冒泡: 解决方案 @click.stop -->
    <div id="app" @click="log('div点击事件')">
      <button @click="log('button点击事件')">事件冒泡</button>
    </div>
  </body>
  <script src="./js/vue.min.js"></script>
  <script>
    var VM = new Vue({
```

```
      el: "#app",
      methods: {
        log(t) {
          alert(t);
        },
      },
    });
  </script>
```

4.3.8 章节状态回显

HTML

```
 <!-- 修改章节状态 -->
 <el-button
 v-if="node.level == 1"
 size="small"
 @click.stop="showStatus(data)"
 >{{ statusMapping[data.status] }}</el-button
 >
```

JS

```
//data函数中定义的章节状态
const statusMapping = {
    0: "已隐藏",
    1: "待更新",
    2: "已更新"
};
```

```
//方法6: 章节状态修改
showStatus(data) {
    this.statusForm.id = data.id;
    this.statusForm.status = data.status.toString();
    this.statusForm.data = data;
    this.showStatusForm = true;//显示状态表单
},
```

4.3.9 Select选择器

1. 打开我们之前编写的VueCLI项目
2. 在component目录下,添加一个组件, **TestSelect.vue**

```
<template></template>

<script>
    export default {};
</script>

<style scoped>
</style>
```

3. 在Index.vue组件中的导航菜单位置添加一个 Select选择器 导航.

注意:要设置index的路径为 /select

```
<el-menu-item index="/select">
   <i class="el-icon-menu"></i>Select选择器
</el-menu-item>
```

4. 在index.js 路由文件中进行配置,在布局路由中再添加一个子路由

```
{
    path: "/select",
    name: "select",
    component: Select,
},
```

5. 在ElementUI官网中查找Select选择器

6. 查看基础用法,将代码复制到 TestSelect.vue中

```
<template>
  <el-select v-model="value" placeholder="请选择">
    <el-option v-for="item in options" :key="item.value" :label="item.label"
:value="item.value"></el-option>
  </el-select>
</template>

<script>
export default {
  data() {
    return {
      options: [
        {
          value: "选项1",
          label: "黄金糕"
        },
        {
          value: "选项2",
          label: "双皮奶"
        },
        {
          value: "选项3",
          label: "蚵仔煎"
        },
        {
          value: "选项4",
          label: "龙须面"
        },
        {
          value: "选项5",
          label: "北京烤鸭"
        }
      ],
```

```
      value: ""
    };
  }
};
</script>
```

7. select 选择器属性分析

- **v-model**: 的值为当前被选中的`el-option`的 value 属性值
- **el-option** : 选项
  - label 选项的标签名
  - value 选择的值

8. 使用Select选择器展示状态信息

```
<template>
  <el-select v-model="status" placeholder="请选择">
    <el-option
      v-for="index in Object.keys(statusMappings)"
      :key="index"
      :label="statusMappings[index]"
      :value="index"
    ></el-option>
  </el-select>
</template>

<script>
export default {
  data() {
    return {
      statusMappings: {
        0: "已隐藏",
        1: "待更新",
        2: "已更新"
      },
      status:0
    };
  }
};
</script>
```

- **Object.keys()**

```
var obj = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.keys(obj)); // console: ['0', '1', '2']
```

4.4.0 章节状态修改

HTML

```
<el-button type="primary" @click="updateStatus">确 定</el-button>
```

JS部分

```
    //方法7: 修改章节状态
    updateStatus(statusForm) {
      axios
        .get("/courseContent", {
          params: {
            methodName: "updateSectionStatus",
            status: this.statusForm.status,
            id: this.statusForm.id
          }
        })
        .then(resp => {
          this.statusForm.data.status = this.statusForm.status;
          this.statusForm = {};
          this.showStatusForm = false;
        })
        .catch(error => {
          this.showStatusForm = false;
          this.$message.error("修改状态失败! ! !");
        });
    },
```

- v-for里面数据层次太多， 修改过数据变了，页面没有重新渲染，需手动强制刷新。

```
@change="$forceUpdate()" 强制刷新
```