项目报告

姓名：洪宇
学号：2022080901022

说明

本项目为搭建一个小型部门管理系统，后端使用Spring Boot + JPA，前端使用Vue及Element UI的库实现，数据库存储则是MySQL，主要用于学习前后端分离以及跨域请求等内容，为简易网盘项目前后端代码的编写作出铺垫。

本实验报告以读者能复现为目标，详细描述项目开发的过程，并且加入了遇到的问题及解决方案。

部门管理系统

环境准备

IDE使用IntelliJ IDEA 2022.3.1 (Ultimate Edition)，Spring Boot版本为3.1.0，vue为2.7.14，vue-cli为5.0.8，随后使用的Element UI为2.15.13，ZooKeeper版本为3.7.1，MySQL在官网选择community downloads中的server即可，为了使用方便直观可以下载图形用户界面workbench。Spring Boot2与3在本实验中差别不大，Vue3与2有较大差别因而注意不要选3。接着安装开发更简单快速的vue-cli：

先进入终端，输入`npm -v`和`node -v`查看是否安装了npm和node.js，若电脑有预先安装则会出现版本号，没有则去官网下载即可，由于mac自带有npm故不再赘述。

为了加快npm的安装各种包的速度，执行以下命令切换为淘宝镜像源：
`npm config set registry https://registry.npm.taobao.org`
随后可输入`npm config get registry`查看是否切换成功。

接着更新npm到较新版本并安装vue-cli：
`sudo npm install -g npm@9.6.6`
`sudo npm install -g @vue/cli`

项目初始化

前端

随后进入Vue项目所要存放的文件夹，在终端执行`vue init webpack my-project`，其中my-project是自己取定的项目名称。建议不要在需要管理员权限的文件夹（如根目录）直接执行，否则在

之后的命令执行中都会出现权限问题，相对而言比较麻烦，自己新建一个文件夹即可。在创建webpack时会询问如项目名称、是否安装单元测试等，单元测试在本实验中没有用到，ESLint则可以选择，router是需要的，其他选择在询问信息中推荐的即可。

随后执行`npm install`安装项目需要的一些基础模块等，接着便可执行`npm run dev`启动前端查看是否正常，默认是运行在 http://localhost:8080。此时页面上会出现Vue的标志和一些超链接等，ctrl+c可以终止运行。

后端

使用Spring Initializr来创建Spring Boot应用。由于创建项目所使用的默认服务器很慢（即Spring Boot官网），因而把URL换为官网对应的国内镜像地址 https://start.springboot.io，各名称没有要求，jdk选择较新的即可，注意类型选择Maven。随后会提示选择依赖项，Spring Boot项目所用到的包的依赖会在项目中的pom.xml文件中给出，因而这一步实际上是快速生成一个写入了依赖项的pom.xml。本项目用到Lombok、Spring Web、Spring Data JPA及MySQL Driver，勾选后创建即可。

数据库

直接在MySQL Workbench中创建schema（数据库）即可，也可以在命令行输入`CREATE DATABASE myProject`创建，这里myProject是自定义的库名。不需要自己手动建表建列，原因在随后的Spring Boot代码中会说明。

后端代码

配置文件

首先在src的resources文件夹下创建application.yml文件，Spring Boot会自动识别该文件名并加载为配置文件，识别成功后会看到文件图标变为一个绿叶。application.yml的代码如下：

```
spring:
  datasource:
    username: root
    password: root
    url: jdbc:mysql://localhost:3306/myProject?useSSL=false
  jpa:
    hibernate:
      # Hibernate ddl auto (create, create-drop, validate, update)
      ddl-auto: update
    properties:
      hibernate:
        # The SQL dialect makes Hibernate generate better SQL for the chosen
database
        dialect: org.hibernate.dialect.MySQLDialect
server:
  port: 8081
```

注意在同一行时冒号后要有空格，另外缩进要正确，jpa和datasource是同一级的。username是MySQL的用户名，password填写创建用户时设置的密码，url中的3306是MySQL工作的端口号，在Workbench的首页便会显示；myProject是要连接的数据库名称，创建数据库时使用的系统默认编码为utf-8，因而没有再在url属性中明确指出。

JPA可以理解为将Java对象与数据库中的数据（列、行等）建立映射关系的工具，同时它本身有CRUD操作的接口，因而在写相关代码时非常方便，随后会对此作出说明。前面提到不需要自己手动建表建列，正是因为在配置文件中设置了ddl-auto: update，从而JPA会根据实体类中的代码自动创建表列。

端口号为8081，即Spring Boot运行在 http://localhost:8081 上，主要因为前端运行在8080，因而为了防止端口冲突选择其他端口。

实体类

```
import jakarta.persistence.*;

@Entity
@Table(name = "dept")
public class Dept {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer deptNo;

    @Column(name = "dName")
    private String dName;

    @Column(name = "loc")
    private String loc;

    public Dept() {

    }

    public Dept(Integer deptNo, String dName, String loc) {
        super();
        this.deptNo = deptNo;
        this.dName = dName;
        this.loc = loc;
    }

    public Integer getDeptNo() {
        return deptNo;
    }
    public void setDeptNo(Integer deptNo) {
        this.deptNo = deptNo;
    }
```

```
    public String getDName() {
        return dName;
    }
    public void setDName(String dName) {
        this.dName = dName;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
}
```

注解@Entity标识了类是一个持久化的实体，并将其字段映射到指定的数据库表中；使用默认的orm规则，即类名就是数据库表中表名，类的字段名即表中字段名。因而类中的3个字段deptNo、dName、loc会被创建为表Dept中的列。需要注意，字段中的大写字母在映射到表名列名后为小写字母，不过会加上下划线以区分，比如dName会变为d_name。注解@Id用于声明一个实体类的属性被映射为数据库的主键列，@GeneratedValue用于标注主键的生成策略，通过strategy属性指定，GenerationType.AUTO表明把主键生成策略交给持久化引擎来选择具体方案。

Dao层

数据持久化操作即把数据放到持久化的介质中同时提供增删改查等操作，也就是CURD方法的定义处。代码如下：

```
import com.example.myProject.entity.Dept;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DeptRepository extends JpaRepository<Dept, Integer>{

}
```

由于JPA已经将CURD方法封装到接口中了，因而只需继承它即可。继承时类型参数传递的是自己创建的实体类以及主键的对象类型。

Controller层

基本的方法已经在Dao层定义，而方法该如何调用则在controller层中定义，它明确了如何与前端交互。代码如下：

```
import com.example.myProject.entity.Dept;
import com.example.myProject.repository.DeptRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
```

```java
import org.springframework.data.domain.PageRequest;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/dept")
public class DeptController {

    @Autowired
    private DeptRepository deptRepository;

    @GetMapping("/findAll/{page}/{size}")
    public Page<Dept> findAll(@PathVariable("page") Integer page,
@PathVariable("size") Integer size) {
        PageRequest request = PageRequest.of(page, size);
        return deptRepository.findAll(request);
    }

    @GetMapping("/findById/{deptNo}")
    public Dept findById(@PathVariable("deptNo") Integer deptNo) {
        return deptRepository.findById(deptNo).get();
    }

    @PostMapping("/save")
    public String save(@RequestBody Dept dept) {
        Dept result = deptRepository.save(dept);
        if (result != null){
            return "success";
        }else {
            return "error";
        }
    }

    @PutMapping("/update")
    public String update(@RequestBody Dept dept) {
        Dept result = deptRepository.save(dept);
        if (result != null) {
            return "success";
        }else {
            return "error";
        }
    }

    @DeleteMapping("/delete/{deptNo}")
    public void delete(@PathVariable("deptNo") Integer deptNo) {
        deptRepository.deleteById(deptNo);
    }
}
```

@RestController的作用等同于@Controller + @ResponseBody，前者表明该类是一个控制器类、生成相应的bean并注入到Spring容器中，后者表示方法的返回值直接以指定的格式写入Http response body中而不是解析为跳转路径，关系到前端正确地接收后端方法的返回值。

@Autowired会在容器中查询对应类型的bean实例并装配给该对象，因而当容器中已经有DeptRepository类型的实例时，由于bean是默认单例的，于是便将该实例的引用赋值给字段deptRepository。

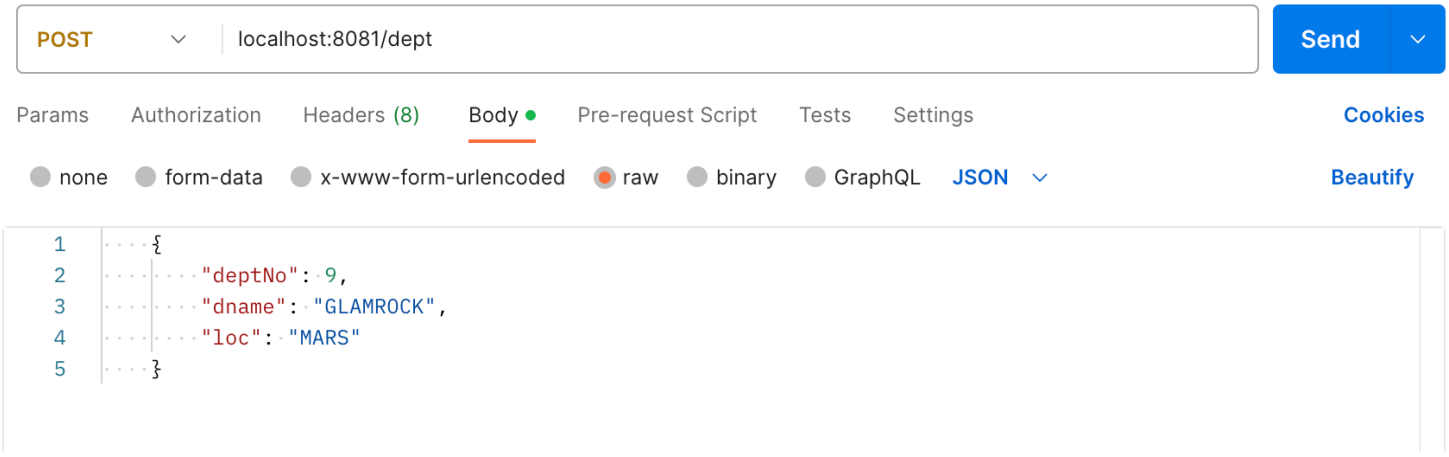
@RequestMapping用于将任意HTTP请求映射到控制器方法上，当它标记类时，指示的路径会是该类中方法的请求路径的上一级路径。比如在浏览器中输入url：http://localhost:8081/dept/save 则会请求调用save方法。

@GetMapping、@PostMapping、@PutMapping、@DeleteMapping是对@RequestMapping的一种简便写法，如@GetMapping的原先写法应为@RequestMapping(method = RequestMethod.GET)。Get主要用于只需要请求数据而不需要写入数据时；Post和Put在用法上区别不大，但一般插入新数据时用Post而更新数据时用Put，这主要是因为http协议规定Post为非幂等请求。所谓幂等的，即表明后面的请求会将前面的请求覆盖掉，这与更新数据是一致的；然而添加数据则会导致数据越来越多，这就是非幂等的，于是使用Post。

url路径不仅可以用于请求特定的后端方法，而且可以在路径中加入要传入的参数。比如@GetMapping("/findAll/{page}/{size}")便表明有两个形参page和size，接着通过方法形参中使用@PathVariable("page")注解来将url中含有的参数作为实参传递给方法。然而如此能传入的参数非常有限，于是有了@RequestBody注解。该常用来处理content-type不是默认的application/x-www-form-urlcoded编码的字符串，一个最常用的类型就是json格式数据。那如何才能将json数据传递给标注有@RequestBody的方法呢？一个方便的测试工具便是Postman，这将在下文提到。至此后端的关键代码就基本结束了，剩余一个跨域请求的问题将在前端代码的编写中提到。

测试

使用Postman测试，如图：



传递的参数在Body中给出，格式为json，注意左框中方法的选择需要与该路径映射到的控制器方法的注解一致。

前端代码

main.js

首先明确App.vue可以被视为整个项目的根组件，是vue页面资源的首加载项，同时也定义了所有页面中公共的动画或样式。如果在main.js中生成Vue实例时注释掉它的话页面会无法加载。但在原先的App.vue中会显示Vue的logo，将该部分删去即可，但要保留路由占位符，代码如下：

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

路由占位符router-view的作用是当路由路径与访问的地址相符时将指定的组件替换该router-view，因而其作用有点类似于变量，每当我们切换路由时就如同对其赋值。main.js的代码如下：

```
import Vue from 'vue'
import App from './App'
import router from './router'
import axios from 'axios'
import ElementUI from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'

Vue.use(ElementUI)

Vue.config.productionTip = false
Vue.prototype.$http = axios

new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>'
})
```

这里涉及到axios和Element UI的引入：

axios

简单而言，axios是前端向浏览器发送请求的库，它有get、post、put等方法，本项目正是通过axios来实现与后端控制器方法的对接。要使用axios，需要先进入当前项目的根文件夹，在终端执行`npm install --save axios`。import语句表明引入axios，$http则是自定义的一个全局变量。

Element UI

Element UI是一个方便的UI库，里面有大量的组件模版可以使用。注意使用的Element UI文档应该是基于Vue 2.x的，因为Vue 3的特性与2差异相对较大。

DeptManager.vue

这里先描述各组件的编写，最后再阐述子页面及路由的切换。在src/components下创建组件DeptManager用于部门数据的显示，这里使用的是Element UI中的Table模版，按自身需要修改后template的代码如下：

```
<template>
  <div>
    <el-table
      :data="tableData"
      border
      style="width: 100%">
      <el-table-column
        fixed
        prop="deptNo"
        label="编号"
        width="150">
      </el-table-column>
      <el-table-column
        prop="dname"
        label="部门名称"
        width="150">
      </el-table-column>
      <el-table-column
        prop="loc"
        label="地址"
        width="150">
      </el-table-column>
      <el-table-column
        label="操作"
        width="100">
        <template slot-scope="scope">
          <el-button @click="edit(scope.row)" type="text" size="small">修改</el-
button>
          <el-button @click="deleteDept(scope.row)" type="text" size="small">删
除</el-button>
        </template>
      </el-table-column>
    </el-table>
    <el-pagination background layout="prev, pager, next"
                   :page-size="pageSize"
                   :total="total"
                   @current-change="page">
    </el-pagination>
  </div>
</template>
```

其中data的名称是script中data函数返回的数据，在下文中会提及。fixed表明该列是左固定的，label是显示在页面上的该列的名称，width则是该列的宽度。可以将该Table直接看成MySQL的一个表，那么prop属性对应的就是数据库中该表的某一列的列名；data的数据格式为json，json文件中每一单元的数据可以看作键值对的集合，因而键就是列名，值则是某一行在该列中的数据。举个例子：

```
[{
deptNo: 2,
loc: 'SIDONIA',
dname: 'KNIGHTS'
},
{
deptNo: 52,
loc: 'TOKYO',
dname: 'NERV'
}]
```

该json数据便展示了表中某两行的数据。prop的作用就是按照给出的名称去挨个地取每个花括号中该键对应的值，并逐行显示在页面的表中。

代码 `<el-button @click="edit(scope.row)" type="text" size="small">修改</el-button>` 会在页面的该列下逐行地显示按钮，click后的双引号内为一个表达式，当表达式为真时按钮才显示；在这里是一个函数的调用，scope.row是该行数据及其他一些属性所构成的一个对象，该函数同样会在下文讲到。而代码：

```
<el-pagination background layout="prev, pager, next"
                :page-size="pageSize"
                :total="total"
                @current-change="page">
</el-pagination>
```

则实现了分页查询的功能，pageSize、total都是data函数所返回的变量值，page是分页查询时调用的函数，同样在下文提及。

另一部分是script的代码：

```
<script>
export default {
  methods: {
    page (currentPage) {
      this.$http.get('http://localhost:8081/dept/findAll/' + (currentPage - 1) +
'/6').then(resp => {
        // console.log(resp)
        this.tableData = resp.data.content
        this.pageSize = resp.data.size
        this.total = resp.data.totalElements
      })
    },
    edit (row) {
      this.$router.push({
        path: '/update',
        query: {
          deptNo: row.deptNo
        }
      })
```

```
    },
    deleteDept (row) {
      this.$http.delete('http://localhost:8081/dept/delete/' +
row.deptNo).then(resp => {
        this.$alert('删除成功! ', '消息', {
          confirmButtonText: '确定',
          callback: action => {
            window.location.reload()
          }
        })
      })
    }
  },
  data () {
    return {
      pageSize: '',
      total: '',
      tableData: []
    }
  },
  created () {
    this.$http.get('http://localhost:8081/dept/findAll/0/6').then(resp => {
      // console.log(resp)
      this.tableData = resp.data.content
      this.pageSize = resp.data.size
      this.total = resp.data.totalElements
    })
  }
}
</script>
```

可以看到大部分自定义的函数都使用了axios请求，因而需要先解决跨域问题（实际上，本项目仅是跨端口也是不行的）。所谓跨域问题即因为浏览器同源策略的限制而不允许不同域的访问请求，这里采取在后端配置CORS来解决。CORS是一个W3C标准，全称是“跨域资源共享”，它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。简单理解便是通过在后端列出允许被访问的地址（即前端运行的url），使前端能请求到后端的方法调用。后端中新增代码如下：

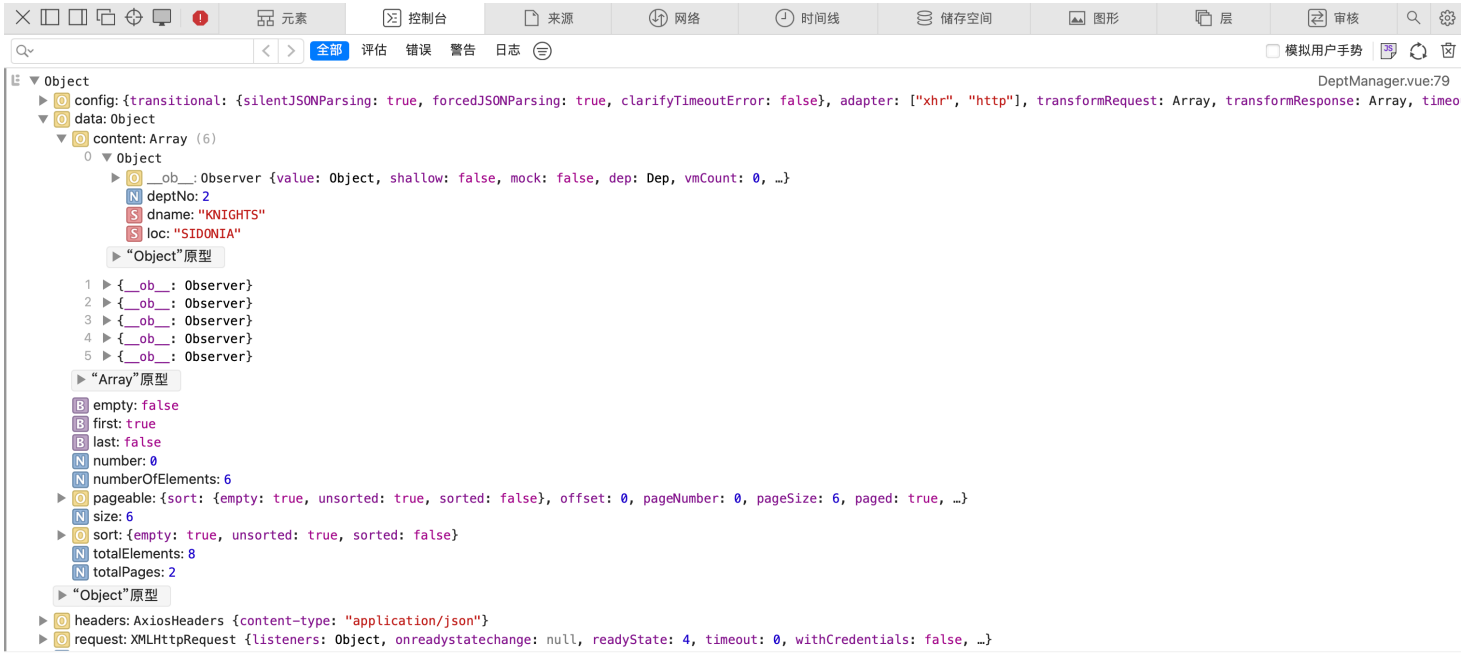
```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:8080")
                .allowedMethods("GET", "HEAD", "POST", "PUT", "DELETE",
"OPTIONS")
```

```
                .allowCredentials(true)
                .maxAge(3600)
                .allowedHeaders("*");
    }
}
```

注意CorsConfig中allowCredentials=true时allowedOrigins必须显式指定而不能为通配符。Access-Control-Allow-Credentials涉及计算机网络的通信原理，是服务端返回给客户端的响应头，意义是允许客户端携带验证信息（如cookie）及可以将对请求的响应暴露给页面。相关内容不在本项目范围内，因此不再深究。

要用到的变量需要预先在data中定义好。created函数在页面加载时被调用，为页面数据作初始化，其中的this指向调用它的vue实例。通过axios的get函数请求后端的findAll方法，get的返回值是一个含有多个属性的Object，可以在浏览器中打开控制台并在程序中调用`console.log(resp)`查看resp的内容，如下所示：



从resp的内容中也可以看到，实际的列名（json的键）为deptNo、dname和loc，dname与在实体类中的定义略有不符，因而前端在使用列名时需要注意一下resp的实际内容。接下来介绍methods中函数的编写：

page与created函数体相似，都是请求后端的findAll方法并将获取到的数据绑定到tableData，随后在template中逐行读取并显示，注意后端方法的返回类型为Page；在el-pagination中调用page时不需要显式传入参数，当前页数会自动传入。注意对tabelData等变量的赋值要放在then中，由于js是异步的，所以语句的执行并不会按照上下文的顺序；而axios的请求相对其他语句所要消耗的时间要长，因而如果将赋值语句放在then之外的话js很可能先将axios请求之外的语句执行了再去执行axios.get，那么最终获取到的数据就是undefined了。

函数deleteDept与page略有相似，注意scope.row传入的是resp.data.content中每一个json元素，表现为一个键值对数组，因而在函数中直接row.deptNo就能取到主键了；this.$alert会弹出一个消息框，如下图所示：

最终DeptManager页面的效果如图：

编号	部门名称	地址	操作
2	KNIGHTS	SIDONIA	修改 删除
52	NERV	TOKYO	修改 删除
102	RESEAECH	NEW YORK	修改 删除
202	GLAMROCK	MARS	修改 删除
303	SHOEGAZE	MBV	修改 删除
304	GOTHROCK	BAUHAUS	修改 删除

1 2

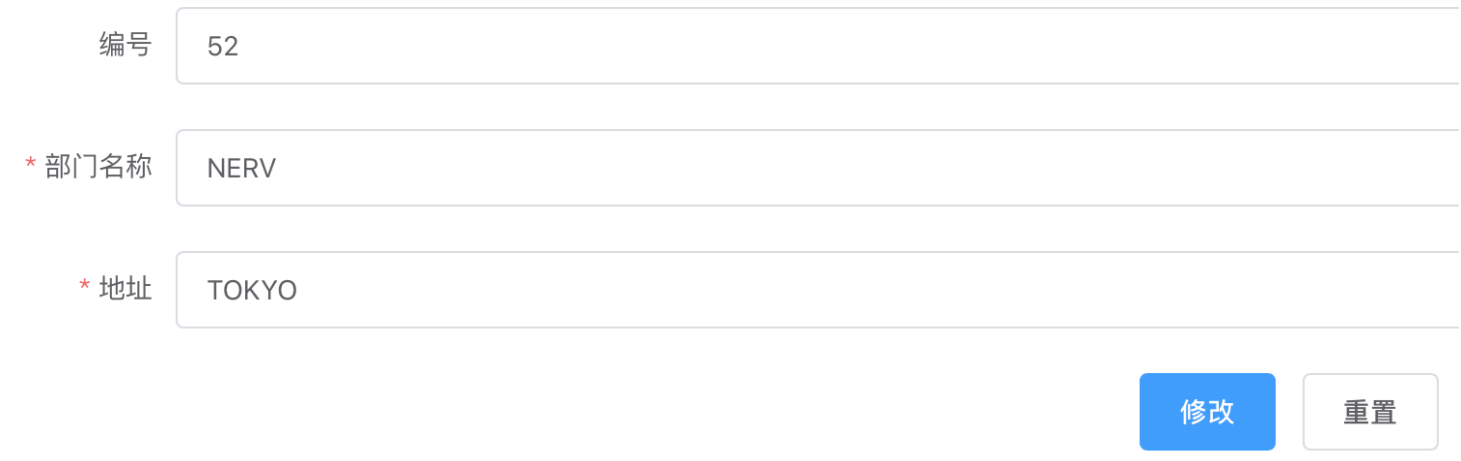
函数edit涉及路由（页面）的跳转，在编写完组件后再介绍。其功能是点击按钮“修改”后页面跳转到另一组件DeptUpdate.vue，于是接下来介绍DeptUpdate的编写。

DeptUpdate.vue

DeptUpdate要实现的是显示一个有输入框的表单，读取用户的输入并将其作为实参传递给后端的update方法进而修改数据库中的数据。表单使用Element UI的Form，依据实际需要修改后代码如下：

```
<template>
  <el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px"
class="demo-ruleForm">
    <el-form-item label="编号" prop="deptNo">
      <el-input v-model="ruleForm.deptNo" readonly></el-input>
    </el-form-item>
    <el-form-item label="部门名称" prop="dname">
      <el-input v-model="ruleForm.dname"></el-input>
    </el-form-item>
    <el-form-item label="地址" prop="loc">
      <el-input v-model="ruleForm.loc"></el-input>
    </el-form-item>
    <el-form-item>
      <el-button type="primary" @click="submitForm('ruleForm')">修改</el-button>
      <el-button @click="resetForm('ruleForm')">重置</el-button>
    </el-form-item>
  </el-form>
</template>
```

model是一个object类型的表单数据对象，rules是表单的验证规则（即用户输入是否合法并允许提交的判断依据），都会在下面的script中提前定义。ref属性在Vue中被用来给元素或子组件注册引用信息，简单而言可以理解为指向这个表单的引用，在script中可以通过this.$refs来获取这个对象。另外，Element UI的Form是已经带有许多方法可以使用了的，部分方法在script中会调用，其余（包括各种属性）可以在官方文档中查询。该表单显示如下图：



下面是script的代码：

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

```
      ruleForm: {
        deptNo: '',
        dname: '',
        loc: ''
      },
      rules: {
        dname: [
          {required: true, message: '请输入部门名称', trigger: 'blur'},
          {min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur'}
        ],
        loc: [
          {required: true, message: '请输入部门地址', trigger: 'blur'},
          {min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur'}
        ]
      }
    }
  },
  methods: {
    submitForm (formName) {
      this.$refs[formName].validate((valid) => {
        if (valid) {
          this.$http.put('http://localhost:8081/dept/update',
this.ruleForm).then(resp => {
            if (resp.data === 'success') {
              this.$alert('修改成功! ', '消息', {
                confirmButtonText: '确定',
                callback: action => {
                  this.$router.push('/DeptManager')
                }
              })
            }
          })
        } else {
          return false
        }
      })
    },
    resetForm (formName) {
      this.$refs[formName].resetFields()
    }
  },
  created () {
    this.$http.get('http://localhost:8081/dept/findById/' +
this.$route.query.deptNo)
      .then(resp => {
        this.ruleForm = resp.data
      })
  }
}
</script>
```

函数validate、resetFields等都是Form本身已经定义好的，直接调用即可。在submitForm中，若校

验成功则将表单内容发送给后端的update方法并弹出消息框，在用户点击“确定”之后通过`this.$router.push('/DeptManager')`跳转回DeptManager页面。

AddDept.vue

最后一个子页面是用于添加部门的组件，由于需求与DeptUpdate类似，都为获取用户输入之后作为实参传递给后端方法，因此同样用Form来作为前端显示，代码如下：

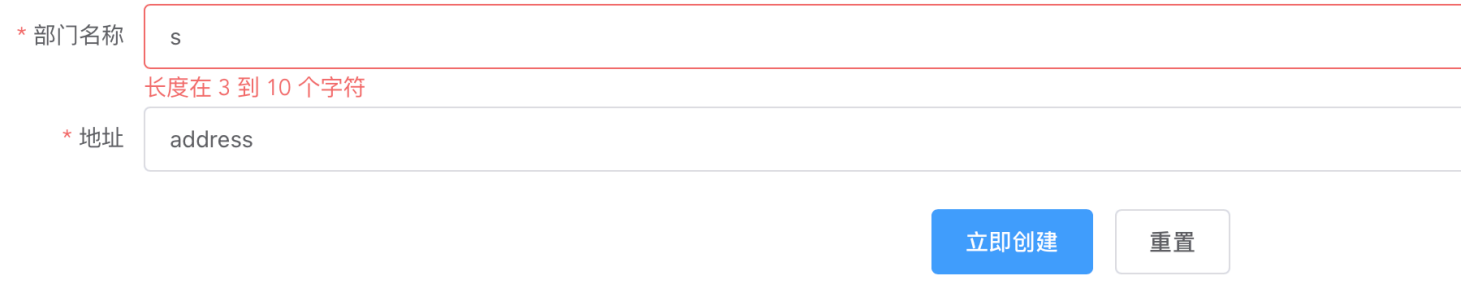
```
<template>
  <el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px"
class="demo-ruleForm">
    <el-form-item label="部门名称" prop="dname">
      <el-input v-model="ruleForm.dname"></el-input>
    </el-form-item>
    <el-form-item label="地址" prop="loc">
      <el-input v-model="ruleForm.loc"></el-input>
    </el-form-item>
    <el-form-item>
      <el-button type="primary" @click="submitForm('ruleForm')">立即创建</el-
button>
      <el-button @click="resetForm('ruleForm')">重置</el-button>
    </el-form-item>
  </el-form>
</template>
```

script部分也是类似的，代码如下：

```
<script>
export default {
  data () {
    return {
      ruleForm: {
        dname: '',
        loc: ''
      },
      rules: {
        dname: [
          {required: true, message: '请输入部门名称', trigger: 'blur'},
          {min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur'}
        ],
        loc: [
          {required: true, message: '请输入部门地址', trigger: 'blur'},
          {min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur'}
        ]
      }
    }
  },
  methods: {
    submitForm (formName) {
      this.$refs[formName].validate((valid) => {
```

```
        if (valid) {
          this.$http.post('http://localhost:8081/dept/save',
this.ruleForm).then(resp => {
            if (resp.data === 'success') {
              this.$alert('添加成功! ', '消息', {
                confirmButtonText: '确定',
                callback: action => {
                  this.$router.push('/DeptManager')
                }
              })
            }
          })
        } else {
          return false
        }
      })
    },
    resetForm (formName) {
      this.$refs[formName].resetFields()
    }
  }
}
</script>
```

效果如下图：



最大的区别在于axios请求的后端方法不同。至此，三个主要的显示页面已经完成，下面制作一个用于页面跳转的路由菜单并解决其他页面跳转的问题：

Index.vue

使用Element UI的Container来制作页面侧边的导航栏，由于三个子页面都已经有完整的显示内容了，这里只需要页面左侧栏的布局即可，代码如下：

```
<template>
  <div>
    <el-container style="height: 500px; border: 1px solid #eee">
      <el-aside width="200px" style="background-color: rgb(238, 241, 246)">
        <el-menu :default-active="this.$route.path" :router="true"
router:default-openeds="['0']">
          <el-submenu
            v-for="(item, index) in $router.options.routes"
            :key="item.name"
            :index="index+''"
          >
            <template slot="title">
              <i class="el-icon-message"></i>{{ item.name }}
            </template>
            <el-menu-item
              v-for="(item2) in item.children"
              :key="item2.name"
              :index="item2.path"
              :class="$route.path==item2.path?'is-active':''">{{ item2.name }}
            </el-menu-item>
          </el-submenu>
        </el-menu>
      </el-aside>

      <el-container>
        <el-main>
          <router-view></router-view>
        </el-main>
      </el-container>
    </el-container>
  </div>
</template>
```

```
<style>
.el-header {
  background-color: #B3C0D1;
  color: #333;
  line-height: 60px;
}

.el-aside {
  color: #333;
}
</style>
```

el-aside表明是侧边栏容器；this.$route表示当前路由对象，它是局部的，通过它可以获取对应的name, path, params, query等属性；default-openeds表示默认展开的菜单，default-active表示默认选中的菜单。稍后将会看到，其效果便是打开页面时“部门查询”为蓝色（表示选中的菜单），且父菜单栏“部门管理”会默认展开。

与this.$route$不同，$this$.router相当于一个全局的路由器对象，其包含了许多属性和对象，任何页面都可以调用它的push(), replace(), go()等方法（其中push的调用在稍后会讲到）；el-submenu表示可展开的菜单，它要遍历的$router.options.routes会在随后的router/index.js中定义，另外`item.name`、`item.children`都是在index.js中会定义的属性。在el-submenu中使用v-for遍历路由，从而在侧边栏显示相关信息。

el-main是主要区域容器，这里只需保留其路由占位符即可，当使用前文已经写好的组件时便会在该区域显示。效果如下图：

index.js

注意这是router而不是config文件夹中的index.js，其主要作用是配置路由，即各个页面所对应的路径和组件，代码如下：

```
import Vue from 'vue'
import Router from 'vue-router'
import Index from '../components/Index.vue'
import DeptManager from '../components/DeptManager.vue'
import AddDept from '../components/AddDept.vue'
import DeptUpdate from '../components/DeptUpdate.vue'

Vue.use(Router)

export default new Router({
  routes: [
```

```
    {
      path: '/',
      name: '部门管理',
      component: Index,
      redirect: '/DeptManager',
      children: [
        {
          path: '/DeptManager',
          name: '部门查询',
          component: DeptManager
        },
        {
          path: '/AddDept',
          name: '添加部门',
          component: AddDept
        },
        {
          path: '/update',
          component: DeptUpdate
        }
      ]
    }
  ]
})
```

path的名称是自行指定的，可以简单理解为一个标识符，其余的属性（name、component）都会与该路径绑定。在本项目中的使用案例便是DeptManager.vue中的edit函数，其通过this.$router.push来修改url完成页面跳转：

```
edit (row) {
    this.$router.push({
    path: '/update',
    query: {
        deptNo: row.deptNo
    }
    })
}
```

至此，小型部门管理系统就已经完成了。分别运行MyProjectApplication及npm run dev便可以访问 http://localhost:8080 看到效果了。

解决问题

application.yml文件没有变为绿叶

在重新打开项目之后便正常了，推测是Maven没有及时加载或者项目没有及时刷新等，可以重新构建工程或者点击“重新加载所有Maven项目”。

前端的delete函数没有正常调用

`el-button @click="delete(scope.row)"` 没能完成删除功能。这是因为delete是一个关键字，用作自定义的函数名时无法正常识别，因而改名为deleteDept。

点击侧边栏没能实现页面跳转

在中需要加上:default-active="this.$route.path" :router="true"来绑定路由表从而实现通过侧边栏来路由跳转，原先没有加上时是没有响应的。