今日目标

1.修改用户,删除用户

2.推送代码到码云

3.权限列表

4.角色列表

5.分配角色

1.修改用户信息

A.为用户列表中的修改按钮绑定点击事件
B.在页面中添加修改用户对话框，并修改对话框的属性
C.根据id查询需要修改的用户数据

```
//展示编辑用户的对话框
async showEditDialog(id) {
    //发送请求根据id获取用户信息
    const { data: res } = await this.$http.get('users/' + id)
    //判断如果添加失败，就做提示
    if (res.meta.status !== 200) return this.$message.error('获取用户信息失败')
    //将获取到的数据保存到数据editForm中
    this.editForm = res.data
    //显示弹出窗
    this.editDialogVisible = true
}
```

D.在弹出窗中添加修改用户信息的表单并做响应的数据绑定以及数据验证

```
<!-- 对话框主体区域 -->
<el-form :model="editForm" :rules="editFormRules" ref="editFormRef" label-
width="70px">
    <el-form-item label="用户名">
        <el-input v-model="editForm.username" disabled></el-input>
    </el-form-item>
    <el-form-item label="邮箱" prop="email">
        <el-input v-model="editForm.email"></el-input>
    </el-form-item>
    <el-form-item label="电话" prop="mobile">
        <el-input v-model="editForm.mobile"></el-input>
    </el-form-item>
</el-form>
```

数据绑定以及验证：

```
//控制修改用户对话框的显示与否
editDialogVisible: false,
//修改用户的表单数据
```

```
editForm: {
    username: '',
    email: '',
    mobile: ''
},
//修改表单的验证规则对象
editFormRules: {
    email: [
        { required: true, message: '请输入邮箱', trigger: 'blur' },
        {
        validator: checkEmail,
        message: '邮箱格式不正确，请重新输入',
        trigger: 'blur'
        }
    ],
    mobile: [
        { required: true, message: '请输入手机号码', trigger: 'blur' },
        {
        validator: checkMobile,
        message: '手机号码不正确，请重新输入',
        trigger: 'blur'
        }
    ]
}
```

E.监听对话框关闭事件，在对话框关闭之后，重置表单

```
<el-dialog title="修改用户" :visible.sync="editDialogVisible" width="50%"
@close="editDialogClosed">

editDialogClosed(){
    //对话框关闭之后，重置表达
    this.$refs.editFormRef.resetFields()
}
```

F.在用户点击确定按钮的时候，验证数据成功之后发送请求完成修改

```
editUser() {
    //用户点击修改表单中的确定按钮之后，验证表单数据
    this.$refs.editFormRef.validate(async valid => {
    if (!valid) return this.$message.error('请填写完整用户信息')
    //发送请求完成修改用户的操作
    const { data: res } = await this.$http.put(
        'users/' + this.editForm.id,
        this.editForm
    )
    //判断如果修改失败，就做提示
    if (res.meta.status !== 200) return this.$message.error('修改用户失败')
    //修改成功的提示
    this.$message.success('修改用户成功')
    //关闭对话框
    this.editDialogVisible = false
    //重新请求最新的数据
    this.getUserList()
    })
}
```

2.删除用户

在点击删除按钮的时候，我们应该跳出提示信息框，让用户确认要进行删除操作。
如果想要使用确认取消提示框，我们需要先将提示信息框挂载到vue中。
A.导入MessageBox组件，并将MessageBox组件挂载到实例。
Vue.prototype.$confirm = MessageBox.confirm
B.给用户列表中的删除按钮添加事件，并在事件处理函数中弹出确定取消窗,最后再根据id发送删除用户的请求

```
async removeUserById(id){
    //弹出确定取消框，是否删除用户
    const confirmResult = await this.$confirm('请问是否要永久删除该用户','删除提示',{
    confirmButtonText:'确认删除',
    cancelButtonText:'取消',
    type:'warning'
    }).catch(err=>err)
    //如果用户点击确认，则confirmResult 为'confirm'
    //如果用户点击取消，则confirmResult获取的就是catch的错误消息'cancel'
    if(confirmResult != "confirm"){
        return this.$message.info("已经取消删除")
    }
    //发送请求根据id完成删除操作
    const {data:res} = await this.$http.delete('users/'+id)
    //判断如果删除失败，就做提示
    if (res.meta.status !== 200) return this.$message.error('删除用户失败')
    //修改成功的提示
    this.$message.success('删除用户成功')
    //重新请求最新的数据
    this.getUserList()
}
```

3.推送代码

创建user子分支，并将代码推送到码云
A.创建user子分支 git checkout -b user
B.将代码添加到暂存区 git add .
C.将代码提交并注释 git commit -m '添加完成用户列表功能'
D.将本地的user分支推送到码云 git push -u origin user
E.将user分支代码合并到master:
切换到master git checkout master
合并user git merge user
F.将本地master分支的代码推送到码云 git push

创建rights子分支
A.创建rights子分支 git checkout -b rights
B.将本地的rights分支推送到码云 git push -u origin rights

4.权限列表

A.添加权限列表路由

创建权限管理组件（Rights.vue），并在router.js添加对应的路由规则

```
import Rights from './components/power/Rights.vue'
......
      path: '/home', component: Home, redirect: '/welcome', children: [
        { path: "/welcome", component: Welcome },
        { path: "/users", component: Users },
        { path: "/rights", component: Rights }
      ]
......
```

B.添加面包屑导航

在Rights.vue中添加面包屑组件展示导航路径

C.显示数据

在data中添加一个rightsList数据，在methods中提供一个getRightsList方法发送请求获取权限列表数据，在created中调用这个方法获取数据

```
<el-table :data="rightsList" stripe>
    <el-table-column type="index"></el-table-column>
    <el-table-column label="权限名称" prop="authName"></el-table-column>
    <el-table-column label="路径" prop="path"></el-table-column>
    <el-table-column label="权限等级" prop="level">
        <template slot-scope="scope">
            <el-tag v-if="scope.row.level === 0">一级权限</el-tag>
            <el-tag v-if="scope.row.level === 1" type="success">二级权限</el-tag>
            <el-tag v-if="scope.row.level === 2" type="warning">三级权限</el-tag>
        </template>
    </el-table-column>
</el-table>
<script>
export default {
    data(){
        return {
            //列表形式的权限
            rightsList:[]
        }
    },
    created(){
        this.getRightsList();
    },
    methods:{
        async getRightsList(){
            const {data:res} = await this.$http.get('rights/list')
            //如果返回状态为异常状态则报错并返回
            if (res.meta.status !== 200)
                return this.$message.error('获取权限列表失败')
            //如果返回状态正常，将请求的数据保存在data中
            this.rightsList = res.data
        }
    }
}
</script>
```

5.角色列表

A.添加角色列表路由

添加角色列表子组件（power/Roles.vue），并添加对应的规则

```
path: '/home', component: Home, redirect: '/welcome', children: [
        { path: "/welcome", component: Welcome },
        { path: "/users", component: Users },
        { path: "/rights", component: Rights },
        { path: "/roles", component: Roles  }
      ]
```

B.添加面包屑导航

在Roles.vue中添加面包屑组件展示导航路径

C.显示数据

在data中添加一个roleList数据，在methods中提供一个getRoleList方法发送请求获取权限列表数据，在created中调用这个方法获取数据

```
<!-- 角色列表区域 -->
<!-- row-key="id" 是2019年3月提供的新特性，
if there's nested data, rowKey is required.
如果这是一个嵌套的数据，rowkey 是必须添加的属性 -->
<el-table row-key="id" :data="roleList" border>
    <!-- 添加展开列 -->
    <el-table-column type="expand"></el-table-column>
    <el-table-column type="index"></el-table-column>
    <el-table-column label="角色名称" prop="roleName"></el-table-column>
    <el-table-column label="角色描述" prop="roleDesc"></el-table-column>
    <el-table-column label="操作" width="300px">
        <template slot-scope="scope">
            <el-button size="mini" type="primary" icon="el-icon-edit">编辑</el-
button>
            <el-button size="mini" type="danger" icon="el-icon-delete">删除</el-
button>
            <el-button size="mini" type="warning" icon="el-icon-setting">分配权限
</el-button>
        </template>
    </el-table-column>
</el-table>

<script>
export default {
    data(){
        return {
            roleList:[]
        }
    },created(){
        this.getRoleList();
    },methods:{
        async getRoleList(){
            const {data:res} = await this.$http.get('roles')
            //如果返回状态为异常状态则报错并返回
```

```
            // if (res.meta.status !== 200)
            //     return this.$message.error('获取角色列表失败')
            // //如果返回状态正常，将请求的数据保存在data中
            // this.roleList = res.data
            console.log(res.data)
            this.roleList = res.data;
        }
    }
}
</script>
```

D.补充说明

之前学习过类似的添加角色，删除角色，编辑角色请参照之前编写过的代码还有接口文档完成效果。

E.生成权限列表

使用三重嵌套for循环生成权限下拉列表

```
<!-- 添加展开列 -->
<el-table-column type="expand">
    <template slot-scope="scope">
        <el-row :class="['bdbottom',i1===0?'bdtop':'']" v-for="(item1,i1) in
scope.row.children" :key="item1.id">
            <!-- 渲染一级权限 -->
            <el-col :span="5">
                <el-tag>
                    {{item1.authName}}
                </el-tag>
                <i class="el-icon-caret-right"></i>
            </el-col>
            <!-- 渲染二，三级权限 -->
            <el-col :span="19">
                <!-- 通过for循环嵌套渲染二级权限  -->
                <el-row :class="[i2===0?'':'bdtop' ]" v-for="(item2,i2) in
item1.children" :key="item2.id">
                    <el-col :span="6">
                        <el-tag type="success">{{item2.authName}}</el-tag>
                        <i class="el-icon-caret-right"></i>
                    </el-col>
                    <el-col :span="18">
                        <el-tag type="warning" v-for="(item3) in item2.children"
:key="item3.id">
                            {{item3.authName}}
                        </el-tag>
                    </el-col>
                </el-row>
            </el-col>
        </el-row>
    </template>
</el-table-column>
```

F.美化样式

通过设置global.css中的#app样式min-width:1366px 解决三级权限换行的问题
，通过给一级权限el-row添加display:flex,align-items:center的方式解决一级权限垂直居中的问题，二级权限也类似添加，因为需要给多个内容添加，可以将这个样式设置为一个.vcenter{display:flex;align-items:center}

G.添加权限删除功能

给每一个权限的el-tag添加closable属性，是的权限右侧出现“X”图标
再给el-tag添加绑定close事件处理函数removeRightById(scope.row,item1.id)
removeRightById(scope.row,item2.id)
removeRightById(scope.row,item3.id)

```
async removeRightById(role,rightId){
    //弹窗提示用户是否要删除
    const confirmResult = await this.$confirm('请问是否要删除该权限','删除提示',{
        confirmButtonText:'确认删除',
        cancelButtonText:'取消',
        type:'warning'
    }).catch(err=>err)
    //如果用户点击确认，则confirmResult 为'confirm'
    //如果用户点击取消，则confirmResult获取的就是catch的错误消息'cancel'
    if(confirmResult != "confirm"){
        return this.$message.info("已经取消删除")
    }

    //用户点击了确定表示真的要删除
    //当发送delete请求之后，返回的数据就是最新的角色权限信息
    const {data:res} = await
this.$http.delete(`roles/${role.id}/rights/${rightId}`)
    if (res.meta.status !== 200)
        return this.$message.error('删除角色权限失败')

    //无需再重新加载所有权限
    //只需要对现有的角色权限进行更新即可
    role.children = res.data
    // this.getRoleList();

}
```

H.完成权限分配功能

先给分配权限按钮添加事件
<el-button size="mini" type="warning" icon="el-icon-setting" @click="showSetRightDialog">分配权限
在showSetRightDialog函数中请求权限树数据并显示对话框

```
async showSetRightDialog() {
    //当点击分配权限按钮时，展示对应的对话框
    this.setRightDialogVisible = true;
    //获取所有权限的数据
    const {data:res} = await this.$http.get('rights/tree')
    //如果返回状态为异常状态则报错并返回
    if (res.meta.status !== 200)
        return this.$message.error('获取权限树失败')
    //如果返回状态正常，将请求的数据保存在data中
    this.rightsList = res.data
}
```

添加分配权限对话框，并添加绑定数据setRightDialogVisible

```
<!-- 分配权限对话框 -->
<el-dialog title="分配权限" :visible.sync="setRightDialogVisible" width="50%">
    <span>这是一段信息</span>
    <span slot="footer" class="dialog-footer">
        <el-button @click="setRightDialogVisible = false">取 消</el-button>
        <el-button type="primary" @click="setRightDialogVisible = false">确 定
</el-button>
    </span>
</el-dialog>
```

l.完成树形结构弹窗

在element.js中引入Tree，注册Tree

```
<!-- 分配权限对话框 -->
<el-dialog title="分配权限" :visible.sync="setRightDialogVisible" width="50%"
@close="setRightDialogClose">
    <!-- 树形组件
    show-checkbox:显示复选框
    node-key:设置选中节点对应的值
    default-expand-all:是否默认展开所有节点
    :default-checked-keys 设置默认选中项的数组
    ref:设置引用 -->
    <el-tree :data="rightsList" :props="treeProps" show-checkbox node-key="id"
default-expand-all :default-checked-keys="defKeys" ref="treeRef"></el-tree>
    <span slot="footer" class="dialog-footer">
        <el-button @click="setRightDialogVisible = false">取 消</el-button>
        <el-button type="primary" @click="allotRights">确 定</el-button>
    </span>
</el-dialog>

<script>
export default {
  data() {
    return {
      //角色列表数据
      roleList: [],
      //控制分配权限对话框的显示
      setRightDialogVisible: false,
      //权限树数据
      rightsList: [],
      //树形控件的属性绑定对象
```

```
      treeProps: {
        //通过label设置树形节点文本展示authName
        label: 'authName',
        //设置通过children属性展示子节点信息
        children: 'children'
      },
      //设置树形控件中默认选中的内容
      defKeys: [],
      //保存正在操作的角色id
      roleId:''
    }
  },
  created() {
    this.getRoleList()
  },
  methods: {
    async getRoleList() {
      const { data: res } = await this.$http.get('roles')
      //如果返回状态为异常状态则报错并返回
      if (res.meta.status !== 200)
        return this.$message.error('获取角色列表失败')
      //如果返回状态正常，将请求的数据保存在data中
      // this.roleList = res.data
      console.log(res.data)
      this.roleList = res.data
    },
    async removeRightById(role, rightId) {
      //弹窗提示用户是否要删除
      const confirmResult = await this.$confirm(
        '请问是否要删除该权限',
        '删除提示',
        {
          confirmButtonText: '确认删除',
          cancelButtonText: '取消',
          type: 'warning'
        }
      ).catch(err => err)
      //如果用户点击确认，则confirmResult 为'confirm'
      //如果用户点击取消，则confirmResult获取的就是catch的错误消息'cancel'
      if (confirmResult != 'confirm') {
        return this.$message.info('已经取消删除')
      }

      //用户点击了确定表示真的要删除
      //当发送delete请求之后，返回的数据就是最新的角色权限信息
      const { data: res } = await this.$http.delete(
        `roles/${role.id}/rights/${rightId}`
      )
      if (res.meta.status !== 200)
        return this.$message.error('删除角色权限失败')

      //无需再重新加载所有权限
      //只需要对现有的角色权限进行更新即可
      role.children = res.data
      // this.getRoleList();
    },
    async showSetRightDialog(role) {
      //将role.id保存起来以供保存权限时使用
```

```
      this.roleId = role.id;
      //获取所有权限的数据
      const { data: res } = await this.$http.get('rights/tree')
      //如果返回状态为异常状态则报错并返回
      if (res.meta.status !== 200) return this.$message.error('获取权限树失败')
      //如果返回状态正常，将请求的数据保存在data中
      this.rightsList = res.data

      //调用getLeafKeys进行递归，将三级权限添加到数组中
      this.getLeafKeys(role, this.defKeys)
      //当点击分配权限按钮时，展示对应的对话框
      this.setRightDialogVisible = true
      console.log(this.defKeys)
    },
    getLeafKeys(node, arr) {
      //该函数会获取到当前角色的所有三级权限id并添加到defKeys中
      //如果当前节点不包含children属性，则表示node为三级权限
      if (!node.children) {
        return arr.push(node.id)
      }
      //递归调用
      node.children.forEach(item => this.getLeafKeys(item, arr))
    },
    setRightDialogClose() {
      //当用户关闭树形权限对话框的时候，清除掉所有选中状态
      this.defKeys = []
    },
    async allotRights() {
      //当用户在树形权限对话框中点击确定，将用户选择的
      //权限发送请求进行更新

      //获取所有选中及半选的内容
      const keys = [
        ...this.$refs.treeRef.getCheckedKeys(),
        ...this.$refs.treeRef.getHalfCheckedKeys()
      ]
      //将数组转换为 , 拼接的字符串
      const idStr = keys.join(',')

      //发送请求完成更新
      const { data: res } = await this.$http.post(
        `roles/${this.roleId}/rights`,
        { rids:idStr }
      )
      if (res.meta.status !== 200)
        return this.$message.error('分配权限失败')

      this.$message.success("分配权限成功")
      this.getRoleList();
      //关闭对话框
      this.setRightDialogVisible = false;
    }
  }
}
</script>
```

6.分配角色

打开Users.vue，完成分配角色的功能
A.添加分配角色对话框

```
<!-- 分配角色对话框 -->
<el-dialog title="分配角色" :visible.sync="setRoleDialogVisible" width="50%">
    <div>
    <p>当前的用户:{{userInfo.username}}</p>
    <p>当前的角色:{{userInfo.role_name}}</p>
    <p>分配新角色:</p>
    </div>
    <span slot="footer" class="dialog-footer">
    <el-button @click="setRoleDialogVisible = false">取 消</el-button>
    <el-button type="primary" @click="setRoleDialogVisible = false">确 定</el-
button>
    </span>
</el-dialog>
```

B.给分配角色按钮添加点击事件，点击之后弹出一个对话框进行角色分配

```
<!-- 分配角色 -->
<el-tooltip class="item" effect="dark" content="分配角色" placement="top"
:enterable="false">
    <el-button type="warning" icon="el-icon-setting" size='mini'
@click="setRole(scope.row)"></el-button>
</el-tooltip>

data(){
    ......
    //控制显示分配角色对话框
    setRoleDialogVisible:false,
    //保存正在操作的那个用户信息
    userInfo:{},
    //保存所有的角色信息
    rolesList:[],
    //保存用户选中的角色id
    selectedRoleId:''
},
methods:{
    ......
    async setRole( userInfo ){
      //保存起来以供后续使用
      this.userInfo = userInfo;
      //获取所有的角色信息，以备下拉列表使用
      //发送请求根据id完成删除操作
      const { data: res } = await this.$http.get('roles')
      //判断如果删除失败，就做提示
      if (res.meta.status !== 200) return this.$message.error('获取角色列表失败')

      this.rolesList = res.data;
      //展示分配角色对话框
      this.setRoleDialogVisible = true;


    }
```

```
}
```

C.在element.js中引入Select，Option，注册Select，Option

```
<!-- 角色选择下拉框
v-model：设置用户选中角色之后的id绑定数据
-->
<el-select v-model="selectedRoleId" placeholder="请选择角色">
<!-- :label 显示文本，:value 选中值 -->
<el-option v-for="item in rolesList" :key="item.id" :label="item.roleName"
:value="item.id">
</el-option>
</el-select>
```

D.当用户点击对话框中的确定之后，完成分配角色的操作

```
<!-- 分配角色对话框 -->
<el-dialog title="分配角色" :visible.sync="setRoleDialogVisible" width="50%"
@close="setRoleDialogClosed">
    <div>
    <p>当前的用户:{{userInfo.username}}</p>
    <p>当前的角色:{{userInfo.role_name}}</p>
    <p>分配新角色:
        <!-- 角色选择下拉框
        v-model：设置用户选中角色之后的id绑定数据
        -->
        <el-select v-model="selectedRoleId" placeholder="请选择角色">
        <!-- :label 显示文本，:value 选中值 -->
        <el-option v-for="item in rolesList" :key="item.id"
:label="item.roleName" :value="item.id">
        </el-option>
        </el-select>
    </p>
    </div>
    <span slot="footer" class="dialog-footer">
    <el-button @click="setRoleDialogVisible = false">取 消</el-button>
    <el-button type="primary" @click="saveRoleInfo">确 定</el-button>
    </span>
</el-dialog>


methods:{
    .......
    async saveRoleInfo(){
      //当用户点击确定按钮之后
      //判断用户是否选择了需要分配的角色
      if(!this.selectedRoleId){
        return this.$message.error('请选择需要分配的角色')
      }
      //发送请求完成分配角色的操作
      const {data:res} = await this.$http.put(`users/${this.userInfo.id}/role`,
{rid:this.selectedRoleId})

      //判断如果删除失败，就做提示
      if (res.meta.status !== 200)
        return this.$message.error('分配角色失败')
```

```
      this.$message.success('分配角色成功')
      this.getUserList();
      //关闭对话框
      this.setRoleDialogVisible = false
    },
    setRoleDialogClosed(){
      //当关闭对话框的时候，重置下拉框中的内容
      this.selectedRoleId = ''
      this.userInfo = {}
    }
}
```

7.将代码推送到码云

A.将代码推送到暂存区 git add .

B.将代码提交到仓库 git commit -m '完成了权限功能开发'

C.将rights分支代码推送到码云 git push

D.将代码合并到master

git checkout master

git merge rights

E.将master代码推送到码云

git push