

## 基于Element的前端管理端与Java后端的联调(中)

### 1、前端管理端的用户 pagination 的实现：

用户的分页组件：



```
1 <el-table :data="tableData" style="width: 60%">
2   <el-table-column label="日期" width="180">
3     <template slot-scope="scope">
4       <i class="el-icon-time"></i>
5       <span style="margin-left: 10px">{{ scope.row.date }}</span>
6     </template>
7   </el-table-column>
8   <el-table-column label="姓名" width="180">
9     <template slot-scope="scope">
10      <el-popover trigger="hover" placement="top">
11        <p>姓名: {{ scope.row.name }}</p>
12        <p>住址: {{ scope.row.address }}</p>
13        <div slot="reference" class="name-wrapper">
14          <el-tag size="medium">{{ scope.row.name }}</el-tag>
15        </div>
16      </el-popover>
17    </template>
18  </el-table-column>
```

当前列表显示的用户信息：

- 日期：scope.row.date
- 用户名：scope.row.name

对照后台java的用户列表信息：

```
1 PageBean{
2   private Integer total;
3   private List<User> row;
4 }
5
6 User{
7   private Long id;
8   private String userName;
9   private String userPassword;
```

```

10     private Integer isDelete;
11     private Integer userRole;
12     private Date createTime;
13     private Date updateTime;
14 }

```

对照上述的User信息，组件处要修改：

- id, 对应scope.row.id
  - userName, 对应scope.row.userName
- updateTime, 对应scope.row.udpateTime

分页组件的"pageSize"、"current-page", "total"变量的绑定以及@current-change事件绑定：

```

1 <el-pagination
2   background
3   layout="prev, pager, next"
4   :page-size="pageSize"
5   :total="total"
6   //新增变量以及绑定方法;
7   :current-page="currentPage"
8   @current-change="handlePageChange"
9 >
10 </el-pagination>

```

参考question那边的handlePageChange方法：

```

1 //数据模型:
2 pageSize: 5, // 每页显示的条数
3 total: 0,    // 总条数
4 currentPage: 1, //首页
5
6 handlePageChange(page) {
7   this.currentPage = page;
8   axios
9     .get(`/users?page=${this.currentPage}&pageSize=${this.pageSize}`)
10    .then((response) => {
11      this.tableData = response.data.data.row;
12      this.total = response.data.data.total;
13    })
14    .catch((error) => {
15      console.error("Error fetching questions:", error);
16    });
17 },
18
19 mounted(){
20   this.handlePageChange(1);
21 }

```

## 2、前端管理端的用户查询



上述输入框中，输入题目的关键词，再调用/findUser接口获取相关的题目；

a、在输入框组件中，设定keyword:

```
1 //组件:
2 <el-form-item label="题目">
3   <el-input
4     v-model="formInline.keyword" //改为keyword
5     placeholder="请输入题目关键词"
6   ></el-input>
7 </el-form-item>
8
9   //绑定onSearch()方法;
10 <el-form-item>
11   <el-button type="primary" @click="onSearch">查询</el-button>
12 </el-form-item>
13
14 //脚本中;
15 formInline: {
16   keyword: "",
17   // region: "",
18 },
19
20 onSearch() {
21   // 这里可以添加查询逻辑
22   console.log("Searching for:", this.formInline.keyword);
23   // 例如, 调用API获取数据
24   axios
25     .get(`/findUser?keyword=${this.formInline.keyword}`)
26     .then((response) => {
27       this.tableData = response.data.data;
28       console.log("Search results:", response.data);
29       this.total = response.data.data.total;
30     })
31     .catch((error) => {
32       console.error("Error searching questions:", error);
33     });
34 },
```

结果:

## 3、前端管理端的用户添加

查询

添加用户

添加新用户

用户名

密码

确认密码

取消

确定

1、在组件中，添加onAddNewUser事件绑定；

```
1 <el-button type="primary" @click="onAddNewUser">确定</el-button>
```

2、js脚本中实现事件处理函数：

```
1 export default {
2   data() {
3     ...
4   },
5   methods: {
6     onAddNewUser() {
7       console.log("Submitting user:", this.form);
8       axios
9         .post("/register", this.form)
10        .then((response) => {
11          console.log("user register successfully:", response.data);
12          this.dialogFormVisible = false;
13          this.handlePageChange(this.currentPage);
14        })
15        .catch((error) => {
16          console.error("Error adding question:", error);
17        });
18    },
19  },
20 };
21 </script>
```

出现的问题：

前端的确有数据发送到后端；

```
1 onSubmItQuestion() {
2   console.log("Submitting question:", this.form);
3   axios
4     .post("/register", this.form)
```

但是后端没有正确接收到数据；

```

1 | @PostMapping("/register")
2 | public Result addUser(String username, String password, String
   | checkpassword) {
3 | }

```

可能的原因：前端发送的数据格式与后端的接收方式不一致。

如何解决：

前后端统一数据格式，均采用json格式：前端按json格式发送数据，后端按json格式接收及解析数据。

```

1 | .post("/register", this.form,{
2 |     headers: {
3 |         "Content-Type": "application/json",
4 |     },
5 | })

```

```

1 |

```

经过上述的修改，可以正常地插入数据(插入到最后)

作业和练习：

前端根据后端的返回，弹出一个对话框，告知用户，添加新用户成功。

#### 4、前端管理端的用户删除

##### 1、删除按钮绑定事件函数：

```

1 | <el-button
2 |     size="mini"
3 |     type="danger"
4 |     @click="handleDelete(scope.$index, scope.row)"
5 | >删除</el-button>
6 | >

```

##### 2、在js脚本中，书写删除命令对应的函数：

```

1 | handleDelete(index, row) {
2 |     const id = row.id; // 获取当前用户的 id
3 |     this.$confirm("此操作将永久删除该用户，是否继续?", "提示", {
4 |         confirmButtonText: "确定",
5 |         cancelButtonText: "取消",
6 |         type: "warning",
7 |     })
8 |     .then(() => {
9 |         // 调用删除接口 (GET 请求)
10 |         axios
11 |             .get(`/deleteById?id=${id}`) // 使用 GET 请求传递 id 参数
12 |             .then((response) => {
13 |                 console.log(response.data);
14 |                 // 删除成功后刷新当前页数据
15 |                 this.handlePageChange(this.currentPage);

```

```

16         })
17         .catch((error) => {
18             console.error("Error deleting question:", error);
19             this.$message({
20                 type: "error",
21                 message: "删除失败，请稍后重试！",
22             });
23         });
24     })
25     .catch(() => {
26         this.$message({
27             type: "info",
28             message: "已取消删除",
29         });
30     });
31 },

```

处理函数中需要注意的：

- `axios.get(/deleteById?id=${id})` // 使用 GET 请求传递 id 参数

作业和练习：

实现"编辑"按钮的功能，点击后弹出对话框(类似于添加用户按钮)，

可以对questionText，各个optionText和answer的内容进行修改更新。