

第2章 预约管理-检查项、检查组管理

学习目标：

- 掌握新增检查项实现过程
- 掌握检查项分页查询实现过程
- 掌握删除检查项实现过程
- 掌握编辑检查项实现过程
- 掌握新增检查组实现过程
- 掌握检查组分页查询实现过程
- 掌握删除检查组实现过程
- 掌握编辑检查组实现过程

1. 新增检查项

本章节完成的功能开发是预约管理功能，包括检查项管理（身高，体重）、检查组管理（外科）、体检套餐管理（公司健康体检）、预约设置等（参见产品原型）。预约管理属于系统的基础功能，主要就是管理一些体检的基础数据。

【目标】

新增检查项

【路径】

1. checkitem.html页面，找到【新建】，绑定事件，弹出新增窗口（修改dialogFormVisible=true）
2. 重置表单，点击【确定】，表单校验通过后，提交formData post请求到后台。对结果提示，如果成功则关闭窗口，刷新列表。
3. checkItemController提供添加的方法 使用CheckItem接收formData, 调用服务，返回结果给页面
4. CheckItemService及实现类, 调用Dao添加
5. CheckItemDao与映射文件 提供添加的方法
insert into

【讲解】

1.1. 前台代码

检查项管理页面对应的是checkitem.html页面，根据产品设计的原型已经完成了页面基本结构的编写，现在需要完善页面动态效果。

1.1.1. 弹出新增窗口

页面中已经提供了新增窗口，只是处于隐藏状态。只需要将控制展示状态的属性dialogFormVisible改为true就可以显示出新增窗口。


新建按钮绑定的方法为handleCreate，所以在handleCreate方法中修改dialogFormVisible属性的值为true即可。同时为了增加用户体验度，需要每次点击新建按钮时清空表单输入项。

```

1 // 重置表单
2 resetForm() {
3     //还原为初始的值
4     this.formData = {};
5 },
6 // 弹出添加窗口
7 handleCreate() {
8     this.resetForm();
9     this.dialogFormVisible = true;
10 },

```

1.1.2. 输入校验

image-20200909082556925

1.1.3. 提交表单数据

点击新增窗口中的确定按钮时，触发handleAdd方法，所以需要在handleAdd方法中进行完善。

```

1 //添加
2 handleAdd () {
3     //校验表单输入项是否合法
4     this.$refs['dataAddForm'].validate((valid) => {
5         if (valid) {
6             //表单数据校验通过，发送ajax请求将表单数据提交到后台
7             axios.post("/checkitem/add.do",this.formData).then((res)=> {
8                 //隐藏新增窗口
9                 this.dialogFormVisible = false;
10                //判断后台返回的flag值，true表示添加操作成功，false为添加操作失败
11                if(res.data.flag){
12                    this.$message({
13                        message: res.data.message,
14                        type: 'success'
15                    });
16                }else{
17                    this.$message.error(res.data.message);
18                }
19            })
20        } else {
21            this.$message.error("表单数据校验失败");
22            return false;
23        }
24    });
25 },
26 //分页查询
27 findPage() {
28 },

```

1.2. 后台代码

1.2.1. Controller

在health_web工程中创建CheckItemController

```

1 package com.itheima.health.controller;

```

```

2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.itheima.health.constant.MessageConstant;
5 import com.itheima.health.entity.Result;
6 import com.itheima.health.pojo.CheckItem;
7 import com.itheima.health.service.CheckItemService;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RestController;
11
12 /**
13  * 体检检查项管理
14  */
15 @RestController
16 @RequestMapping("/checkitem")
17 public class CheckItemController {
18
19     @Reference
20     private CheckItemService checkItemService;
21
22     /**
23      * 新增
24      * @param checkItem 接收前端提交过来的formData
25      * @return
26      */
27     @PostMapping("/add")
28     public Result add(@RequestBody CheckItem checkItem){
29         // 调用服务添加
30         checkItemService.add(checkItem);
31         return new Result(true, MessageConstant.ADD_CHECKITEM_SUCCESS);
32     }
33 }

```

1.2.2. 服务接口

在health_interface工程中创建CheckItemService接口

```

1 package com.itheima.health.service;
2
3 import com.itheima.health.pojo.CheckItem;
4
5 /**
6  * 检查项服务接口
7  */
8 public interface CheckItemService {
9
10     /**
11      * 添加检查项
12      * @param checkItem
13      */
14     void add(CheckItem checkItem);
15 }

```

1.2.3. 服务实现类

在health_service工程中创建CheckItemServiceImpl实现类

```
1 package com.itheima.health.service;
2
3 import com.alibaba.dubbo.config.annotation.Service;
4 import com.itheima.health.dao.CheckItemDao;
5 import com.itheima.health.pojo.CheckItem;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.transaction.annotation.Transactional;
8
9 /**
10  * 检查项服务
11  */
12 @Service(interfaceClass = CheckItemService.class)
13 public class CheckItemServiceImpl implements CheckItemService {
14
15     @Autowired
16     private CheckItemDao checkItemDao;
17
18     /**
19      * 添加检查项
20      * @param checkItem
21      */
22     public void add(CheckItem checkItem) {
23         checkItemDao.add(checkItem);
24     }
25 }
```

1.2.4. Dao接口

在health_dao工程中创建CheckItemDao接口，本项目是基于Mybatis的Mapper代理技术实现持久层操作，故只需要提供接口和Mapper映射文件，无须提供实现类

```
1 package com.itheima.health.dao;
2
3 import com.itheima.health.pojo.CheckItem;
4 /**
5  * 持久层Dao接口
6  */
7 public interface CheckItemDao {
8
9     /**
10      * 添加检查项
11      * @param checkItem
12      */
13     void add(CheckItem checkItem);
14 }
```

1.2.5. Mapper映射文件

在health_dao工程中创建CheckItemDao.xml映射文件，需要和CheckItemDao接口在同一目录下

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3     "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
4 <mapper namespace="com.itheima.health.dao.CheckItemDao">
5     <!--新增-->
6     <insert id="add" parameterType="com.itheima.health.pojo.CheckItem">
7         insert into
8         t_checkitem(code,name,sex,age,price,type,remark,attention)
9         values
10        (#{code},#{name},#{sex},#{age},#{price},#{type},#{remark},#{
11        {attention}})
12    </insert>
13 </mapper>

```

【小结】

1. 先分析，业务对象是谁，对应表结构在哪里，表中的字段是什么意思
2. 新增，表中数据从哪里来，从前端, 提供输入formData双向绑定，提交数据formData
3. 写出操作的步骤，按着步骤完成代码
4. 套路：
前端提请求 数据类型->Controller接收数据型->调用service->调用dao->映射文件
dao->service->controller->页面，提示结果，如果成功怎么样，失败怎么样

2. 检查项分页

【目标】

- 1: 熟悉分页功能中的请求参数
- 2: 熟悉分页功能中的响应数据
- 3: 检查项分页功能实现

【路径】

- 1: 前台代码 checkitem.html

```

1 提交的数据 pagination
2  {
3      页码 currentPage
4      大小 pageSize
5      条件 queryString
6  }
7  后端响应的数据
8  Result{
9      flag
10     message
11     data: { => result
12         total:
13         rows:
14     }
15 }

```

步骤:

1. 页面加载时分页查询created方法调用findPage()
2. findPage来实现分页查询, 发送post请求, 提交pagination, 得到后端响应的数据, 失败要提示, 成功则绑定数据
总记录数 `this.pagination.total = res.data(result).data.total`
结果集 `this.dataList = res.data.data.rows;`
3. CheckItemController findPage 使用QueryPageBean接收pagination,调用服务查询, 返回PageResult(total,rows), 封装到Result再返回给前端
4. CheckItemService 分页查询 使用pageHelper, startPage(页码, 大小), 判断是否有查询条件, 有则需要拼接%, 查询, 返回PageResult
5. CheckItemDao与映射文件,提交findByCondition条件查询 返回Page (total,rows) 对象,即可
6. 分页插件PageHelper会帮我分页查询

本项目所有分页功能都是基于ajax的异步请求来完成的, 请求参数和后台响应数据格式都使用json数据格式。

1: 请求参数包括页码、每页显示记录数、查询条件。

请求参数的json格式为: {currentPage:1,pageSize:10,queryString:"itcast"}

2: 后台响应数据包括总记录数、当前页需要展示的数据集合。

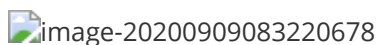
响应数据的json格式为: {total:1000,rows:[]}

如下图:



2.1. 前台代码

2.1.1. 定义分页相关模型数据



2.1.2. 定义分页方法

在页面中提供了findPage方法用于分页查询, 为了能够在checkitem.html页面加载后直接可以展示分页数据, 可以在VUE提供的钩子函数created中调用findPage方法

```
1 //钩子函数, VUE对象初始化完成后自动执行
2 created() {
3   this.findPage();
4 },
```

在VUE的methods中的findPage方法里添加

```
1 // 发送查询的请求, 提交pagination(currentPage,pageSize)
2 findPage() {
3   axios.post('/checkitem/findPage.do', this.pagination).then(res => {
4     if(res.data.flag){
5       //res.data => {flag, message,data : {total, rows}}
6       // 分页的结果
```

```

7         this.dataList = res.data.data.rows;
8         // 总记录
9         this.pagination.total = res.data.data.total;
10    }else{
11        // 失败则提示错误信息
12        this.$message.error(res.data.message);
13    }
14    });
15 }

```

2.1.3. 完善分页方法执行时机

除了在created钩子函数中调用findPage方法查询分页数据之外，当用户点击查询按钮或者点击分页条中的页码时也需要调用findPage方法重新发起查询请求。

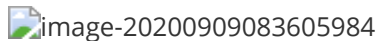
为查询按钮绑定单击事件，调用handleCurrentChange方法

```

1 | <el-button @click="handleCurrentChange(1)" class="dalfBut">查询</el-button>

```

为分页条组件绑定current-change事件，此事件是分页条组件自己定义的事件，当页码改变时触发，对应的处理函数为handleCurrentChange



修改Vue中methods中的handleCurrentChange方法

```

1 //切换页码
2 handleCurrentChange(currentPage) {
3     //currentPage 为修改后的页面
4     // 设置分页查询时使用的页码
5     this.pagination.currentPage = currentPage;
6     // 分页查询
7     this.findPage();
8 }

```

2.2. 后台代码

2.2.1. Controller

在CheckItemController中增加分页查询方法

```

1  /**
2   * 分页条件查询
3   */
4  @PostMapping("/findPage")
5  public Result findPage(@RequestBody QueryPageBean queryPageBean){
6      // 调用业务来分页，获取分页结果
7      PageResult<CheckItem> pageResult =
8      checkItemService.findPage(queryPageBean);
9
10     //return pageResult;
11     // 返回给页面，包装到Result，统一风格
12     return new Result(true,
13         MessageConstant.QUERY_CHECKITEM_SUCCESS,pageResult);
14 }

```

2.2.2. 服务接口

在CheckItemService服务接口中扩展分页查询方法

```

1  /**
2   * 分页条件查询
3   * @param queryPageBean
4   * @return
5   */
6  PageResult<CheckItem> findPage(QueryPageBean queryPageBean);

```

2.2.3. 服务实现类

在CheckItemServiceImpl服务实现类中实现分页查询方法，基于Mybatis分页助手插件实现分页

```

1  /**
2   * 分页查询
3   * @param queryPageBean
4   * @return
5   */
6  @Override
7  public PageResult<CheckItem> findPage(QueryPageBean queryPageBean) {
8      //第二种，Mapper接口方式的调用，推荐这种使用方式。
9      PageHelper.startPage(queryPageBean.getCurrentPage(),
10         queryPageBean.getPageSize());
11      // 模糊查询 拼接 %
12      // 判断是否有查询条件
13      if(!StringUtils.isEmpty(queryPageBean.getQueryString())){
14          // 有查询条件，拼接%
15          queryPageBean.setQueryString("%" + queryPageBean.getQueryString() +
16              "%");
17      }
18      // 紧接着的查询语句会被分页
19      Page<CheckItem> page =
20      checkItemDao.findByCondition(queryPageBean.getQueryString());
21      // 封装到分页结果对象中
22      PageResult<CheckItem> pageResult = new PageResult<CheckItem>
23      (page.getTotal(), page.getResult());
24      return pageResult;
25 }

```


正常分页逻辑（即：不使用PageHelper）：

- 写2条sql
- 第一条sql: select count(0) from t_checkitem where name = '传智身高' -->封装到PageResult中 total* 第二条sql: select * from t_checkitem where name = '传智身高' limit ?,? -->封装到PageResult中rows*
- limit ?,? : 第一个问号: 表示从第几条开始检索 计算: (currentPage-1)*pageSize 第二个问号: 表示当前页最多显示的记录数, 计算: pageSize
- 问题: 1: 麻烦; 2: 不能通用, 当前切换数据库的时候改动的代码多。

2.2.4. Dao接口

在CheckItemDao接口中扩展分页查询方法

```
1  /**
2   * 分页条件查询
3   * @return
4   */
5  Page<CheckItem> findByCondition(String queryString);
```

2.2.5. Mapper映射文件

在CheckItemDao.xml文件中增加SQL定义

```
1  <!--条件查询-->
2  <select id="findByCondition" resultType="Checkitem" parameterType="String">
3      select * From t_checkitem
4      <if test="value !=null and value.length > 0">
5          where code like #{value} or name like #{value}
6      </if>
7  </select>
8  <!-- Mybatis 动态参数赋值 DynamicContext
9   <if>标签里的变量, 如果参数类型是基本数据类型, 只能用 value 或 _parameter, 这个是由它
   的底层ognl表达式决定的。如果参数类型是对象类型, 则可以填它的属性。另外, 使用#的参数可以是
   形参名也可以是value
10  -->
```

如果使用if进行判断, 这里需要是value读取值, 不能改成其他参数。

【小结】

1. 提交currentPage, pageSize
2. 使用PageHelper

```
1  配置它
2  mybatis的拦截器
3  PageHelper.startPage(页码, 大小)
4  dao的查询语句会被分页
5
6  拦截sql语句, 拼接查询总计录数的语句
7      拼接分页的语句
8  ThreadLocal 线程绑定数据
```

3. 前端接收返回的结果, 格式Result(PageResult)

```
1 result={
2     flag:true,
3     message:'查询成功',
4     data:{ // pageResult
5         total: 100,
6         rows: [{checkitem}]
7     }
8 }
9 res.data => result
10 查询的结果集 res.data.data.rows
11 总计录数 res.data.data.total
```

4. 分页查询的时机:

- 进入页面
- 页码变更时要查询
- 点击查询按钮时要查, 回到第一页
- 添加成功后要查
- 修改成功后要查
- 删除成功后要查

3. 删除检查项

【目标】

删除检查项

【路径】

1. 绑定【删除】, 获取要删除的id, 弹出询问窗口, 确定后, 发送删除的请求, 提交id, 提示操作的结果, 如果成功则要刷新列表
2. CheckItemController, 接收id, 调用服务方法删除, 返回结果给页面
3. CheckItemService
 - 判断是否被检查组使用了
 - 如果使用了, 则要报错
 - 没使用, 则可以删除
4. CheckItemDao
 - 提供根据检查项的id查询检查组与检查项关系表的个数
 - 提供通过id删除检查项

【讲解】

3.1. 前台代码

为了防止用户误操作, 点击删除按钮时需要弹出确认删除的提示, 用户点击取消则不做任何操作, 用户点击确定按钮再提交删除请求。

3.1.1. 绑定单击事件

需要为删除按钮绑定单击事件, 并且将当前行数据作为参数传递给处理函数

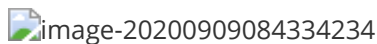
```
1 | <el-button size="mini" type="danger" @click="handleDelete(scope.row)">删除
   | </el-button>
```

调用的方法

```
1 | // 删除
2 | handleDelete(row) {
3 |     // 测试一下，弹出要删除的检查项的id，这里的row即为要删除的检查项整行记录
4 |     alert(row.id);
5 | }
```

3.1.2. 弹出确认操作提示

用户点击删除按钮会执行handleDelete方法，此处需要完善handleDelete方法，弹出确认提示信息。ElementUI提供了\$confirm方法来实现确认提示信息弹框效果



把它的源码复制到我们的handleDelete方法中，运行测试一下效果

```
1 | // 删除
2 | handleDelete(row) {
3 |     // alert(row.id);
4 |     this.$confirm('此操作将永久删除该文件，是否继续?', '提示', {
5 |         confirmButtonText: '确定',
6 |         cancelButtonText: '取消',
7 |         type: 'warning'
8 |     }).then(() => {
9 |         this.$message({
10 |             type: 'success',
11 |             message: '删除成功!'
12 |         });
13 |     }).catch(() => {
14 |         this.$message({
15 |             type: 'info',
16 |             message: '已取消删除'
17 |         });
18 |     });
19 | }
```

3.1.3. 发送请求

修改handleDelete方法

如果用户点击确定按钮就需要发送ajax请求，并且将当前检查项的id作为参数提交到后台进行删除操作

```
1 | // 删除
2 | handleDelete(row) {
3 |     // row行数据，数据库中的一条记录，checkitem实体对象
4 |     // 获取删除的id
5 |     var id = row.id;
6 |     //alert(JSON.stringify(row));
7 |     this.$confirm('此操作将【永久删除】该检查项，是否继续?', '提示', {
8 |         confirmButtonText: '确定',
```

```

9         cancelButtonText: '取消',
10        type: 'warning'
11    }).then(() => {
12        // 点击确定后调用
13        axios.post('/checkitem/deleteById.do?id=' + id).then(res=>{
14            this.$message({
15                message: res.data.message,
16                type: res.data.flag?"success":"error"
17            })
18            if(res.data.flag){
19                // 成功
20                // 刷新列表数据
21                this.findPage();
22            }
23        })
24    }).catch(() => {
25        // 点击取消后调用
26        // 空着，防止报错
27    });
28 }

```

3.2. 后台代码

3.2.1 自定义异常与处理

检查项不能随便删除，被检查组使用了的检查项不能删除，否则检查组将不完整。因此在删除前要判断一下是否被检查组使用了。如果被使用了，这时我们要终止代码的继续执行。那我们就要用到自定义异常。

自定义异常的好处：

1. 区分业务异常与系统异常
2. 友好提示
3. 终止已知不符合业务逻辑代码的断续执行

3.2.1.1 自定义异常

在health_common工程中创建

```

1 package com.itheima.health.exception;
2
3 /**
4  * Description: 自定义异常
5  * 友好提示
6  * 区分系统与自定义的异常
7  * 终止已经不符合业务逻辑的代码
8  * User: Eric
9  */
10 public class HealthException extends RuntimeException{
11     public HealthException(String message){
12         super(message);
13     }
14 }
15

```

3.2.1.2 全局异常处理

在health_web工程的controller中添加 全局异常处理类 HealExceptionAdvice

```
1 package com.itheima.health.controller;
2
3 import com.itheima.health.entity.Result;
4 import com.itheima.health.exception.HealthException;
5 import org.springframework.web.bind.annotation.ExceptionHandler;
6 import org.springframework.web.bind.annotation.RestControllerAdvice;
7
8 /**
9  * Description: 全局异常处理
10  * Advice: 通知
11  * User: Eric
12  * ExceptionHandler 获取的异常 异常的范围从小到大, 类似于try catch中的catch
13  * 与前端约定好的, 返回的都是json数据
14  */
15 @RestControllerAdvice
16 public class HealExceptionAdvice {
17
18     /**
19      * info: 打印日志, 记录流程性的内容
20      * debug: 记录一些重要的数据 id, orderId, userId
21      * error: 记录异常的堆栈信息, 代替e.printStackTrace();
22      * 工作中不能有System.out.println(), e.printStackTrace();
23      */
24     private static final Logger log =
25         LoggerFactory.getLogger(HealExceptionAdvice.class);
26
27     /**
28      * 自定义抛出的异常处理
29      * @param he
30      * @return
31      */
32     @ExceptionHandler(HealthException.class)
33     public Result handleHealthException(HealthException he){
34         // 我们自己抛出的异常, 把异常信息包装下返回即可
35         return new Result(false, he.getMessage());
36     }
37
38     /**
39      * 所有未知的异常处理
40      * @param he
41      * @return
42      */
43     @ExceptionHandler(Exception.class)
44     public Result handleException(Exception he){
45         log.error("发生异常", e);
46         return new Result(false, "发生未知错误, 操作失败, 请联系管理员");
47     }
48 }
```

3.2.2. Controller

在CheckItemController中增加删除方法

```

1  /**
2   * 删除
3   */
4  @PostMapping("/deleteById")
5  public Result deleteById(int id){
6      // 调用业务服务
7      //try {
8          checkItemService.deleteById(id);
9      //} catch (Exception e) {
10         // e.printStackTrace();
11     //}
12     // 响应结果
13     return new Result(true, MessageConstant.DELETE_CHECKITEM_SUCCESS);
14 }

```

3.2.3. 服务接口

在CheckItemService服务接口中扩展删除方法

注意：接口方法上要加上异常的抛出声明，否则health_web的controller在dubbo的反序列化中将无法返回服务提供方抛出的异常，会把服务实现类抛出的HealthException包装成RuntimeException，那么在HealthExceptionAdvice中将无法通过handleHealthException方法来捕获，而被handleException捕获

```

1  void delete(Integer id) throws HealthException;

```

3.2.4. 服务实现类

注意：不能直接删除，需要判断当前检查项是否和检查组关联，如果已经和检查组进行了关联则不允许删除

```

1  /**
2   * 删除
3   * @param id
4   */
5  @Override
6  public void deleteById(int id) throws HealthException {
7      //先判断这个检查项是否被检查组使用了
8      //调用dao查询检查项的id是否在t_checkgroup_checkitem表中存在记录
9      int cnt = checkItemDao.findCountByCheckItemId(id);
10     //被使用了则不能删除
11     if(cnt > 0){
12         // 已经被检查组使用了，则不能删除，报错
13         //??? health_web能捕获到这个异常吗?
14         throw new HealthException(MessageConstant.CHECKITEM_IN_USE);
15     }
16     //没使用就可以调用dao删除
17     checkItemDao.deleteById(id);
18 }

```

3.2.5. Dao接口

在CheckItemDao接口中扩展方法findCountByCheckItemId和deleteById

```

1  /**
2   * 检查 检查项是否被检查组使用了
3   * @param id
4   * @return
5   */
6  int findCountByCheckItemId(int id);
7
8  /**
9   * 通过id删除检查项
10  * @param id
11  */
12 void deleteById(int id);

```

3.2.6. Mapper映射文件

在CheckItemDao.xml中扩展SQL语句

```

1  <select id="findCountByCheckItemId" parameterType="int" resultType="int">
2      select count(1) from t_checkgroup_checkitem where checkitem_id=#{id}
3  </select>
4
5  <delete id="deleteById" parameterType="int">
6      delete from t_checkitem where id=#{id}
7  </delete>

```

【小结】

1. 企业开发分为2种：物理删除（删除表中的数据），逻辑删除(多)(更新表的字段status,state,active为失效, 代表着这条记录不再使用)
2. 自定义异常 作用
3. 全局异常处理@RestControllerAdvice, @ExceptionHandler, 必须进入容器
4. 删除时要询问提示一下，防止用户操作失误
5. 在dubbo的架构中，服务端抛出自定义异常要在接口的方法中声明抛出，否则会被dubbo转成RuntimeException

4. 编辑检查项

【目标】

- 1: 编辑检查项（ID查询，回显）
- 2: 编辑检查项（更新保存，执行update）

【路径】

1. 回显数据
 - 编辑 时弹出【编辑】的窗口 dialogFormVisiable4Edit=true, 重置表单
 - 获取编辑的id, 发送请求通过id查询检查项信息, 把返回的结果绑定到formData里, 如果失败则要提示
 - CheckItemController->service-dao提供通过id查询检查项
2. 提交修改

- 确定【注意】编辑窗口的确定，form表单校验通过后，提交formData给后台，提示操作的结果。如果成功则要关闭编辑的窗口，刷新列表数据
- CheckItemController 用 checkItem接收formData，调用服务更新，返回结果给页面
- CheckItemService -> dao, 提供更新的方法

【讲解】

4.1. 前台代码

用户点击编辑按钮时，需要弹出编辑窗口并且将当前记录的数据进行回显，用户修改完成后点击确定按钮将修改后的数据提交到后台进行数据库操作。

4.1.1. 绑定单击事件

需要为编辑按钮绑定单击事件，并且将当前行数据作为参数传递给处理函数

```
1 <el-button type="primary" size="mini" @click="handleUpdate(scope.row)">编辑
  </el-button>
```

处理事件: handleUpdate();

```
1 // 弹出编辑窗口
2 handleUpdate(row) {
3     alert(row.id);
4 },
```

4.1.2. 弹出编辑窗口回显数据

当前页面中的编辑窗口已经提供好了，默认处于隐藏状态。在handleUpdate方法中需要将编辑窗口展示出来，并且需要发送ajax请求查询当前检查项数据用于回显

```
1 // 弹出编辑窗口
2 handleUpdate(row) {
3     // 重置表单
4     this.resetForm();
5     // 要编辑的检查项id
6     var id = row.id;
7     // 弹出编辑窗口，【注意】：这里是弹出编辑窗口，比新增的窗口的那个变量是多了4Edit字符
    的。
8     this.dialogFormVisible4Edit = true;
9     // 回显 发送请求
10    axios.get('/checkitem/findById.do?id=' + id).then(res => {
11        if(res.data.flag){
12            // 绑定数据回显，编辑的form表单也绑定了这个formData
13            this.formData = res.data.data;
14        }else{
15            this.$message.error(res.data.message);
16        }
17    })
18 },
```

4.1.3. 发送请求更改数据

在编辑窗口中修改完成后，点击确定按钮需要提交请求，所以需要为确定按钮绑定事件并提供处理函数 `handleEdit`

```
1 <div slot="footer" class="dialog-footer">
2   <el-button @click="dialogFormVisible4Edit = false">取消</el-button>
3   <el-button type="primary" @click="handleEdit()">确定</el-button>
4 </div>
```

`handleEdit()`方法

```
1 //编辑提交
2 handleEdit() {
3   // 提交修改，
4   // 【注意】：这里提交的是编辑表单，refs里填的是dataEditForm而不dataAddEditForm，编写时要区分开
5   this.$refs['dataEditForm'].validate((valid) => {
6     if (valid) {
7       // 所有检验通过，提交数据给后台this.formData
8       axios.post('/checkitem/update.do', this.formData).then(res=>{
9         this.$message({
10          message: res.data.message,
11          type: res.data.flag?"success":"error"
12        })
13        if(res.data.flag){
14          // 成功的处理
15          // 关闭窗口。
16          // 【注意】：这里关闭的是编辑的窗口，非添加的窗口
17          this.dialogFormVisible4Edit = false;
18          // 刷新列表数据
19          this.findPage();
20        }
21      })
22    }
23  });
24 },
```

4.2. 后台代码

4.2.1. Controller

在 `CheckItemController` 中增加编辑方法

```
1 /**
2  * 通过id查询
3  */
4 @GetMapping("/findById")
5 public Result findById(int id){
6   CheckItem checkItem = checkItemService.findById(id);
7   return new Result(true,
8     MessageConstant.QUERY_CHECKITEM_SUCCESS, checkItem);
9 }
10 /**
11  * 更新
12  * @param checkItem
```

```

13     * @return
14     */
15     @PostMapping("/update")
16     public Result update(@RequestBody CheckItem checkItem){
17         // 调用服务更新
18         checkItemService.update(checkItem);
19         return new Result(true, MessageConstant.EDIT_CHECKITEM_SUCCESS);
20     }

```

4.2.2. 服务接口

在CheckItemService服务接口中扩展编辑方法

```

1  /**
2   * 通过id查询
3   * @param id
4   * @return
5   */
6  CheckItem findById(int id);
7
8  /**
9   * 更新检查项
10  * @param checkitem
11  */
12  void update(CheckItem checkitem);

```

4.2.3. 服务实现类

在CheckItemServiceImpl实现类中实现编辑方法

```

1  /**
2   * 通过id查询
3   * @param id
4   * @return
5   */
6  @Override
7  public CheckItem findById(int id) {
8      return checkItemDao.findById(id);
9  }
10
11  /**
12   * 更新
13   * @param checkitem
14   */
15  @Override
16  public void update(CheckItem checkitem) {
17      checkItemDao.update(checkitem);
18  }

```

4.2.4. Dao接口

在CheckItemDao接口中扩展edit方法

```

1  /**
2   * 通过id查询
3   * @param id
4   * @return
5   */
6  CheckItem findById(int id);
7
8  /**
9   * 更新检查项
10  * @param checkitem
11  */
12 void update(CheckItem checkitem);

```

4.2.5. Mapper映射文件

在CheckItemDao.xml中扩展SQL语句

```

1  <select id="findById" parameterType="int" resultType="checkitem">
2      select * From t_checkitem where id=#{id}
3  </select>
4
5  <update id="update" parameterType="checkitem">
6      update t_checkitem
7      set
8          code=#{code},
9          name=#{name},
10         sex=#{sex},
11         age=#{age},
12         price=#{price},
13         type=#{type},
14         remark=#{remark},
15         attention=#{attention}
16         where id=#{id}
17 </update>

```

【小结】

- 数据回显
通过id查询，绑定到formData 【注意】弹出的窗口使用的变量是dialogFormVisiableEdit
- 修改后提交 【确定 按钮一定是编辑窗口的】
跟添加类似，表单校验，提交formData

【注意】：数据类型要与实体匹配，价格的输入框的值必须是数字

5. 新增检查组

【需求】

检查组其实就是多个检查项的集合，例如有一个检查组为“一般检查”，这个检查组可以包括多个检查项：身高、体重、收缩压、舒张压等。所以在添加检查组时需要选择这个检查组包括的检查项。

检查组对应的实体类为CheckGroup，对应的数据表为t_checkgroup。检查组和检查项为多对多关系，所以需要中间表t_checkgroup_checkitem进行关联。

业务关系：检查组与检查项 1：多

【目标】

新增检查组

【路径】

1：前台代码

- 新建 弹出新增检查组的窗口this.dialogFormVisible=true, 重置表单。
- 查询所有的检查项信息，绑定到tableData
- 确定 按钮，提交请求，提交formData, checkitemIds，提示操作的结果，如果成功则关闭新增窗口，刷新列表数据

2：后台处理

- CheckGroupController add用CheckGroup接收formData, 用Integer[] 接收checkitemIds。调用服务添加检查组，返回结果给页面
- CheckGroupService与实现类
 - 添加检查组
 - 获取新增的检查组的id
 - 遍历选中的检查项id
 - 添加检查组与检查项的关系
 - 事务控制
- CheckGroupDao与映射文件
 - 添加检查组 insert into 可以获取新增的
keyProperty="id" userGenerateKey=true
 - 添加检查组与检查项

【讲解】

5.1. 前台代码

检查组管理页面对应的是checkgroup.html页面，根据产品设计的原型已经完成了页面基本结构的编写，现在需要完善页面动态效果。

5.1.1. 弹出新增窗口

页面中已经提供了新增窗口，只是出于隐藏状态。只需要将控制展示状态的属性dialogFormVisible改为true即可显示出新增窗口。点击新建按钮时绑定的方法为handleCreate，所以在handleCreate方法中修改dialogFormVisible属性的值为true即可。同时为了增加用户体验度，需要每次点击新建按钮时清空表单输入项。

由于新增检查组时还需要选择此检查组包含的检查项，所以新增检查组窗口分为两部分信息：基本信息和检查项信息，如下图：



image-20200622081030498

(1) 新建按钮绑定单击事件，对应的处理函数为handleCreate

```
1 <el-button type="primary" class="butt" @click="handleCreate()">新建</el-button>
```

(2) handleCreate()方法

```
1 // 重置表单
2 resetForm() {
3   // 清空表单
4   this.formData = {};
5   this.activeName='first'; //选中基本信息标签项
6   // 清除勾选
7   this.checkitemIds=[];
8 },
9 // 弹出添加窗口
10 handleCreate() {
11   //重置表单，弹出窗口
12   this.resetForm();
13   this.dialogFormVisible = true;
14 },
```

5.1.2. 新增窗口中，动态展示检查项列表

现在虽然已经完成了新增窗口的弹出，但是在检查项信息标签页中需要动态展示所有的检查项信息列表数据，并且可以进行勾选。具体操作步骤如下：

(1) 定义模型数据

```
1 tableData: [], //新增和编辑表单中对应的检查项列表数据
2 checkitemIds: [], //新增和编辑表单中检查项对应的复选框，基于双向绑定可以进行回显和数据提交，传递检查项id的数组
```

(2) 动态展示检查项列表数据，数据来源于上面定义的tableData模型数据

```
1 <el-tab-pane label="检查项信息" name="second">
2   <div class="checkScrol">
3     <table class="datatable">
4       <thead>
5         <tr>
6           <th>选择</th>
7           <th>项目编码</th>
8           <th>项目名称</th>
9           <th>项目说明</th>
10        </tr>
11      </thead>
12      <tbody>
13        <tr v-for="c in tableData">
14          <td>
15            <input :id="c.id" v-model="checkitemIds"
16              type="checkbox" :value="c.id">
17            <td><label :for="c.id">{{c.code}}</label></td>
18            <td><label :for="c.id">{{c.name}}</label></td>
19            <td><label :for="c.id">{{c.remark}}</label></td>
20          </tr>
21        </tbody>
```

```

22     </table>
23 </div>
24 </el-tab-pane>

```

(3) 完善handleCreate方法，发送ajax请求查询所有检查项数据并将结果赋值给tableData模型数据用于页面表格展示

```

1  // 弹出添加窗口
2  handleCreate() {
3      //重置表单，弹出窗口
4      this.resetForm();
5      this.dialogFormVisible = true;
6      //发送请求后台获取所有检查项数据，得到后绑定tableData(检查项列表)
7      axios.get('/checkitem/findAll.do').then(res => {
8          if(res.data.flag){
9              this.tableData = res.data.data;
10         }else{
11             this.$message.error(res.data.message);
12         }
13     })
14 },

```

(4) 分别在CheckItemController、CheckItemService、CheckItemServiceImpl、CheckItemDao、CheckItemDao.xml中扩展方法查询所有检查项数据

【1】：CheckItemController:

```

1  @GetMapping("/findAll")
2  public Result findAll(){
3      // 调用服务查询所有的检查项
4      List<CheckItem> list = checkItemService.findAll();
5      // 封装返回的结果
6      return new Result(true, MessageConstant.QUERY_CHECKITEM_SUCCESS,list);
7  }

```

【2】：CheckItemService:

```

1  List<CheckItem> findAll();

```

【3】：CheckItemServiceImpl:

```

1  public List<CheckItem> findAll() {
2      return checkItemDao.findAll();
3  }

```

【4】：CheckItemDao:

```

1  List<CheckItem> findAll();

```

【5】：CheckItemDao.xml:

```

1 <select id="findAll" resultType="CheckItem">
2     select * from t_checkitem
3 </select>

```

5.1.3. 提交请求，执行保存

当用户点击新增窗口中的确定按钮时发送ajax请求将数据提交到后台进行数据库操作。提交到后台的数据分为两部分：检查组基本信息（对应的模型数据为formData）和检查项id数组（对应的模型数据为checkitemIds）。

(1) 为确定按钮绑定单击事件，对应的处理函数为handleAdd

```

1 <el-button type="primary" @click="handleAdd()">确定</el-button>

```

(2) 完善handleAdd方法

```

1 //添加
2 handleAdd () {
3     //提交检查组信息this.formData，选中的检查项id this.checkitemIds
4     axios.post('/checkgroup/add.do?checkitemIds=' + this.checkitemIds,
5     this.formData).then(res => {
6         this.$message({
7             message: res.data.message,
8             type: res.data.flag?"success":"error"
9         })
10        if(res.data.flag){
11            // 关闭窗口
12            this.dialogFormVisible = false;
13            // 刷新列表数据
14            this.findPage();
15        }
16    })
17 },

```

5.2. 后台代码

2.2.1. Controller

在health_web工程中创建CheckGroupController

```

1 package com.itheima.health.controller;
2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.itheima.health.constant.MessageConstant;
5 import com.itheima.health.entity.PageResult;
6 import com.itheima.health.entity.QueryPageBean;
7 import com.itheima.health.entity.Result;
8 import com.itheima.health.pojo.CheckGroup;
9 import com.itheima.health.service.CheckGroupService;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.RequestBody;
12 import org.springframework.web.bind.annotation.RequestMapping;

```

```

13 import org.springframework.web.bind.annotation.RestController;
14
15 /**
16  * Description: No Description
17  * User: Eric
18  */
19 @RestController
20 @RequestMapping("/checkgroup")
21 public class CheckGroupController {
22
23     @Reference
24     private CheckGroupService checkGroupService;
25
26     /**
27      * 添加检查组
28      * @param checkGroup
29      * @param checkitemIds
30      * @return
31      */
32     @PostMapping("/add")
33     public Result add(@RequestBody CheckGroup checkGroup, Integer[]
checkitemIds){
34         // 调用业务服务
35         checkGroupService.add(checkGroup, checkitemIds);
36         // 响应结果
37         return new Result(true, MessageConstant.ADD_CHECKGROUP_SUCCESS);
38     }
39 }

```

5.2.2. 服务接口

在health_interface工程中创建CheckGroupService接口

```

1 package com.itheima.health.service;
2
3 import com.itheima.health.entity.PageResult;
4 import com.itheima.health.entity.QueryPageBean;
5 import com.itheima.health.pojo.CheckGroup;
6
7 /**
8  * Description: No Description
9  * User: Eric
10  */
11 public interface CheckGroupService {
12     /**
13      * 添加检查组
14      * @param checkGroup
15      * @param checkitemIds
16      */
17     void add(CheckGroup checkGroup, Integer[] checkitemIds);
18 }

```

5.2.3. 服务实现类

在health_service工程中创建CheckGroupServiceImpl实现类

```
1 package com.itheima.health.service.impl;
2
3 import com.alibaba.dubbo.config.annotation.Service;
4 import com.github.pagehelper.Page;
5 import com.github.pagehelper.PageHelper;
6 import com.itheima.health.dao.CheckGroupDao;
7 import com.itheima.health.entity.PageResult;
8 import com.itheima.health.entity.QueryPageBean;
9 import com.itheima.health.pojo.CheckGroup;
10 import com.itheima.health.service.CheckGroupService;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.transaction.annotation.Transactional;
13 import org.springframework.util.StringUtils;
14
15 /**
16  * Description: No Description
17  * User: Eric
18  */
19 @Service(interfaceClass = CheckGroupService.class)
20 public class CheckGroupServiceImpl implements CheckGroupService {
21
22     @Autowired
23     private CheckGroupDao checkGroupDao;
24
25     /**
26      * 添加
27      * @param checkGroup
28      * @param checkitemIds
29      */
30     @Override
31     @Transactional
32     public void add(CheckGroup checkGroup, Integer[] checkitemIds) {
33         // 添加检查组
34         checkGroupDao.add(checkGroup);
35         // 获取检查组的id
36         Integer checkGroupId = checkGroup.getId();
37         // 遍历检查项id, 添加检查组与检查项的关系
38         if(null != checkitemIds){
39             // 有勾选
40             for (Integer checkitemId : checkitemIds) {
41                 //添加检查组与检查项的关系
42                 checkGroupDao.addCheckGroupCheckItem(checkGroupId,
43                     checkitemId);
44             }
45         }
46     }
47 }
```

2.4. Dao接口

在health_dao工程中创建CheckGroupDao接口

```
1 package com.itheima.health.dao;
2
```

```

3  import com.github.pagehelper.Page;
4  import com.itheima.health.pojo.CheckGroup;
5  import org.apache.ibatis.annotations.Param;
6
7  /**
8   * Description: No Description
9   * User: Eric
10  */
11 public interface CheckGroupDao {
12     /**
13      * 添加检查组
14      * @param checkGroup
15      */
16     void add(CheckGroup checkGroup);
17
18     /**
19      * 添加检查组与检查项的关系
20      * @param checkGroupId 注意要取别名，类型相同
21      * @param checkItemId
22      */
23     void addCheckGroupCheckItem(@Param("checkGroupId") Integer checkGroupId,
24                                  @Param("checkItemId") Integer checkItemId);
25 }

```

上面我们使用@Param区别两个参数类型相同的参数

5.2.5. Mapper映射文件

在health_dao工程中创建CheckGroupDao.xml映射文件，需要和CheckGroupDao接口在同一目录下

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6  <mapper namespace="com.itheima.health.dao.CheckGroupDao">
7      <insert id="add" parameterType="checkgroup">
8          <selectKey resultType="int" order="AFTER" keyProperty="id">
9              select last_insert_id()
10          </selectKey>
11          insert into t_checkgroup (code,name,helpCode,sex,remark,attention)
12          values (#{code},#{name},#{helpCode},#{sex},#{remark},#{attention})
13      </insert>
14
15      <insert id="addCheckGroupCheckItem" parameterType="int">
16          insert into t_checkgroup_checkitem (checkgroup_id, checkitem_id)
17          values (#{checkGroupId}, #{checkItemId})
18      </insert>
19  </mapper>

```

【小结】

1: 前台代码

(1) 弹出新增窗口 this.dialogFormVisable=tur

- 使用选项卡
- 选项卡一：检查组信息 放到form保存及展示 检查组信息
- 选项卡二：检查项列表，并提供复选框 供给用户来选择

(2) 新增窗口中，动态展示检查项列表

- 查询所有检查项，弹出 窗口时，加载检查项列表数据 绑定到this.tableData

(3) 提交请求(检查组信息this.formData, 选中的检查项id this.checkitemIds)，执行保存，关闭窗口且刷新列表

2: 后台代码

1. Controller用@RequestBody(只能有一个)前端传过来form表单数据=>CheckGroup
2. Integer[] 数组来接收选中的检查项id数组 (提交是字符串，后台拿到的是数组，springMVC)
3. 业务服务中，对多数据操作，开启事务 (注意)
4. 插入数据后获取id使用 selectKey select last_insert_id()或调协keyProperty useGenerateKey=true

优化：添加检查组与检查项关系使用的for循环，效率比较，使用批量的方式提交

知识点：关系维护

基本信息，列表来勾选(关系), 只提交选中的id。

6. 检查组分页查询

【目标】

检查组查询、分页

【路径】

1: 前台代码

- (1) 定义分页相关模型数据pagination{页码，大小，查询的条件}
- (2) 定义分页方法 findPage
- (3) 完善分页方法执行时机 (点击“查询”，点击“页码”)

2: 后台代码

执行

- 检查组分页查询

- (1) CheckGroupController.java (Controller) result {flag,message,data:pageResult{total,rows}}
- (2) CheckGroupService.java (服务接口)
- (3) CheckGroupServiceImpl.java (服务实现类) 使用 pagehelper 返回分页的结果
- (4) CheckGroupDao.java (Dao接口)
- (5) CheckGroupDao.xml (Mapper映射文件)

【讲解】

6.1. 前台代码

6.1.1. 定义分页相关模型数据

```
1 pagination: { //分页相关模型数据
2   currentPage: 1, //当前页码
3   pageSize: 10, //每页显示的记录数
4   total: 0, //总记录数
5   queryString: null //查询条件
6 },
7 dataList: [], //当前页要展示的分页列表数据
```

6.1.2. 定义分页方法

在页面中提供了findPage方法用于分页查询，为了能够在checkgroup.html页面加载后直接可以展示分页数据，可以在VUE提供的钩子函数created中调用findPage方法

```
1 //钩子函数，VUE对象初始化完成后自动执行
2 created() {
3   this.findPage();
4 },
```

findPage()方法。

```
1 //分页查询
2 findPage() {
3   axios.post('/checkgroup/findPage.do', this.pagination).then(res => {
4     if(res.data.flag){
5       this.dataList = res.data.data.rows;
6       this.pagination.total = res.data.data.total;
7     } else{
8       this.$message.error(res.data.message);
9     }
10   });
11 },
```

6.1.3. 完善分页方法执行时机

除了在created钩子函数中调用findPage方法查询分页数据之外，当用户点击查询按钮或者点击分页条中的页码时也需要调用findPage方法重新发起查询请求。

(1) 为查询按钮绑定单击事件，调用handleCurrentChange方法，查询后回到第一页。

```
1 <el-button @click="handleCurrentChange(1)" class="da1fBut">查询</el-button>
```

(2) 为分页条组件绑定current-change事件，此事件是分页条组件自己定义的事件，当页码改变时触发，对应的处理函数为handleCurrentChange

```

1 <div class="pagination-container">
2     <el-pagination
3         class="pagiantion"
4         @current-change="handleCurrentChange"
5         :current-page="pagination.currentPage"
6         :page-size="pagination.pageSize"
7         layout="total, prev, pager, next, jumper"
8         :total="pagination.total">
9     </el-pagination>
10 </div>

```

(3) 定义handleCurrentChange方法

```

1 //切换页码
2 handleCurrentChange(currentPage) {
3     //currentPage为切换后的页码
4     this.pagination.currentPage = currentPage;
5     this.findPage();
6 },

```

6.2. 后台代码

6.2.1. Controller

在CheckGroupController中增加分页查询方法

```

1 /**
2  * 分页条件查询
3  */
4 @PostMapping("/findPage")
5 public Result findPage(@RequestBody QueryPageBean queryPageBean){
6     // 调用业务查询
7     PageResult<CheckGroup> pageResult =
8     checkGroupService.findPage(queryPageBean);
9     return new Result(true, MessageConstant.QUERY_CHECKGROUP_SUCCESS,
10     pageResult);
11 }

```

6.2.2. 服务接口

在CheckGroupService服务接口中扩展分页查询方法

```

1 /**
2  * 分页条件查询
3  * @param queryPageBean
4  * @return
5  */
6 PageResult<CheckGroup> findPage(QueryPageBean queryPageBean);

```

6.2.3. 服务实现类

在CheckGroupServiceImpl服务实现类中实现分页查询方法，基于Mybatis分页助手插件实现分页

```

1  /**
2   * 分页条件查询
3   * @param queryPageBean
4   * @return
5   */
6  @Override
7  public PageResult<CheckGroup> findPage(QueryPageBean queryPageBean) {
8      // 使用PageHelper.startPage
9      PageHelper.startPage(queryPageBean.getCurrentPage(),
10         queryPageBean.getPageSize());
11      // 有查询条件的处理，模糊查询
12      if(!StringUtils.isEmpty(queryPageBean.getQueryString())){
13          // 拼接%
14          queryPageBean.setQueryString("%" + queryPageBean.getQueryString()+
15             "%");
16      }
17      // 紧接着的查询会被分页
18      Page<CheckGroup> page =
19         checkGroupDao.findByCondition(queryPageBean.getQueryString());
20      return new PageResult<CheckGroup>(page.getTotal(), page.getResult());
21  }

```

6.2.4. Dao接口

在CheckGroupDao接口中扩展分页查询方法

```

1  Page<CheckGroup> findByCondition(String queryString);

```

6.2.5. Mapper映射文件

在CheckGroupDao.xml文件中增加SQL定义

```

1  <select id="findByCondition" resultType="checkgroup" parameterType="String">
2      select * From t_checkgroup
3      <if test="value !=null and value.length > 0">
4          where code like #{value} or name like #{value} or helpCode like #
5          {value}
6      </if>
7  </select>

```

【小结】

1: 前台代码

- (1) 定义分页相关模型数据
- (2) 定义分页方法
- (3) 完善分页方法执行时机（点击“查询”，点击“分页”）

2: 后台代码

- (1) CheckGroupController.java (Controller)
- (2) CheckGroupService.java (服务接口)

(3) CheckGroupServiceImpl.java (服务实现类)

(4) CheckGroupDao.java (Dao接口)

(5) CheckGroupDao.xml (Mapper映射文件)

7. 编辑检查组

【目标】

编辑检查组

【路径】

1. 回显

- 编辑 获取检查组的id, 发送请求通过id查询检查组信息, 绑定formData
- 发送请求查询所有检查项列表, 绑定tableData, 供用户选择
- 再次查询, 通过检查组id查询选中的检查项id集合

2. 提交修改

- 提交formData, checkitemIds, 提示操作的结果, 成功, 则要关闭编辑 的窗口, 刷新列表
- CheckGroupController 用 checkgroup接收formData, integer[] 接收checkitemIds, 调用服务, 返回操作结果给页面
- CheckgroupService
 - 先更新检查组信息
 - 删检查组与检查项的关系
 - 添加新关系
 - 事务控制
- CheckGroupDao与映射文件
 - 更新检查组
 - 删除检查组与检查项的关系

【讲解】

7.1. 前台页面

用户点击编辑按钮时, 需要弹出编辑窗口并且将当前记录的数据进行回显, 用户修改完成后点击确定按钮将修改后的数据提交到后台进行数据库操作。此处进行数据回显的时候, 除了需要检查组基本信息的回显之外, 还需要回显当前检查组包含的检查项 (以复选框勾选的形式回显)。

7.1.1. 绑定单击事件

- (1) 需要为编辑按钮绑定单击事件, 并且将当前行数据作为参数传递给处理函数

```

1 <el-table-column label="操作" align="center">
2   <template slot-scope="scope">
3     <el-button type="primary" size="mini"
@click="handleUpdate(scope.row)">编辑</el-button>
4     <el-button size="mini" type="danger"
@click="handleDelete(scope.row)">删除</el-button>
5   </template>
6 </el-table-column>

```

(2) handleUpdate事件

```

1 // 弹出编辑窗口
2 handleUpdate(row) {
3   alert(row.id);
4 },

```

7.1.2. 弹出编辑窗口回显数据

当前页面的编辑窗口已经提供好了，默认处于隐藏状态。在handleUpdate方法中需要将编辑窗口展示出来，并且需要发送多个ajax请求分别查询当前检查组数据、所有检查项数据、当前检查组包含的检查项id用于基本数据回显

```

1 // 重置表单
2 resetForm() {
3   // 清空表单
4   this.formData = {};
5   this.activeName='first'; //选中基本信息标签项
6   // 清除勾选
7   this.checkitemIds=[];
8 },

```

```

1 // 弹出编辑窗口
2 handleUpdate(row) {
3   this.resetForm();
4   // 弹出编辑窗口 【注意】这里的变量是多了4Edit的，是控制编辑窗口展示的变量。
5   this.dialogFormVisible4Edit = true;
6   // 获取检查组的id
7   var checkGroupId = row.id;
8   axios.get("/checkgroup/findById.do?checkGroupId=" +
checkGroupId).then(res =>{
9     if(res.data.flag){
10       // 成功绑定数据
11       this.formData = res.data.data;
12       //发送请求后台获取所有检查项数据，得到后绑定tableData(检查项列表)
13       axios.get('/checkitem/findAll.do').then(resp => {
14         if(resp.data.flag){
15           this.tableData = resp.data.data;
16           // 获取选中的检查项id，回显数据时，检查项列表要勾选中
17
18       axios.get("/checkgroup/findCheckItemIdsByCheckGroupId.do?checkGroupId=" +
checkGroupId).then(response=>{

```



```

18 // 后台要返回data必须为List
19 if(response.data.flag){
20     this.checkitemIds = response.data.data;
21 }else{
22     this.$message.error(response.data.message);
23 }
24 })
25 }else{
26     this.$message.error(resp.data.message);
27 }
28 })
29 }else{
30     this.$message.error(res.data.message);
31 }
32 })
33 },

```

7.1.3. 发送请求，编辑保存检查组

(1) 在编辑窗口中修改完成后，点击确定按钮需要提交请求，所以需要为确定按钮绑定事件并提供处理函数handleEdit

```

1 | <el-button type="primary" @click="handleEdit()">确定</el-button>

```

(2) handleEdit()方法

```

1 //编辑 修改后提交
2 handleEdit() {
3     //提交检查组信息this.formData, 选中的检查项id this.checkitemIds
4     axios.post('/checkgroup/update.do?checkitemIds=' + this.checkitemIds,
5     this.formData).then(res => {
6         this.$message({
7             message: res.data.message,
8             type: res.data.flag?"success":"error"
9         })
10        if(res.data.flag){
11            // 【注意】关闭的是编辑窗口
12            this.dialogFormVisible4Edit = false;
13            // 刷新列表数据
14            this.findPage();
15        }
16    })
17 },

```

7.2. 后台代码

7.2.1. Controller

在CheckGroupController中增加方法

```

1 /**
2  * 通过id获取检查组
3  */

```

```

4  @GetMapping("/findById")
5  public Result findById(int checkGroupId){
6      // 调用业务服务
7      CheckGroup checkGroup = checkGroupService.findById(checkGroupId);
8      return new Result(true,
9      MessageConstant.QUERY_CHECKGROUP_SUCCESS, checkGroup);
10 }
11 /**
12  * 通过检查组id查询选中的检查项id
13  */
14 @GetMapping("/findCheckItemIdsByCheckGroupId")
15 public Result findCheckItemIdsByCheckGroupId(int checkGroupId){
16     // 调用服务查询
17     List<Integer> checkItemIds =
18     checkGroupService.findCheckItemIdsByCheckGroupId(checkGroupId);
19     return new Result(true,
20     MessageConstant.QUERY_CHECKITEM_SUCCESS, checkItemIds);
21 }
22 /**
23  * 修改提交
24  */
25 @PostMapping("/update")
26 public Result update(@RequestBody CheckGroup checkGroup, Integer[]
27 checkitemIds){
28     // 调用业务服务
29     checkGroupService.update(checkGroup, checkitemIds);
30     // 响应结果
31     return new Result(true, MessageConstant.EDIT_CHECKGROUP_SUCCESS);
32 }

```

7.2.2. 服务接口

在CheckGroupService服务接口中扩展方法

```

1  /**
2   * 通过检查组id查询选中的检查项id
3   * @param checkGroupId
4   * @return
5   */
6  List<Integer> findCheckItemIdsByCheckGroupId(int checkGroupId);
7
8  /**
9   * 通过id获取检查组
10  * @param checkGroupId
11  * @return
12  */
13  CheckGroup findById(int checkGroupId);
14
15  /**
16  * 修改检查组
17  * @param checkGroup
18  * @param checkitemIds
19  */

```

```
20 void update(CheckGroup checkGroup, Integer[] checkitemIds);
```

7.2.3. 服务实现类

在CheckGroupServiceImpl实现类中实现编辑方法

```
1  /**
2   * 通过检查组id查询选中的检查项id
3   * @param checkGroupId
4   * @return
5   */
6  @Override
7  public List<Integer> findCheckItemIdsByCheckGroupId(int checkGroupId) {
8      return checkGroupDao.findCheckItemIdsByCheckGroupId(checkGroupId);
9  }
10
11  /**
12   * 通过id获取检查组
13   * @param checkGroupId
14   * @return
15   */
16  @Override
17  public CheckGroup findById(int checkGroupId) {
18      return checkGroupDao.findById(checkGroupId);
19  }
20
21  /**
22   * 修改检查组
23   * @param checkGroup
24   * @param checkitemIds
25   */
26  @Override
27  @Transactional
28  public void update(CheckGroup checkGroup, Integer[] checkitemIds) {
29      // 先更新检查组
30      checkGroupDao.update(checkGroup);
31      // 删除旧关系
32      checkGroupDao.deleteCheckGroupCheckItem(checkGroup.getId());
33      // 建立新关系
34      if(null != checkitemIds){
35          for (Integer checkitemId : checkitemIds) {
36              checkGroupDao.addCheckGroupCheckItem(checkGroup.getId(),
37                  checkitemId);
38          }
39      }
```

7.2.4. Dao接口

在CheckGroupDao接口中扩展方法

```
1  /**
2   * 通过检查组id查询选中的检查项id
3   * @param checkGroupId
4   * @return
5   */
```

```

6  List<Integer> findCheckItemIdsByCheckGroupId(int checkGroupId);
7
8  /**
9   * 通过id获取检查组
10  * @param checkGroupId
11  * @return
12  */
13  CheckGroup findById(int checkGroupId);
14
15  /**
16  * 更新检查组
17  * @param checkGroup
18  */
19  void update(CheckGroup checkGroup);
20
21  /**
22  * 删除检查组与检查项的关系
23  * @param id
24  */
25  void deleteCheckGroupCheckItem(Integer id);

```

7.2.5. Mapper映射文件

在CheckGroupDao.xml中扩展SQL语句

```

1
2  <select id="findCheckItemIdsByCheckGroupId" parameterType="int"
resultType="int">
3      select checkitem_id from t_checkgroup_checkitem where
checkgroup_id=#{checkGroupId}
4  </select>
5
6  <select id="findById" parameterType="int" resultType="checkgroup">
7      select * From t_checkgroup where id=#{checkGroupId}
8  </select>
9
10 <update id="update" parameterType="checkgroup">
11     update t_checkgroup
12     set
13         code=#{code},
14         name=#{name},
15         helpCode=#{helpCode},
16         sex=#{sex},
17         remark=#{remark},
18         attention=#{attention}
19     where id=#{id}
20 </update>
21
22 <delete id="deleteCheckGroupCheckItem" parameterType="int">
23     delete from t_checkgroup_checkitem where checkgroup_id=#{id}
24 </delete>

```

【小结】

1. 回显数据，除了要回显检查组信息外，还要去查检查项列表，查询选中的检查项id集合(列表之后)
默认勾选, 把id的集合绑定this.checkitemIds

2. 关系维护前端

```
1 <table class="datatable">
2   <thead>
3     <tr>
4       <th>选择</th>
5       <th>项目编码</th>
6       <th>项目名称</th>
7       <th>项目说明</th>
8     </tr>
9   </thead>
10  <tbody>
11    <tr v-for="c in tableData">
12      <td>
13        <input :id="c.id" v-model="checkitemIds"
14        type="checkbox" :value="c.id">
15      </td>
16      <td><label :for="c.id">{{c.code}}</label></td>
17      <td><label :for="c.id">{{c.name}}</label></td>
18      <td><label :for="c.id">{{c.remark}}</label></td>
19    </tr>
20  </tbody>
</table>
```

3. 提交，对已经关系维护（多对多），先删除再添加新关系

4. 业务层如果多种修改数据库的操作，记得要加事务管理@Transactional打在方法上

8. 删除检查组

【目标】

删除检查组

【路径】

1: 前台代码

- 删除 弹出询问窗口，确定后，获取要删除的id, 提交删除，提示操作的结果，成功则要刷新列表

2: 后台代码

- CheckGroupController接收传递过来的id,调用服务删除,返回结果给页面
- CheckGroupService
 - 判断是否被套餐使用了
 - 使用了则要报错, 接口方法上做异常的抛出声明
 - 没使用
 - 先删除检查组与检查项的关系表
 - 再删除检查组
- CheckGroupDao与映射文件
 - 通过检查组的id统计套餐与检查组关系表的个数
 - 通过id删除检查组

【讲解】

3.1. 前台代码

为了防止用户误操作，点击删除按钮时需要弹出确认删除的提示，用户点击取消则不做任何操作，用户点击确定按钮再提交删除请求。

3.1.1. 绑定单击事件

需要为删除按钮绑定单击事件，并且将当前行数据作为参数传递给处理函数

```
1 | <el-button size="mini" type="danger" @click="handleDelete(scope.row)">删除
   | </el-button>
```

调用的方法

```
1 | // 删除
2 | handleDelete(row) {
3 |     alert(row.id);
4 | }
```

3.1.2. 弹出确认操作 发送删除请求

用户点击删除按钮会执行handleDelete方法，此处需要完善handleDelete方法，弹出确认提示信息。ElementUI提供了\$confirm方法来实现确认提示信息弹框效果，如果用户点击确定按钮就需要发送ajax请求，并且将当前检查组的id作为参数提交到后台进行删除操作

```
1 | // 删除
2 | handleDelete(row) {
3 |     // 获取删除的id
4 |     var id = row.id;
5 |     //alert(JSON.stringify(row));
6 |     this.$confirm('此操作将【永久删除】该检查组，是否继续?', '提示', {
7 |         confirmButtonText: '确定',
8 |         cancelButtonText: '取消',
9 |         type: 'warning'
10 |    }).then(() => {
11 |        // 点击确定后调用
12 |        axios.post('/checkgroup/deleteById.do?id=' + id).then(res=>{
13 |            this.$message({
14 |                message: res.data.message,
15 |                type: res.data.flag?"success":"error"
16 |            })
17 |            if(res.data.flag){
18 |                // 成功
19 |                // 刷新列表数据
20 |                this.findPage();
21 |            }
22 |        })
23 |    }).catch(() => {
24 |        // 点击取消后调用
25 |        // 空着，防止报错
26 |    });
27 | }
```

3.2. 后台代码

3.2.1. Controller

在CheckGroupController中增加删除方法

```
1  /**
2   * 删除检查组
3   * @param id
4   * @return
5   */
6  @PostMapping("/deleteById")
7  public Result deleteById(int id){
8      //调用业务服务删除
9      checkGroupService.deleteById(id);
10     return new Result(true, MessageConstant.DELETE_CHECKGROUP_SUCCESS);
11 }
```

3.2.2. 服务接口

在CheckGroupService服务接口中扩展删除方法

```
1  /**
2   * 删除检查组
3   * @param id
4   * 【注意】这里的异常是我们自己抛出的异常类，不要导错包了(HandlerException是错的)
5   */
6  void deleteById(int id) throws HealthException;
```

3.2.3. 服务实现类

注意：不能直接删除，需要判断当前检查组是否和检查项、套餐关联，如果已经和检查项、套餐进行了关联则不允许删除

```
1  /**
2   * 删除检查组
3   * @param id
4   */
5  @Override
6  @Transactional
7  public void deleteById(int id) throws HealthException {
8      // 检查 这个检查组是否被套餐使用了
9      int count = checkGroupDao.findSetmealCountByCheckGroupId(id);
10     if(count > 0){
11         // 被使用了
12         throw new HealthException(MessageConstant.CHECKGROUP_IN_USE);
13     }
14     // 没有被套餐使用,就可以删除数据
15     // 先删除检查组与检查项的关系
16     checkGroupDao.deleteCheckGroupCheckItem(id);
17     // 删除检查组
18     checkGroupDao.deleteById(id);
19 }
```

3.2.4. Dao接口

在CheckGroupDao接口中扩展方法findSetmealAndCheckGroupCountByCheckGroupId和deleteById

```
1  /**
2   * 通过检查组id查询是否被套餐使用了
3   * @param id
4   * @return
5   */
6  int findSetmealCountByCheckGroupId(int id);
7
8  /**
9   * 删除检查组
10  * @param id
11  */
12 void deleteById(int id);
```

3.2.5. Mapper映射文件

在CheckGroupDao.xml中扩展SQL语句

```
1  <select id="findSetmealCountByCheckGroupId" parameterType="int"
2     resultType="int">
3     select count(1) from t_setmeal_checkgroup where checkgroup_id=#{id}
4 </select>
5
6  <delete id="deleteById" parameterType="int">
7     delete from t_checkgroup where id=#{id}
8 </delete>
```

【小结】


1. 删除前要判断是否被套餐使用了，业务关系使然，业务：1个套餐下多个检查组，1个检查组下多个检查项
2. 删除主表数据(t_checkgroup)，就先删除从表(t_checkgroup_checkitem)数据
3. 注意事务，抛出的异常一定要在接口中声明，异常类名称不要写成HandlerException

事务问题

AnnotationBean.postProcessAfterInitialization 在dubbo包下的

1 开启事务控制的注解支持
2 注意：此处必须加入`proxy-target-class="true"`，
3 需要进行事务控制，会由Spring框架产生代理对象，
4 Dubbo需要将Service发布为服务，要求必须使用cglib创建代理对象。
5 如果没 `proxy-target-class="true"`，业务实现类/方法加上`@Transaction`，类创建的方式为
jdk动态代理，发布服务时，类名与接口包名不一致，所以不发布服务
6 加上`proxy-target-class="true"`，业务实现类/方法加上`@Transaction`，类的创建方式为
SpringProxy(CGLIB)，找`@Service`，看有接口申明(`interfaceClass=`)
7 如果有接口声明，则发布的服务接口为声明的接口，
8 如果没有的情况下则发布的服务接口为`org.springframework.aop.SpringProxy`
9 那么服务的消费者(controller 找的是业务服务接口)没有提供者
10
11 解决事务导致找不到服务的问题
12 `proxy-target-class="true"` 同时service实现类上加上`@Service(interfaceClass=接口`
类字节码)

1587788820602

1587788834530