day49-传智健康第二天

学习目标

- [ ] 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
- [ ] 掌握编辑检查组实现过程

第一章 - 检查项模块

案例-检查项分页

1. 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

2. 路径

1：前台代码

在checkitem.html给created钩子函数调用findPage方法，findPage方法，发送请求，提交this.pagination，结果处理。失败则要提示，成功： 绑定数据（this.dataList分页结果集，this.pagination.total=总记录数）

2: 后台代码

CheckItemController

提供findPage方法，使用QueryPageBean接收pagination对象，调用业务查询，返回的结果PageResult,包装到Result里，再返回给页面

CheckItemService接口与实现类

提供findPage方法：

```
1. 判断是否有查询条件，如果有查询条件则要使用模糊查询，则查询的条件就要拼接%，再调用dao查询

2. 使用PageHelper来查询
   * PageHelper.startPage(页码，大小)
   * 条件查询即可,返回Page对象
   * 分页结果就是page.getResult
   * 总记录数page.getTotal就可以了
3. 封装到PageResult返回给controller
```

CheckItemDao与映射文件

```
1. 提供条件查询

2. sql语句使用<if>条件判断
```

3. 讲解

1. 点击左侧的 菜单 ， 会发起请求执行分页查询（调用findPage方法） ， 此时不带任何的过滤（筛选）条件，面对的是整张表的数据
2. 点击顶部的查询按钮，也会发起请求执行分页查询（调用findPage方法），但是这种方式有可能携带了查询的参数（过滤的条件） ， 当然也可以不带。不管有没有输入内容，都需要把查询条件（输入框的内容传递给后台的服务器）
3. 在后台的dao里面，就要对这个查询参数进行判定 ， 就需要使用到mybatis里面的动态Sql。判定如果有查询的条件，那么就追加where条件，没有查询的条件，就不用追加。

3.1. 前台代码

3.1.1. 定义分页相关模型数据

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

3.1.2. 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

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

```
//分页查询
findPage() {

    //1. 发起请求，由于左侧的菜单和顶部的查询按钮，都会执行这个findPage方法，为了能够囊括更多的内容
    // 所以我们在这里要携带上 查询的条件，由后台去判定这个查询条件到底有还是没有！
    console.log("查询分页的参数---before---:");
    console.log(this.pagination);
    console.log("查询分页的参数---after---:");

    axios.post("/checkitem/findPage.do" , this.pagination).then(response=>{
       if(response.data.flag){
           //this.$message.success("查询分页成功！");
           console.log(response);
           /*
               response.data =  {data: {…}, flag: true, message: "查询检查项成功"}
               response表示响应的对象，包含所有响应的内容
```

```
                response.data :表示取到服务器返回回来的数据，其实就是取到后台返回的Result对象
                response.data.data 表示取到reslut对象里面设置数据的内容 其实就是取到了PageResult对象
                response.data.data.total : 表示取到pageResult对象的总记录数数据
            */
            this. pagination.total= response.data.data.total;
            this.dataList = response.data.data.rows;
        } else{
            this.$message.success("查询分页失败！");

        }
    });
},
```

3.1.3. 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findPage方法

```
<el-button @click="findPage()" 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

```
<div class="pagination-container">
    <el-pagination
        class="pagiantion"
        @current-change="handleCurrentChange"
        :current-page="pagination.currentPage"
        :page-size="pagination.pageSize"
        layout="total, prev, pager, next, jumper"
        :total="pagination.total">
    </el-pagination>
</div>
```

- 定义handleCurrentChange方法

```
//切换页码
handleCurrentChange(currentPage) {
    // currentPage为切换后的页码
    this.pagination.currentPage = currentPage;
    this.findPage();
},
```

3.2. 后台代码

3.2.1. Controller

在CheckItemController中增加分页查询方法

```
/**
     * 检查项分页查询
     * @param bean  提交上来的参数，包含当前页，每页个数以及查询的条件
     *               {currentPage:1 , pageSize:10 , queryString:"视力"}
     *               {currentPage:1 , pageSize:10 , queryString:null}
     * @return
     */
    @RequestMapping("/findPage")
    public Result findPage(@RequestBody QueryPageBean bean){
        Result result = null;
        try {
            //1. 调用service
            PageResult<CheckItem> pr = cs.findPage(bean);

            //2. 组装result返回
            result  = new Result(true , MessageConstant.QUERY_CHECKITEM_SUCCESS
, pr);

        } catch (Exception e) {
            e.printStackTrace();
            //2. 组装result返回
            result = new Result(false , MessageConstant.QUERY_CHECKITEM_FAIL );
        }

        //3. 返回result
        return result;
    }
```

3.2.2. Service服务实现类

在CheckItemService服务接口中扩展分页查询方法

```
    /**
     * 查询分页
     * @param bean
     * @return 由于页面使用elementUI来编写的，所以返回的数据必须包含两个数据： total &
list
     *          正好PageResult里面包含了这两个数据，所以返回值就写PageResult。
     */
    PageResult<CheckItem> findPage(QueryPageBean bean);
```

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

```
/**
     * 分页查询
     *  1. 一般来说，service向上(controller)返回的数据，通常都是问dao层要的，dao层 去查询
数据库然后得到这份数据
     *  2. 但是现在比较特殊，要返回的是PageResult， dao没有办法去查询具体某一张表，然后得到
PageResult这种对象。
     *  3. 所以在service层里面，我们需要手动分装PageResult。，也就是我们要手动创建
PageResult对象，缺什么东西就
     *      问Dao层要即可。
```

```
     * @param bean
     * @return
     */
    @Override
    public PageResult<CheckItem> findPage(QueryPageBean bean) {

        //0. 设置查询第几页，每页查询多少条
        PageHelper.startPage(bean.getCurrentPage(), bean.getPageSize());

        //1.  调用dao层返回数据
        Page<CheckItem> page = dao.findPage(bean);

        //2. 准备数据
        long total = page.getTotal() ; //总记录数
        List<CheckItem> rows = page.getResult(); //当前页的集合数据

        //3. 创建PageResult
        PageResult<CheckItem> pr = new PageResult<CheckItem>(total , rows);

        return pr;
    }
```

3.2.3. Dao接口

在CheckItemDao接口中扩展分页查询方法

```
    /**
     * 分页查询
     * @param bean
     * @return
     */
    Page<CheckItem> findPage(QueryPageBean bean);
```

3.2.4. Mapper映射文件

在CheckItemDao.xml文件中增加SQL定义

```
  <!--分页查询-->
    <select id="findPage" parameterType="com.itheima.entity.QueryPageBean"
resultType="checkItem">
        select * from t_checkitem
         <!--判断是否含有查询条件:  判断查询条件不是null, 以及长度大于0 即可-->
         <where>
             <if test="queryString != null and queryString.length > 0">
                code like '%${queryString}%' or name like '%${queryString}%'
                 <!--code like "%"#{queryString}"%" or name like  "%"#
{queryString}"%"-->
              </if>
          </where>
     </select>
```

4. 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

- 使用钩子函数，初始化数据

（3）完善分页方法执行时机

- 查询按钮

2：后台代码

（1）CheckItemController.java

（2）CheckItemService.java（服务接口）

（3）CheckItemServiceImpl.java（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

```
//第二种，Mapper接口方式的调用，推荐这种使用方式。底层使用ThreadLocal,创建page对象后存入线程中，将来在dao执行过程中就可以自由的存取
PageHelper.startPage(queryPageBean.getCurrentPage(),
queryPageBean.getPageSize());
```

（4）CheckItemDao.java（Dao接口）

（5）CheckItemDao.xml（Mapper映射文件）

```
  <!--分页查询-->
    <select id="findPage" parameterType="com.itheima.entity.QueryPageBean"
resultType="checkItem">
        select * from t_checkitem
         <!--判断是否含有查询条件:  判断查询条件不是null, 以及长度大于0 即可-->
         <where>
             <if test="queryString != null and queryString.length > 0">
                code like '%${queryString}%' or name like '%${queryString}%'
                 <!--code like "%"#{queryString}"%" or name like  "%"#
{queryString}"%"-->
              </if>
          </where>
     </select>
```

案例-删除检查项

1. 目标

完成检查项的删除

2. 路径

- 前台代码
  - checkitem.html 中给删除按钮绑定事件，当触发时
  - 获取要删除的检查项id
  - 弹出询问窗口
  - 确认后，提交删除请求，要把检查项的id传过去
  - 结果提示，如果成功则要刷新列表
- 后台代码
  - CheckItemController

    提供删除的方法，用Integer接收id，调用业务删除，返回结果给页面
  - CheckItemService与实现类
    - 提供删除的方法
    - 判断检查项是否被检查组使用了，通过检查项id查询检查组检查项关系，其实就是统计检查项的引用个数
    - 如果个数>0，被使用了，报错(MyException, 终止不符合业务代码的执行) 不能删除
    - 如果个数=0，删除检查项
  - CheckItemDao接口与映射文件
    - 通过检查项id查询检查组检查项关系，统计个数, t_checkgroup_checkitem
    - 通过检查项id删除检查项

3. 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

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

调用的方法

```
// 删除
handleDelete(row) {
    alert(row.id);
}
```

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

```
// 删除
handleDelete(row) {
    // alert(row.id);
    this.$confirm("确认删除当前选中记录吗？","提示",{type:'warning'}).then(()=>{
        //点击确定按钮时只需此处代码
        alert('用户点击的是确定按钮');
    });
}
```

3.1.3. 发送请求

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

```
// 删除 : row 表示当前这一条记录的数据。
handleDelete(row) {
    console.log(row);
    //1. 弹窗询问是否真的已经决定删除。
    this.$confirm('确定删除该检查项吗?', '提示', {
        confirmButtonText: '确定',
        cancelButtonText: '取消',
        type: 'warning'
    }).then(() => { //点击确定之后，进入这个位置
        //1. 发请求
        axios.get("/checkitem/delete.do?id="+row.id).then(response=>{
            if(response.data.flag){
                //2. 删除成功，就弹出提示
                this.$message.success(response.data.message);

                //3. 刷新页面
                this.findPage();
            }else{
                this.$message.error(response.data.message);
            }
        });
    });
```

```
}
```

3.2. 后台代码

3.2.1. Controller

在CheckItemController中增加删除方法

```
/**
     * 删除检查项
     * @param id
     * @return
     */
    @RequestMapping("/delete")
    public Result delete(int id){
        //1. 调用service删除
        int row = cs.delete(id);

        //2.判定结果
        Result result = null;
        if(row > 0 ){
            result = new Result(true ,
MessageConstant.DELETE_CHECKITEM_SUCCESS);
        }else{
            result = new Result(false , MessageConstant.DELETE_CHECKITEM_FAIL);
        }

        //3. 返回
        return result;
    }
```

3.2.2. Service服务实现类

在CheckItemService服务接口中扩展删除方法

```

    /**
     * 删除检查项
     * @param id 检查项的id
     * @return
     */
    int delete(int id);
```

在CheckItemServiceImpl服务实现类中扩展删除方法实现

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

```
/**
     * 删除检查项
     *      1. 不能像以前一样，直接上来就删除数据，需要做判断。
     *      2. 如果检查项现在被某一个检查组所使用，那么禁止删除这个检查项
     *          2.1 要想知道检查项是否被检查组使用，其实就是拿着检查项的id 去
t_checkgroup_checkitem 查询总记录数
     *          2.2 如果总记录数 > 0  即表示该检查项被检查组使用了，那么禁止删除。
     *          2.3 如果总记录数 = 0  即表示该检查项没有被检查组使用，那么可以删除。
```

```
     * @param id 检查项的id
     * @return
     */
    @Override
    public int delete(int id) {

        //1. 先查询这个检查项是否有被检查组使用
        int count = dao.findCountById(id);

        //如果 > 0 即表示有记录，那么禁止删除
        if(count >0 ){
            System.out.println("存在检查组使用的情况，禁止删除该检查项：" + id);
            return 0 ;
        }

        //2. 如果没有，就执行删除的操作。
        return dao.delete(id);
    }
```

3.2.3. Dao接口

在CheckItemDao接口中扩展方法findCountByCheckItemId和deleteById

```
    /**
     * 根据检查项的id，去t_checkgroup_checkitem表查询它是否有记录。
     * @param checkItemId
     * @return 存在的记录数
     */
    int findCountById(int checkItemId);

    /**
     * 删除检查项
     * @param checkItemId id值
     * @return
     */
    int  delete (int checkItemId);
```

3.2.4. Mapper映射文件

在CheckItemDao.xml中扩展SQL语句

```
    <!--根据检查项的id 去查询 t_checkgroup_checkitem ，看看是否有记录存在-->
    <select id="findCountById" parameterType="int" resultType="int">
        select count(0) from t_checkgroup_checkitem where checkitem_id = #
{checkItemId};
    </select>

    <!--根据id删除检查项-->
    <delete id="delete" parameterType="int" >
        delete from t_checkitem where id = #{checkItemId}
    </delete>
```

4. 小结

- 前台代码
    1. 绑定删除单击事件
    2. 弹出确认操作提示
    3. 发送ajax请求，执行删除
- 后台代码
    1. CheckItemController.java
    2. CheckItemService.java（服务接口）
    3. CheckItemServiceImpl.java（服务实现类）
    4. CheckItemDao.java（Dao接口）
    5. CheckItemDao.xml（Mapper映射文件）

案例-编辑检查项

1. 目标

1. 完成检查项的数据回显
2. 完成检查项的更新

2. 路径

1. 回显数据
2. 直接把row赋值给formData即可
3. 提交更新
    1. 填入数据，点击【确定】按钮（编辑窗口），绑定事件，校验表单，成功后，提交数据，发送axios.post提交formData。响应结果：不管成功与失败，都提示操作的结果。如果成功：关闭【编辑窗口】，刷新列表数据。
    2. 在CheckItemController, 设置映射路径@RequestMapping，添加update方法，用CheckItem接收formData，调用service.update(checkItem)，构建结果再返回给页面。
    3. 在CheckItemService接口与实现类，提供update(CheckItem), 调用dao更新
    4. CheckItemDao接口与映射文件， 提供update方法，映射文件添加update t_checkitem....

3. 讲解

3.1. 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。

3.1.1. 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

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

处理事件：handleUpdate();

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

3.1.2. 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，直接把row的数据赋值给formData即可

```
  // 弹出编辑窗口
handleUpdate(row) {

    //1. 要想让编辑的窗口显示，其实就是设置dialogFormVisible4Edit = true;
    this.dialogFormVisible4Edit = true;

    //2. 让这个弹出的对话框回显点击的条目数据。
    // row就代表当前这条记录的数据，
    console.log(row);
    this.formData = row;

},
```

3.1.3. 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

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

handleEdit()方法

```
 //编辑 提交请求，去更新检查项
handleEdit() {
    console.log("更新检查项的参数：");
    console.log(this.formData);

    //1. 发请求
    axios.post("/checkitem/update.do" , this.formData).then(response=>{
        if(response.data.flag){
            //1. 更新成功
            this.$message.success(response.data.message);

            //2. 刷新列表页面
            this.findPage();
        }else{
            this.$message.error(response.data.message);
        }
        //3. 隐藏对话框
        this.dialogFormVisible4Edit = false;
    })
```

```
},
```

3.2. 后台代码

3.2.1. Controller

在CheckItemController中增加编辑方法

```
/**
     * 更新检查项
     * @param checkItem
     * @return
     */
    @RequestMapping("/update")
    public Result update(@RequestBody CheckItem checkItem){

        //1. 调用service
        int row = cs.update(checkItem);

        //2. 响应结果
        Result result = null;
        if(row > 0){
            result = new Result(true  , MessageConstant.EDIT_CHECKITEM_SUCCESS);
        }else{
            result = new Result(false  , MessageConstant.EDIT_CHECKITEM_FAIL);
        }
        return result;
    }
```

3.2.2. Service服务实现类

在CheckItemService服务接口中扩展编辑方法

```
    /**
     * 更新检查项
     * @param checkItem
     * @return 影响的行数
     */
    int update(CheckItem checkItem);
```

在CheckItemServiceImpl实现类中实现编辑方法

```
/**
 * 更新检查项
 * @param checkitem
 */
@Override
    public int update(CheckItem checkItem) {
        return dao.update(checkItem);
    }
```

3.2.3. Dao接口

在CheckItemDao接口中扩展edit方法

```
    /**
     * 更新检查项
     * @param checkItem
     * @return
     */
    int update(CheckItem checkItem);
```

3.2.4. Mapper映射文件

在CheckItemDao.xml中扩展SQL语句

```
    <!--更新检查项-->
    <update id="update" parameterType="checkItem">
        update t_checkitem set code = #{code} , name = #{name} , sex=#{sex} ,
age=#{age}
            , price=#{price} , type=#{type},attention=#{attention} , remark = #
{remark}
            where id = #{id}
    </update>
```

4. 小结

- 前台代码
    1. 点击编辑按钮，绑定单击事件
    2. 弹出编辑窗口回显数据
    3. 发送ajax请求，更改数据保存
- 后台代码
    1. CheckItemController.java
    2. CheckItemService.java（服务接口）
    3. CheckItemServiceImpl.java（服务实现类）
    4. CheckItemDao.java（Dao接口）
    5. CheckItemDao.xml（Mapper映射文件）

第二章 - 检查组模块

案例-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。

1. 目标

实现检查组新增功能

2. 路径

1. 弹出新增检查组窗口处理
    1. 弹出对话框，其实就是通过设置隐藏显示即可，设置dialogFormVisiable 为true即可
    2. 弹出的新建检查组的对话框里面有两个选项卡： 基本信息， 检查项信息
        1. 基本信息的选项卡是用来填写检查组的基本内容
        2. 检查项信息是一个表格，要展示出来所有的检查项内容，供检查组选中，表示这个检查组包含这么多的检查项内容。
    3. 当弹出新建检查组对话框的之前， 先去查询所有的检查项出来，然后赋值给选项卡显示(tableData 属性)
2. 提交新增检查组
    - 绑定新增的【确定】按钮，提交formData(checkgroup),checkitemIds(选中的检查项id)，对结果提示，成功则关闭新增窗口且刷新列表数据
    - 创建CheckGroupController，提供add的方法，用户checkgroup接收formData 用Integer[]接收checkitemIds，调用业务添加检查组，返回结果给页面
    - 创建CheckGroupService与实现类，提供add方法
        1. 添加检查组信息
        2. 获取新增的检查组id
        3. 遍历选中的检查项id数组，空判断
        4. 添加检查组与检查项的关系
        5. 事务控制
    - 创建CheckGroupDao与映射文件
        5. 添加检查组信息 返回id
        6. 添加检查组与检查项的关系

3. 讲解

3.1. 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。

3.1.1. 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。

（1）新建按钮绑定单击事件，对应的处理函数为handleCreate

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

（2）handleCreate()方法

```
// 重置表单
resetForm() {
    this.formData = {};
},
// 弹出添加窗口
handleCreate() {
    this.resetForm();
    this.dialogFormVisible = true;
},
```

3.1.2. 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

```
tableData:[],//新增和编辑表单中对应的检查项列表数据
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

```
<el-tab-pane label="检查项信息" name="second">
<div class="checkScrol">
        <table class="datatable">
            <thead>
                <tr>
                    <th>选择</th>
                    <th>项目编码</th>
                    <th>项目名称</th>
                    <th>项目说明</th>
                </tr>
            </thead>
            <tbody>
                <tr v-for="c in tableData">
                   <td>
                       <input :id="c.id" v-model="checkitemIds" type="checkbox"
:value="c.id">
                   </td>
                   <td><label :for="c.id">{{c.code}}</label></td>
                   <td><label :for="c.id">{{c.name}}</label></td>
                   <td><label :for="c.id">{{c.remark}}</label></td>
                </tr>
            </tbody>
      </table>
</div>
</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

```
 // 重置表单
resetForm() {
```

```
    this.activeName = 'first'; //让选项卡默认选中第一个基本信息
    this.formData = {};  //让基本信息内容清空掉
    this.checkitemIds = []; //让选中的id也清空掉
    this.tableData = [] ; // 让检查项的数据清空

},
// 弹出添加窗口
handleCreate() {
    //0 . 清空表单
    this.resetForm();

    // 1. 让对话框展示出来
    this.dialogFormVisible = true;

    // 2. 查询所有的检查项内容下来。
    axios.get("/checkitem/findAll.do").then(response=>{
       if(response.data.flag){
           this.$message.success("查询所有的检查项成功! ");
           console.log(response.data);

           //让检查项显示出来
           this.tableData = response.data.data;
       } else{
           this.$message.success("查询所有的检查项失败! ");
       }
    });

},
```

（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：

```
    /**
     * 查询所有的检查项
     * @return
     */
    @RequestMapping("/findAll")
    public Result findAll(){
        Result result = null;
        try {
            //1. 调用service，查询所有的检查项
            List<CheckItem> list = cs.findAll();

            //2. 返回
            result = new Result(true , MessageConstant.QUERY_CHECKITEM_SUCCESS ,
list);
        } catch (Exception e) {
            e.printStackTrace();
            result = new Result(false , MessageConstant.QUERY_CHECKITEM_FAIL);
        }
        return result;
    }
```

【2】：CheckItemService：

```
    /**
     * 查询所有的检查项
     * @return
     */
    List<CheckItem> findAll();
```

【3】：CheckItemServiceImpl：

```
    @Override
    public List<CheckItem> findAll() {
        return dao.findAll();
    }
```

【4】：CheckItemDao：

```
    /**
     * 查询所有的检查项
     * @return
     */
    List<CheckItem> findAll();
```

【5】：CheckItemDao.xml：

```
    <!--查询所有的检查项-->
    <select id="findAll" resultType="checkItem">
        select * from t_checkitem
    </select>
```

3.1.3. 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

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

（2）完善handleAdd方法

```
 //添加
handleAdd () {

    /*
        1. 发起请求，把添加的数据，提交到后台
            1.1 提交的数据有两部分：基本信息 + 检查项信息
            1.2 基本信息就是formData数据，这是一份json数据 {name:"xxx",code:"xxx"
,...}
            1.3 检查项信息就是checkitemIds ，这是一份jsonArray 数据  [28,29,30]
        2. 要同时携带者两个数据有两种做法：
            2.1 定义一个更大的json对象来包装他们两个数据，那么此时后台的代码也得有一个对应的
JavaBean来接受它们
```

```
            2.2 不需要定义更大的json对象，formData包含了很多的表单项，那么把它放在参数位
置，
                checkitemIds 数据量比较少，所以把它放在地址上。
            2.3 还是采用post请求提交。
     */
    console.log("新建检查组的参数：");
    console.log(this.formData);
    console.log(this.checkitemIds);

    //1. 发起请求
    axios.post("/checkgroup/add.do?checkitemIds="+this.checkitemIds ,
this.formData ).then(response=>{
        if(response.data.flag){
            this.$message.success("新建检查组成功");
            console.log(response);


            //2. 成功之后，让对话框消失
            this.dialogFormVisible = false;

            //3. 重新刷新一遍列表页面
            this.findPage();
        }else{
            this.$message.success("新建检查组失败");

        }
    });
},
```

3.2. 后台代码

3.2.1. Controller

在health_web工程中创建CheckGroupController

```
package com.itheima.controller;

import com.itheima.constant.MessageConstant;
import com.itheima.entity.Result;
import com.itheima.health.pojo.CheckGroup;
import com.itheima.service.CheckGroupService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Arrays;

/*
    检查组的控制器
 */
@RestController
@RequestMapping("/checkgroup")
public class CheckGroupController {

    @Autowired
    private CheckGroupService cs ;
```

```
    @RequestMapping("/add")
    public Result add(@RequestBody CheckGroup checkGroup , int [] checkitemIds){
        System.out.println(checkGroup);
        System.out.println(Arrays.toString(checkitemIds));

        //1. 调用Service干活
        int row = cs.add(checkGroup, checkitemIds);

        //2. 判定
        Result result = null;
        if(row >0 ){
            result = new Result(true , MessageConstant.ADD_CHECKGROUP_SUCCESS);
        }else{
            result = new Result(false , MessageConstant.ADD_CHECKGROUP_FAIL);
        }

        return result;
    }
}
```

3.2.2. Service服务实现类

在health_service工程中创建CheckGroupService接口

```
package com.itheima.service;

import com.itheima.health.pojo.CheckGroup;

public interface CheckGroupService {

    /**
     * 添加检查组
     * @param checkGroup  检查组的基本信息
     * @param checkitemIds 检查组包含的检查项的id值
     * @return >0 : 添加成功，否则：添加失败。
     */
    int add(CheckGroup checkGroup , int [] checkitemIds);
}
```

在health_service工程中创建CheckGroupServiceImpl实现类

```
package com.itheima.service.impl;

import com.itheima.dao.CheckGroupDao;
import com.itheima.health.pojo.CheckGroup;
import com.itheima.service.CheckGroupService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class CheckGroupServiceImpl implements CheckGroupService {

    //注入dao
```

```
    @Autowired
    private CheckGroupDao dao ;

    /**
     * 新增检查组
     *      1. 检查组包含两份数据，一份是自己的检查组的基本信息，一份是检查组使用了哪些检查项
     *      2. 这两份数据需要存到两张不同的表： t_checkgroup & t_checkgroup_checkitem
     *          2.1 先把基本的信息存入到检查组的表里面 ：t_checkgroup
     *          2.2 再把这个检查组用到了哪些检查项，把这些记录保存到中间表 ：
t_checkgroup_checkitem
     *      3. 一定要先往t_checkgroup这张表添加数据，这样子我们才能得到主键的返回，才能知道
这个检查组的
     *          id值是多少。有了id值，才能去往中间表里面添加记录。
     *
     * @param checkGroup  检查组的基本信息
     * @param checkitemIds 检查组包含的检查项的id值
     * @return
     */
    @Override
    public int add(CheckGroup checkGroup, int[] checkitemIds) {

        //1. 往t_checkgroup 添加基本信息
        int row = dao.add(checkGroup);

        /*
            2. 往t_checkgroup_checkitem添加检查项信息
                2.1 由于从页面过来的时候，这个检查组可能选择了很多的检查项，
                    所以这里要遍历出来每一个检查项
                2.2 遍历一次，就往中间表里面添加一条记录。
                    checkitemIds = 【28,29,30】;
         */
        //只有主表（检查组的表）添加成功了之后，再去考虑添加从表（中间表）的数据
        int row2 = 0 ;
        if(row > 0 ){
            for (int checkitemId : checkitemIds) {
                row2 += dao.addItem(checkGroup.getId() , checkitemId );
            }
        }

        //当所有的操作都成功的时候，就返回1， 否则就返回 0 。
        return (row > 0 && row2 == checkitemIds.length) ?  1 :  0 ;
    }
}
```

3.2.3. Dao接口

在health_dao工程中创建CheckGroupDao接口

```
package com.itheima.dao;

import com.itheima.health.pojo.CheckGroup;
import org.apache.ibatis.annotations.Param;

public interface CheckGroupDao {

    /**
```

```
     * 添加检查组的基本信息
     * @param checkGroup 检查组的对象
     * @return 影响的行数
     */
    int add(CheckGroup checkGroup);

    /**
     * 添加检查组和检查项的关系到中间表去
     * @param checkGroupId 检查组的id
     * @param checkItemId 检查项的id
     * @return  影响的行数
     */
    int addItem(@Param("checkGroupId") int checkGroupId , @Param("checkItemId")
int checkItemId);
}
```

3.2.4. 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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.itheima.dao.CheckGroupDao">


    <!--
        往t_checkgroup 添加检查组的基本信息
            id: 与方法名一样
            parameterType: 参数类型
            keyProperty:  返回的主键数据，使用CheckGroup这个类里面的什么属性来装
            useGeneratedKeys :   是否使用数据库自己创建出来的id主键。 是否由数据库生成主
键，还是由mybatis自己搞出来主键

     -->
    <insert id="add" parameterType="checkGroup" keyProperty="id"
useGeneratedKeys="true">
        insert into t_checkgroup values (null , #{code} , #{name} , #{helpCode} ,
#{sex} , #{remark} , #{attention})
    </insert>


    <!--
        往中间表里面添加记录
            在mybatis里面，方法的参数存在多个情况。
    -->
    <insert id="addItem" parameterType="int">
        insert  into t_checkgroup_checkitem values ( #{checkGroupId},  #
{checkItemId} );
    </insert>

 </mapper>
```

4. 小结

- 前台代码
    1. 弹出新增窗口
        1. 使用选项卡
        2. 选项卡一：检查组信息
        3. 选项卡二：检查项列表，并提供复选框
    2. 新增窗口中，动态展示检查项列表
- 查询所有检查项

  提交请求，执行保存

2：后台代码

- 保存检查组数据
- 保存检查组和检查项中间表数据

  （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）
- 插入检查组的表
- 插入检查组和检查项的中间表

案例-检查组分页

1. 目标

```
完成检查组查询分页
```

2. 路径

1：前台代码

（1）定义分页相关模型数据 pagination

（2）定义分页方法 findPage

提交pagination,响应结果处理，如果失败则提示，成功则绑定数据(分页结果集绑定dataList, 总记录数pagination.total)

（3）完善分页方法执行时机（点击“查询”，点击“分页”）

2：后台代码

执行

- 检查组分页查询

（1）CheckGroupController.java（Controller）

提供findPage方法，调用业务查询，PageResult，把pageResult封装到result再返回给页面

（2）CheckGroupService.java（服务接口）

（3）CheckGroupServiceImpl.java（服务实现类）

添加findPage的方法，

```
* 判断是否有查询条件，如果有则实现模糊查询，拼接%
* 使用PageHelper.startPage(页码，大小)
* 调用dao的findByCondition条件查询，返回page对象
* 通过page对象获取total,result分页结果集
* 封装到pageResult，返回给controller
```

（4）CheckGroupDao.java（Dao接口）

（5）CheckGroupDao.xml（Mapper映射文件）

提供 findByCondition， 使用条件 判断

3. 讲解

3.1. 前台代码

3.1.1. 定义分页相关模型数据

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

3.1.2. 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

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

findPage()方法。

```
//分页查询
findPage() {
    axios.post('/checkgroup/findPage.do',this.pagination).then(res => {
        if(res.data.flag){
            // 绑定数据
            this.dataList = res.data.data.rows;
            // 总记录数
            this.pagination.total = res.data.data.total;
        }else{
            this.$message.error(res.data.message);
        }
    })
}
```

3.1.3. 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findPage方法

```
<el-button @click="handleCurrentChange(1)" class="dalfBut">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

```
<div class="pagination-container">
    <el-pagination
        class="pagiantion"
        @current-change="handleCurrentChange"
        :current-page="pagination.currentPage"
        :page-size="pagination.pageSize"
        layout="total, prev, pager, next, jumper"
        :total="pagination.total">
    </el-pagination>
</div>
```

（3）定义handleCurrentChange方法

```
//切换页码
handleCurrentChange(currentPage) {
    //currentPage为切换后的页码
    this.pagination.currentPage = currentPage;
    this.findPage();
},
```

3.2. 后台代码

3.2.1. Controller

在CheckGroupController中增加分页查询方法

```
/**
 * 分页条件查询
 */
@PostMapping("/findPage")
public Result findPage(@RequestBody QueryPageBean queryPageBean){
    // 调用业务查询
    PageResult<CheckGroup> pageResult =
checkGroupService.findPage(queryPageBean);
    // 封装到Result返回
    return new Result(true,
MessageConstant.QUERY_CHECKGROUP_SUCCESS,pageResult);
}
```

3.2.2. Service服务实现类

在CheckGroupService服务接口中扩展分页查询方法

```
    /**
     * 分页条件查询
     * @param queryPageBean
     * @return
     */
    PageResult<CheckGroup> findPage(QueryPageBean queryPageBean);
```

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

```
/**
 * 分页条件查询
 * @param queryPageBean
 * @return
 */
@Override
public PageResult<CheckGroup> findPage(QueryPageBean queryPageBean) {
    //- 判断是否有查询条件，如果有则实现模糊查询，拼接%
    if(!StringUtils.isEmpty(queryPageBean.getQueryString())){
        queryPageBean.setQueryString("%"+queryPageBean.getQueryString()+"%");
    }
    //- 使用PageHelper.startPage(页码，大小)
    PageHelper.startPage(queryPageBean.getCurrentPage(),
queryPageBean.getPageSize());
    //- 调用dao的findByCondition条件查询，返回page对象
    Page<CheckGroup> page =
checkGroupDao.findByCondition(queryPageBean.getQueryString());
    //- 通过page对象获取total,result分页结果集
    //- 封装到pageResult，返回给controller
    PageResult<CheckGroup> pageResult = new PageResult<CheckGroup>
(page.getTotal(),page.getResult());
    return pageResult;
}
```

3.2.3. Dao接口

在CheckGroupDao接口中扩展分页查询方法

```
Page<CheckGroup> findByCondition(String queryString);
```

3.2.4. Mapper映射文件

在CheckGroupDao.xml文件中增加SQL定义

```
<select id="findByCondition" parameterType="String" resultType="checkgroup">
    select id,code,name,helpCode,sex,remark,attention from t_checkgroup
    <where>
        <if test="value != null and value.length > 0">
            code like #{queryString} or name like #{queryString} or helpCode like
#{queryString}
        </if>
    </where>
</select>
```

4. 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findPage()

（3）完善分页方法执行时机（点击“查询”，点击“分页”）

2：后台代码

- 检查组分页查询

（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）

案例-编辑检查组

1.目标

完成编辑检查组

2. 路径

- 回显

  直接把row赋值给formData即可

- 更新
  - 绑定编辑窗口的【确定】按钮，提交formData(checkgroup),checkitemIds(选中的检查项id)，对结果提示，成功则关闭编辑窗口且刷新列表数据
  - CheckGroupController，提供update的方法，用户checkgroup接收formData 用Integer[]接收checkitemIds，调用业务更新检查组，返回结果给页面
  - CheckGroupService与实现类，提供update方法
    - 更新检查组信息
    - 获取新增的检查组id
    - 先通过检查组id删除检查组与检查项的关系
    - 遍历选中的检查项id数组，空判断
    - 添加检查组与检查项的新关系
    - 事务控制
  - 创建CheckGroupDao与映射文件
    - 更新检查组信息
    - 通过检查组id删除检查组与检查项的关系

3. 讲解

3.1. 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）。

3.1.1. 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

```
<el-table-column label="操作" align="center">
    <template slot-scope="scope">
        <el-button type="primary" size="mini" @click="handleUpdate(scope.row)">编
辑</el-button>
        <el-button size="mini" type="danger" @click="handleDelete(scope.row)">删
除</el-button>
    </template>
</el-table-column>
```

（2）handleUpdate事件

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

3.1.2. 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

```
// 弹出编辑窗口
handleUpdate(row) {
    var id = row.id;// 检查组的id
    // 1. 重置表单
    this.resetForm();
    // 2. 弹出编辑的窗口
    this.dialogFormVisible4Edit = true;
    // 3. 通过id查询检查组信息
    axios.get('/checkgroup/findById.do?id=' + id).then(res => {
        if(res.data.flag){
            // 绑定form表单，检查组信息
            this.formData = res.data.data;
            // 成功后查询检查项数据【注意res变量名不要重复了】
            // 4. 查询检查项列表数据
            axios.get('/checkitem/findAll.do').then(res1 => {
                if(res1.data.flag){
                    this.tableData = res1.data.data;
                    // 成功后查询选中的检查项id
                    // 5. 通过检查组id查询选中的检查项id集合，钩选
                    axios.get('/checkgroup/findCheckItemIdsByCheckGroupId.do?
id=' + id).then(res2 => {
                        if(res2.data.flag){
```

```
                            // 成功则绑定
                            this.checkitemIds = res2.data.data;
                        }else{
                            this.$message.error(res2.data.message);
                        }
                    })
                }else{
                    this.$message.error(res1.data.message);
                }
            })
        }else{
            this.$message.error(res.data.message);
        }
    })
}
```

3.1.3. 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

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

（2）handleEdit()方法

```
//编辑提交
handleEdit() {
    //提交formData(checkgroup),checkitemIds(选中的检查项id),
    axios.post('/checkgroup/update.do?checkitemIds=' +
this.checkitemIds,this.formData).then(res => {
        // 对结果提示，
        this.$message({
            message: res.data.message,
            type: res.data.flag?"success":"error"
        })
        // 成功则关闭编辑窗口且刷新列表数据
        if(res.data.flag){
            // 关闭编辑窗口
            this.dialogFormVisible4Edit = false;
            // 刷新列表数据
            this.findPage();
        }
    });
}
```

3.2. 后台代码

3.2.1. Controller

在CheckGroupController中增加方法

```
/**
 * 通过id检查组信息
 */
@GetMapping("/findById")
public Result findById(int id){
```

```
    // 调用业务查询
    CheckGroup checkGroup = checkGroupService.findById(id);
    return new Result(true,
MessageConstant.QUERY_CHECKGROUP_SUCCESS,checkGroup);
}

/**
 * 通过检查组id查询选中的检查项id集合
 * @param id
 * @return
 */
@GetMapping("/findCheckItemIdsByCheckGroupId")
public Result findCheckItemIdsByCheckGroupId(int id){
    List<Integer> list = checkGroupService.findCheckItemIdsByCheckGroupId(id);
    return new Result(true, MessageConstant.QUERY_CHECKGROUP_SUCCESS,list);
}

/**
 * 更新检查组
 * @param checkGroup
 * @param checkitemIds 注意与前端提交的参数名要一致
 * @return
 */
@PostMapping("/update")
public Result update(@RequestBody CheckGroup checkGroup, Integer[] checkitemIds)
{
    // 调用业务更新
    checkGroupService.update(checkGroup, checkitemIds);
    // 返回结果
    return new Result(true, MessageConstant.EDIT_CHECKGROUP_SUCCESS);
}
```

3.2.2. Service服务实现类

在CheckGroupService服务接口中扩展方法

```
/**
 * 通过id检查组信息
 * @param id
 * @return
 */
CheckGroup findById(int id);

/**
 * 通过检查组id查询选中的检查项id集合
 * @param id
 * @return
 */
List<Integer> findCheckItemIdsByCheckGroupId(int id);

/**
 * 更新检查组
 * @param checkGroup
 * @param checkitemIds
 */
void update(CheckGroup checkGroup, Integer[] checkitemIds);
```

在CheckGroupServiceImpl实现类中实现编辑方法

```
/**
 * 通过id检查组信息
 * @param id
 * @return
 */
@Override
public CheckGroup findById(int id) {
    return checkGroupDao.findById(id);
}

/**
 * 通过检查组id查询选中的检查项id集合
 * @param id
 * @return
 */
@Override
public List<Integer> findCheckItemIdsByCheckGroupId(int id) {
    return checkGroupDao.findCheckItemIdsByCheckGroupId(id);
}

/**
 * 更新检查组
 * @param checkGroup
 * @param checkitemIds
 */
@Override
@Transactional
public void update(CheckGroup checkGroup, Integer[] checkitemIds) {
    //1. 更新检查组信息
    checkGroupDao.update(checkGroup);
    //2. 获取新增的检查组id
    Integer checkGroupId = checkGroup.getId();
    //3. 先通过检查组id删除检查组与检查项的旧关系
    checkGroupDao.deleteCheckGroupCheckItem(checkGroupId);
    //4. 遍历选中的检查项id数组，空判断
    if(null != checkitemIds) {
        //5. 添加检查组与检查项的新关系
        for (Integer checkitemId : checkitemIds) {
            checkGroupDao.addCheckGroupCheckItem(checkGroupId,checkitemId);
        }
    }
    //6. 事务控制
}
```

3.2.3. Dao接口

在CheckGroupDao接口中扩展方法

```
/**
 * 通过id检查组信息
 * @param id
 * @return
 */
CheckGroup findById(int id);
```

```
/**
 * 通过检查组id查询选中的检查项id集合
 * @param id
 * @return
 */
List<Integer> findCheckItemIdsByCheckGroupId(int id);

/**
 * 更新检查组信息
 * @param checkGroup
 */
void update(CheckGroup checkGroup);
/**
 * 通过检查组id删除检查组与检查项的旧关系
 * @param checkGroupId
 */
void deleteCheckGroupCheckItem(Integer checkGroupId);
/**
 * 通过检查组id删除检查组与检查项的旧关系
 * @param checkGroupId
 */
void addCheckGroupCheckItem(@Param("checkGroupId") Integer checkGroupId,
@Param("checkitemId")Integer checkitemId);
```

3.2.4. Mapper映射文件

在CheckGroupDao.xml中扩展SQL语句

```
<select id="findById" parameterType="int" resultType="checkgroup">
    select id,code,name,helpCode,sex,remark,attention from t_checkgroup where
id=#{id}
</select>

<select id="findCheckItemIdsByCheckGroupId" parameterType="int"
resultType="int">
    select checkitem_id from t_checkgroup_checkitem where checkgroup_id=#{id}
</select>

<update id="update" parameterType="Checkgroup">
    update t_checkgroup
    set
        code=#{code},
        name=#{name},
        helpCode=#{helpCode},
        sex=#{sex},
        remark=#{remark},
        attention=#{attention}
    where id=#{id}
</update>

<delete id="deleteCheckGroupCheckItem" parameterType="int">
    delete from t_checkgroup_checkitem where checkgroup_id=#{checkGroupId}
</delete>
    <!--新增CheckGroupCheckItem关系-->
<insert id="addCheckGroupCheckItem" >
    insert into t_checkgroup_checkitem values (#{checkGroupId},#{checkitemId})
</insert>
```

4. 小结

关系维护：先删除旧关系，再添加新关系，使用事务来保证完整性

1：前台页面

（1）绑定“编辑”单击事件

（2）弹出编辑窗口回显数据

- 回显检查组数据
- 查询检查项列表
- 当前检查组具有的检查项的复选框需要选中

（3）发送请求，编辑保存检查组

- 编辑检查组

2：后台编码

- 编辑检查组保存
- 删除检查项和检查组中间表数据
- 重新新增检查项和检查组中间表数据

（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）

课后作业-删除检查组分析

页面：

1. 给【删除】按钮绑定事件
2. 获取要删除的检查组id
3. 弹出询问窗口
4. 确定后，提交删除，把id传递给后台
5. 对结果提示，如果成功则要刷新列表数据

后台

1. Controller提供删除的方法，接收传过来的id, 调用业务删除，返回结果给页面
2. Service 提供删除的方法
    - 判断这个检查组是否被套餐使用，统计检查组的id在t_setmeal_checkgroup 中的个数
    - 个数>0 被使用了，抛出异常报错，不能删除
    - 个数=0 则可以删除
        - 先通过检查组id删除检查组与检查项的关系
        - 删除检查组
3. Dao
    - 统计检查组的id在t_setmeal_checkgroup 中的个数
    - 通过id删除检查组

#