一、如何学习

1.1 项目阶段目标

1. 本项目的重点是熟悉企业真实项目的开发流程，对实际项目开发有一个较为深刻的理解
2. 掌握项目中的一些重要的框架/第三方工具，比如SpringBoot，MyBatis，Bootstrap，Thymeleaf等技术的使用
3. 锻炼自主思考能力，需求分析能力，解决bug的能力

1.2 项目学习方法

1. 学习和理解项目中的相关业务流程，学会自己去阅读文档，进行需求分析，画图，并使用代码实现需求
2. 在学习的时候，不要纠结于细节的代码实现，要从更大的范围去理解一个项目/产品的设计和开发过程

1.3 做作业的方法

1. 上课做笔记/截图，粗略版本，下课后补充完整，上课若有不理解的先记录问题，下课必须解决
2. 必须做到的是，写代码的时候要先自己思考和梳理， 需求是什么? 实现步骤是什么? 如何通过代码来实现功能?写代码前必须思考，并在文档中列出来
3. 如果没有思路， 就打开需求文档/上课笔记/课件/截图/回顾和理解 , 如果实在理解不了才去看视频 , 理解后把资料的窗口关闭 , 自己重新思考如何实现 , 直到自己能把步骤梳理出来，再开始写代码
4. 遇到问题，先自己解决 > 不行找百度查询 > 再不行找同学/老师帮助，千万不要养成依赖别人的习惯，这样以后工作了是很难独立生存的
5. 当天的作业第一遍做完后，写第二遍，第二遍的目标是独立完成，当天写不完那就自习或休息天时再继续写，自习或休息天还要把总结补上
6. 如果有的同学，作业实在是做得太晚，第一遍都做不完，给讲师或班主任发个微信，申请延交作业，并说明什么原因导致，需要在自习把进度赶上来，如果连续3次都延迟，说明问题较严重，要主动找老师沟通

二、技术选型

在项目开发前，需要对技术方案先做方向性的评估。在投资有限、硬件资源有限的条件下，为了满足需求，需要进行技术方案选型、技术点使用范围进行分析。

常见的技术选型度量点：

（1）快速开发

（2）学习成本低

（3）技术成熟度

（4）稳定性

（5）性能

当前项目技术选型:

主框架：Spring Boot 2.3.x+Spring Framework 5.2.x+Apache Shiro 1.7

持久层：Apache MyBatis 3.5.x+Hibernate Validation 6.0.x+Alibaba Druid 1.2.x

视图层：Bootstrap 3.3.7+Thymeleaf 3.0.

其他： Activiti 7+POI 4.1.2

三、代码讲解

3.1 页面组件讲解

项目中是使用Bootstrap Table来做数据展示的,而且对Api做了一层的封装.所以我们要了解数据如何展示，我们需要先来了解Bootstrap Table是如何使用的.

Bootstrap Table参考文档

3.1.1 列表展示

- 添加Bootstrap Table相关的CSS和JS文件

```
<!-- bootstrap-table 表格插件 -->
<link href="/static/js/plugins/bootstrap-table/bootstrap-table.min.css?
v=20210202" rel="stylesheet"/>
<script src="/static/js/plugins/bootstrap-table/bootstrap-table.min.js?
v=20210202"></script>
<script src="/static/js/plugins/bootstrap-table/locale/bootstrap-table-zh-
CN.min.js?v=20210202"></script>
<script src="/static/js/plugins/bootstrap-table/extensions/mobile/bootstrap-
table-mobile.min.js?v=20210202"></script>
```

- 把 thymeleaf 生成表格的代码删除，添加如下 html 代码

```
<table id="table"></table>
```

- 添加 js 代码把 table 标签渲染成组件，默认分页给后台传递的是 offset 和 limit。

```
$('#table').bootstrapTable({
            url: '/department/listData',
            method: 'GET',   //数据请求方式
            sidePagination:'server',//服务端分页
            pagination:true,//开启分页
            pageNumber:1, //当前地基页
            pageSize:5, //每页显示数据条数
            uniqueId:"id",
            columns: [{
                field: 'id',
                title: '编号'
            }, {
                field: 'name',
                title: '名称'
            }, {
                field: 'sn',
                title: '缩写'
            }, {
                title: '操作',
                align: 'center',
```

```
                formatter: function(value, row, index) {
                    var actions = [];
                    actions.push('<a class="btn btn-success btn-xs"
href="javascript:void(0)" onclick="editOp('+row.id+')"><i class="fa fa-edit">
</i> 编辑</a> ');
                    actions.push('<a class="btn btn-danger btn-xs btn-delete"
href="javascript:void(0)" onclick="deleteOp('+row.id+')"><i class="fa fa-
remove"></i> 删除</a> ');
                    return actions.join('');
                }
            }]
        });
```

- 后台需要提供一个方法返回组件所需要的数据,对返回的数据格式有要求，需要如下格式

```
{"total":8,"rows":[{},{},{}]}
```

- 我们需要提供一个类,把数据封装成如下格式

```
@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
public class TableDataInfo<T> {
    /** 总记录数 */
    private long total;
    /** 列表数据 */
    private List<T> rows;
}
```

- 后台控制器方法需要返回 JSON 格式数据

```
@RequestMapping("/list")
public String list() {
    return "permission/list";
}

@RequestMapping("/listData")
@ResponseBody
public TableDataInfo<Permission> listData(QueryObject qo) {
    PageHelper.offsetPage(qo.getOffset(),qo.getLimit());//设置分页信息
    PageInfo<Permission> pageInfo = permissionService.query(qo);
    return new TableDataInfo<>(pageInfo.getTotal(), pageInfo.getList());
}
```

3.1.2 高级查询

- 在页面中增加输入框和查询按钮

```
 <form class="form-inline">
    <input class="form-control" type="text" name="keyword" placeholder="关键字">
    <a href="#" class="btn btn-success" onclick="searchOp()">
        <span class="glyphicon glyphicon-search"></span> 查询
    </a>
</form>
```

- 在 bootstrapTable 配置中添加如下配置信息

```
queryParams:function(params) {
    params.keyword=$("[name=keyword]").val();
    return params;
},
```

- 增加搜索的点击事件处理

```
function searchOp(){
    $('#table').bootstrapTable('refresh');
}
```

3.1.3 新增功能

- 给添加按钮增加点击事件

```
<a href="#" class="btn btn-success btn-input" style="margin: 10px"
onclick="addOp()">
   <span class="glyphicon glyphicon-plus"></span> 添加
</a>
```

```
function addOp() {
    $('#myModal input[name]').val('');
    $("#myModalLabel").html("部门新增");
    $('#myModal').modal('show');
}
```

3.1.4 编辑功能

- 添加编辑点击事件,需要将数据回显到对话框中

```
function editOp(id){
    var row = $('#table').bootstrapTable('getRowByUniqueId', id);
    $('#myModal input[name]').val('');
    $('input[name=id]').val(row.id);
    $('input[name=name]').val(row.name);
    $('input[name=sn]').val(row.sn);
    $("#myModalLabel").html("部门修改");
    $('#myModal').modal('show');
}
```

3.1.5 保存功能

- 点击保存的时候，使用发送AJAX请求

```
function saveOp(){
    $.ajax({
        url: "/department/saveOrUpdate",
        type: "post",
        dataType: "json",
        data: $('#dataForm').serialize(),
        success: function(data) {
            if(data.success){
                $('#table').bootstrapTable('refresh');
                $('#myModal').modal('hide');
            }else{
                Swal.fire({
                    text: data.data,
                    icon: 'warning'
                })
            }
        }
    });
}
```

- 后台需要返回 JSON 格式数据

```
@RequestMapping("/saveOrUpdate")
@ResponseBody
public JsonResult saveOrUpdate(Department department) {
    if (department.getId() == null) { // 新增
        departmentService.save(department);
    } else {
        departmentService.update(department);
    }
    return new JsonResult(true, "操作成功");
}
```

3.1.6 删除功能

- 添加删除按钮点击事件

```
function deleteOp(id){
    Swal.fire({
        title: '您确定要删除吗？',
        text: "此操作不可撤销!",
        icon: 'warning',
        showCancelButton: true,
        confirmButtonColor: '#3085d6',
        cancelButtonColor: '#d33',
        confirmButtonText: '确定',
        cancelButtonText: '取消'
    }).then((result) => {
        if(result.value) {
            $.ajax({
                url: "/department/delete?id="+id,
                type: "get",
                dataType: "json",
                success: function(data) {
                    if(data.success){
                        $('#table').bootstrapTable('refresh');
```

```
                    }else{
                        Swal.fire({
                            text: data.msg,
                            icon: 'warning'
                        })
                    }
                }
            })
        }
    });
}
```

- 后台控制器方法需要返回JSON格式数据

```
@RequestMapping("/delete")
@ResponseBody
public JsonResult delete(Long id) {
    if (id != null) {
        departmentService.delete(id);
    }
    return new JsonResult(true, "删除成功");
}
```

3.2 Excel导入导出

3.2.1 什么是POI

POI简介（Apache POI），Apache POI是Apache软件基金会的开放源码函式库，POI提供API给Java程序对Microsoft Office格式档案读和写的功能。

Apache POI官网

- HSSF - 提供读写Microsoft Excel格式档案的功能。（.xls）
- XSSF - 提供读写Microsoft Excel OOXML格式档案的功能。（.xlsx）
- HWPF - 提供读写Microsoft Word格式档案的功能。
- HSLF - 提供读写Microsoft PowerPoint格式档案的功能。
- HDGF - 提供读写Microsoft Visio格式档案的功能。

3.2.2 环境准备

- 创建普通Maven项目
- 导入相关依赖

```
<dependencies>
    <!--xls(03)-->
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>4.1.2</version>
    </dependency>

    <!--xlsx(07)-->
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
```

```
        <version>4.1.2</version>
    </dependency>

    <!--日期格式化工具-->
    <dependency>
        <groupId>joda-time</groupId>
        <artifactId>joda-time</artifactId>
        <version>2.10.1</version>
    </dependency>

    <!--test-->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

3.2.3 写Excel功能

我们需要导出效果如下:

	A	B	C
1	今日人数	666	
2	统计时间	2021-06-24 10:30:48	
3			

Excel的文件格式分为Excel2003【后缀名为xls】和Excel2007【后缀名为xlsx】两种，这两种的代码基本上一致的.

- Excel2003写功能

```
@Test
public void testWrite03() throws IOException {
    // 创建新的Excel 工作簿
    Workbook workbook = new HSSFWorkbook();
    // 在Excel工作簿中建一工作表，其名为缺省值 Sheet0
    //Sheet sheet = workbook.createSheet();
    // 如要新建一名为"会员登录统计"的工作表，其语句为:
    Sheet sheet = workbook.createSheet("疫苗接种统计");
    // 创建行（row 1）
    Row row1 = sheet.createRow(0);

    // 创建单元格（col 1-1）
    Cell cell11 = row1.createCell(0);
    cell11.setCellValue("今日人数");

    // 创建单元格（col 1-2）
    Cell cell12 = row1.createCell(1);
    cell12.setCellValue(666);

    // 创建行（row 2）
    Row row2 = sheet.createRow(1);

    // 创建单元格（col 2-1）
    Cell cell21 = row2.createCell(0);
    cell21.setCellValue("统计时间");
```

```
    //创建单元格（第三列）
    Cell cell22 = row2.createCell(1);
    String dateTime = new DateTime().toString("yyyy-MM-dd HH:mm:ss");
    cell22.setCellValue(dateTime);

    // 新建一输出文件流（注意：要先创建文件夹）
    FileOutputStream out = new FileOutputStream("d:/test-write03.xls");
    // 把相应的Excel 工作簿存盘
    workbook.write(out);
    // 操作结束，关闭文件
    out.close();

    System.out.println("文件生成成功");
}
```

- Excel2007写功能

```
@Test
public void testWrite07() throws IOException {
     // 创建新的Excel 工作簿
     Workbook workbook = new XSSFWorkbook();
     ...
     // 新建一输出文件流（注意：要先创建文件夹）
     FileOutputStream out = new FileOutputStream("d:/test-write07.xlsx");
     ...
}
```

3.2.4 大文件写入

- 使用HSSF【Excel2003】

缺点：最多只能处理65536行，否则会抛出异常

java.lang.IllegalArgumentException: Invalid row number (65536) outside allowable range (0..65535)

优点：过程中写入缓存，不操作磁盘，最后一次性写入磁盘，速度快

```
@Test
public void testWrite03BigData() throws IOException {
    //记录开始时间
    long begin = System.currentTimeMillis();

    //创建一个SXSSFWorkbook
    Workbook workbook = new HSSFWorkbook();

    //创建一个sheet
    Sheet sheet = workbook.createSheet();

    //xls文件最大支持65536行
    for (int rowNum = 0; rowNum < 65537; rowNum++) {
        //创建一个行
        Row row = sheet.createRow(rowNum);
        for (int cellNum = 0; cellNum < 10; cellNum++) {//创建单元格
            Cell cell = row.createCell(cellNum);
            cell.setCellValue(cellNum);
```

```
        }
    }

    System.out.println("done");
    FileOutputStream out = new FileOutputStream("d:/test-write03-bigdata.xls");
    workbook.write(out);
    // 操作结束，关闭文件
    out.close();

    //记录结束时间
    long end = System.currentTimeMillis();
    System.out.println((double)(end - begin)/1000);
}
```

- 使用XSSF【Excel2007】

缺点：写数据时速度非常慢，非常耗内存，也会发生内存溢出，如100万条

优点：可以写较大的数据量，如20万条

```
@Test
public void testWrite07BigData() throws IOException {
    ...
    //创建一个SXSSFWorkbook
    Workbook workbook = new XSSFWorkbook();
    ...
    FileOutputStream out = new FileOutputStream("d:/test-write07-bigdata.xlsx");
    ...
}
```

- 使用SXSSF

优点：可以写非常大的数据量，如100万条甚至更多条，写数据速度快，占用更少的内存

注意：

过程中会产生临时文件，需要清理临时文件

默认由100条记录被保存在内存中，如果查过这数量，则最前面的数据被写入临时文件

如果想自定义内存中数据的数量，可以使用new SXSSFWorkbook(数量)

```
@Test
public void testWrite07BigDataFast() throws IOException {
    //记录开始时间
    long begin = System.currentTimeMillis();
    //创建一个SXSSFWorkbook
    Workbook workbook = new SXSSFWorkbook();
    ...
    FileOutputStream out = new FileOutputStream("d:/test-write07-bigdata-
fast.xlsx");
    workbook.write(out);
    // 操作结束，关闭文件
    out.close();
    //清除临时文件
    ((SXSSFWorkbook)workbook).dispose();
    //记录结束时间
    long end = System.currentTimeMillis();
```

```
    System.out.println((double)(end - begin)/1000);
}
```

SXSSFWorkbook-来至官方的解释：实现“BigGridDemo”策略的流式XSSFWorkbook版本。这允许写入非常大的文件而不会耗尽内存，因为任何时候只有可配置的行部分被保存在内存中。

请注意，仍然可能会消耗大量内存，这些内存基于您正在使用的功能，例如合并区域，注释.....仍然只存储在内存中，因此如果广泛使用，可能需要大量内存。

3.2.5读Excel功能

- Excel2003读功能

```
@Test
public void testRead03() throws Exception{
    InputStream is = new FileInputStream("d:/test-write03.xls");

    Workbook workbook = new HSSFWorkbook(is);
    Sheet sheet = workbook.getSheetAt(0);

    // 读取第一行第一列
    Row row = sheet.getRow(0);
    Cell cell = row.getCell(0);

    // 输出单元内容
    System.out.println(cell.getStringCellValue());

    // 操作结束，关闭文件
    is.close();
}
```

- Excel2007读功能

```
@Test
public void testRead07() throws Exception{

    InputStream is = new FileInputStream("d:/test-write07.xlsx");

    Workbook workbook = new XSSFWorkbook(is);
    Sheet sheet = workbook.getSheetAt(0);

    // 读取第一行第一列
    Row row = sheet.getRow(0);
    Cell cell = row.getCell(0);

    // 输出单元内容
    System.out.println(cell.getStringCellValue());

    // 操作结束，关闭文件
    is.close();
}
```

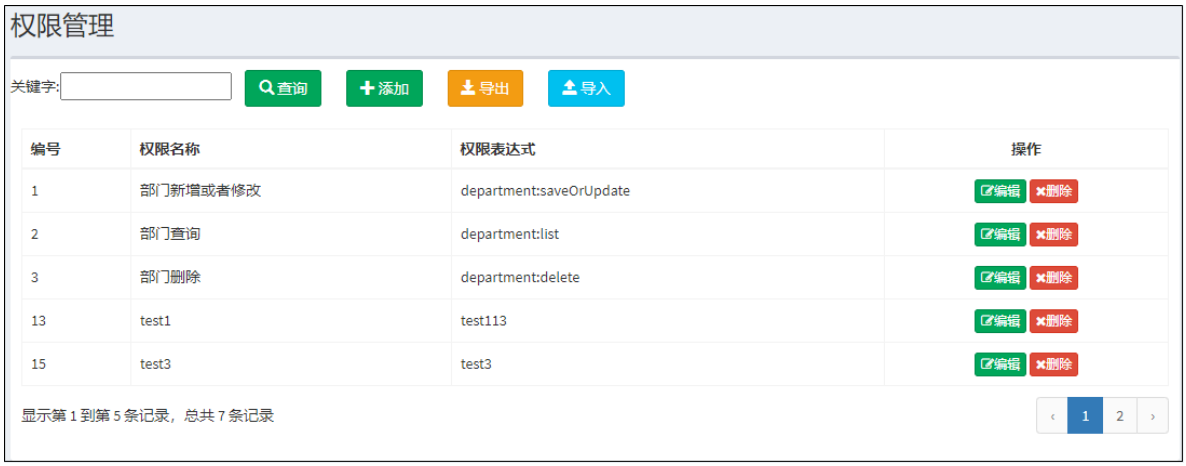
- 读取不同类型的数据

```
@Test
public void testCellType() throws Exception {

    InputStream is = new FileInputStream("d:/商品信息表.xlsx");

    Workbook workbook =  new XSSFWorkbook(is);
    Sheet sheet = workbook.getSheetAt(0);
    // 读取标题所有内容
    Row rowTitle = sheet.getRow(0);
    if (rowTitle != null) {// 行不为空
        // 读取cell
        int cellCount = rowTitle.getPhysicalNumberOfCells();
        for (int cellNum = 0; cellNum < cellCount; cellNum++) {
            Cell cell = rowTitle.getCell(cellNum);
            if (cell != null) {
                String cellValue = cell.getStringCellValue();
                System.out.print(cellValue + "|");
            }
        }
        System.out.println();
    }

    // 读取商品列表数据
    int rowCount = sheet.getPhysicalNumberOfRows();
    for (int rowNum = 1; rowNum < rowCount; rowNum++) {

        Row rowData = sheet.getRow(rowNum);
        if (rowData != null) {// 行不为空
            // 读取cell
            int cellCount = rowTitle.getPhysicalNumberOfCells();
            for (int cellNum = 0; cellNum < cellCount; cellNum++) {
                System.out.print("【" + (rowNum + 1) + "-" + (cellNum + 1) +
"】");
                Cell cell = rowData.getCell(cellNum);
                if (cell != null) {
                    CellType cellType = cell.getCellType();
                    //判断单元格数据类型
                    String cellValue = "";
                    if(CellType.STRING.equals(cellType)){
                        System.out.print("【STRING】");
                        cellValue = cell.getStringCellValue();
                    }else if(CellType.BOOLEAN.equals(cellType)){
                        System.out.print("【BOOLEAN】");
                        cellValue = String.valueOf(cell.getBooleanCellValue());
                    }else if(CellType.NUMERIC.equals(cellType)){
                        System.out.print("【NUMERIC】");
                        if(DateUtil.isCellDateFormatted(cell)){
                            System.out.print("【日期】");
                            Date date = cell.getDateCellValue();
                            cellValue = new DateTime(date).toString("yyyy-MM-
dd");
                        }else{
                            System.out.print("【转换成字符串】");
                            cellValue =
String.valueOf(cell.getNumericCellValue());
                        }
                    }else if(CellType.BLANK.equals(cellType)){
```

```
                    System.out.print("【BLANK】");
                }else{
                    System.out.println(cellType);
                }
                System.out.println(cellValue);
            }
        }
    }

    is.close();
}
```

[Excel大文件内存溢出解决]

3.2.6 Web环境集成Excel



- 添加POI依赖
- 完成导出功能

页面添加按钮和点击事件

```
<a href="#" class="btn btn-warning btn-input" style="margin: 10px"
onclick="exportOP()">
    <span class="glyphicon glyphicon-save"></span> 导出
</a>
```

```
function exportOP(){
    window.open("/permission/export")
}
```

后台代码

```
@RequestMapping("/export")
@ResponseBody
public void export(HttpServletResponse response) throws IOException {
    response.setHeader("Content-
Disposition","attachment;filename=permissionData.xlsx");
    Workbook is = permissionService.export();
    is.write(response.getOutputStream());
}
```

```
@Override
public Workbook export() {
    List<Permission> permissions = listAll();
    Workbook workbook = new XSSFWorkbook();
    Sheet sheet = workbook.createSheet("权限列表");
    Row row = sheet.createRow(0);
    row.createCell(0).setCellValue("编号");
    row.createCell(1).setCellValue("权限名称");
    row.createCell(2).setCellValue("权限表达式");
    Permission permission  = null;
    for(int i=0;i<permissions.size();i++){
        row = sheet.createRow(i+1);
        permission = permissions.get(i);
        row.createCell(0).setCellValue(permission.getId());
        row.createCell(1).setCellValue(permission.getName());
        row.createCell(2).setCellValue(permission.getExpression());
    }
    return workbook;
}
```

- 完成导入功能

添加按钮

```
<a href="#" class="btn btn-info btn-input" style="margin: 10px"
onclick="importOP()">
    <span class="glyphicon glyphicon-open"></span> 导入
</a>
```

添加模块框

```
<#-- 模态框 -->
<div class="modal fade" id="importModal" tabindex="-1" role="dialog" aria-
labelledby="myModalLabel">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
                <h4 class="modal-title" id="myModalLabel">导入</h4>
```

```
            </div>
            <form class="form-horizontal"  enctype="multipart/form-data"
method="post" id="importForm">
                <div class="modal-body">
                    <div class="form-group" style="margin-top: 10px;">
                        <label for="name" class="col-sm-3 control-label">
</label>
                        <div class="col-sm-6">
                            <!-- 文件上传框 -->
                            <input id="uploadFile" type="file" name="file"/>
                        </div>
                    </div>
                    <div class="form-group" style="margin-top: 10px;">
                        <div class="col-sm-3"></div>
                        <div class="col-sm-6">
                            <a href="#" onclick="downloadTemplateOP()"
class="btn btn-success" >
                                <span class="glyphicon glyphicon-download">
</span> 下载模板
                            </a>
                        </div>
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-default" data-
dismiss="modal">取消</button>
                    <button type="button" onclick="importSave()" class="btn btn-
primary btn-submit">保存</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

添加点击事件

```
function importOP(){
    $("#importModal").modal("show");
}
function downloadTemplateOP(){
    window.open("/permission/downloadTemplate")
}
function importSave(){
        var $file1 = $("#uploadFile").val();//用户文件内容(文件)
        // 判断文件是否为空
        if ($file1 == "") {
            Swal.fire({
                text: "请选择上传的目标文件! ",
                icon: 'warning',
            })
            return false;
        }
        var formData = new FormData();//这里需要实例化一个FormData来进行文件上传
        formData.append("file",$("#uploadFile")[0].files[0]);
        $.ajax({
            type : "post",
            url : "/permission/importExcel",
```

```
            data : formData,
            processData : false,
            contentType : false,
            success : function(data){
                if (data.success) {
                    Swal.fire({
                        text: data.msg,
                        icon: 'success',
                    })
                    $("#importModal").modal("hide");
                    $('#table').bootstrapTable('refresh');
                }else{
                    Swal.fire({
                        text: "请选择上传的目标文件！ ",
                        icon: 'warning',
                    })
                }}
        });
}
```

后台代码

```
@RequestMapping("/importExcel")
@ResponseBody
public JsonResult importExcel(MultipartFile file){
    try{
        int count = permissionService.importExcel(file);
        return new JsonResult(true,"成功导入:"+count+"条记录");
    }catch(Exception ex){
        return new JsonResult(false,"导入数据失败");
    }
}
```

```
@Override
public int importExcel(MultipartFile file) throws IOException {
    Workbook workbook = new XSSFWorkbook(file.getInputStream());
    Sheet sheet = workbook.getSheetAt(0);
    int rowNum = sheet.getLastRowNum();
    int insertCount = 0;
    Row row = null;
    Cell c1 = null;
    Cell c2 = null;
    Permission permission;
    //跳过第一行
    for(int i=1;i<rowNum;i++){
        row = sheet.getRow(i);
        c1 = row.getCell(0);
        c2 = row.getCell(1);
        //判断单元格为空就不处理
        if(c1!=null && c2!=null){
            String name = c1.getStringCellValue();
            String expression = c2.getStringCellValue();
            //判断是否有内容
            if(StringUtils.hasText(name) && StringUtils.hasText(expression)){
                //判断表达式是否已经存在
                int count = permissionMapper.getCountByExpression(expression);
```

```
                if(count==0){
                    Permission p = new Permission();
                    p.setName(name);
                    p.setExpression(expression);
                    permissionMapper.insert(p);
                    insertCount++;
                }

            }
        }
    }
    return insertCount;
}
```

3.3 登录功能

```
1.因为项目中集成了Shiro,所以我们可以先看Shiro的配置
2.登录页面是login.html,登录的时候发送的是Ajax请求，具体逻辑在login.js
3.点击登录之后访问的是SysLoginController#ajaxLogin方法
4.具体登录逻辑在UserRealm#doGetAuthenticationInfo方法中.
```

3.4 请求流程

3.5 代码生成器使用

四、业务流程讲解

4.1 项目背景

为了更好的进行商户门店运营，以及维护新老客户的信息，广州e店邦汽车服务公司决定实施汽车门店管理系统。但市面上出售的系统价格昂贵，笨重，所以经过市场调研研究，单独对该公司，做一个轻量级的互联网门店运营系统，主要功能包括购车询价、续保询价、养修预约、试驾预约、故障救援、个人中心、车主宝典、消费结算，套餐维护，流程审核，客户评价，报表分析等功能。

4.2 用户端功能

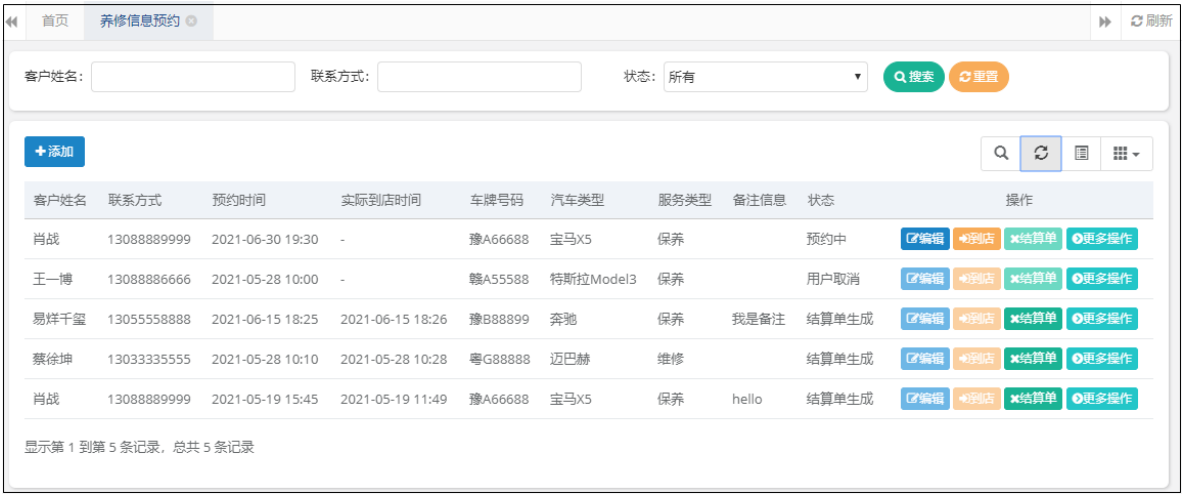
4.2.1 养修预约功能

4.2.2 查看预约记录

4.3 后台管理功能

4.3.1 养修列表展示

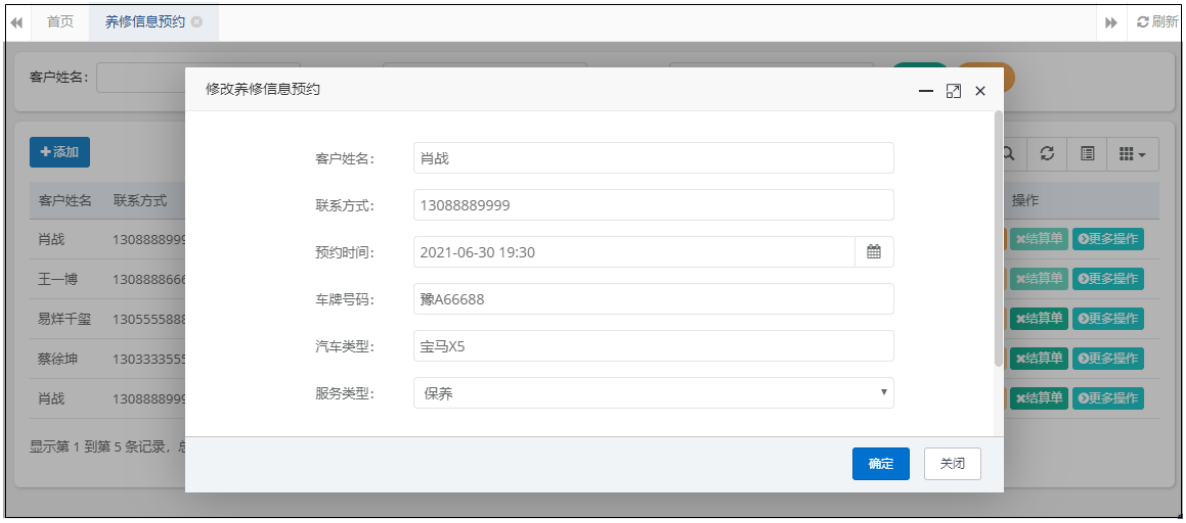
用户在PC端/小程序/公众号填写了预约信息之后，我们后台就可以看到用户的预约信息



4.3.2 电话预约

我们是支持用户直接通过电话联系销售客服/销售顾问进行预约，所以我们是提供添加的功能，客服可以手动添加预约信息.

同时支持对预约的信息进行编辑操作，比如用户想修改预约时间.



4.3.3 用户取消预约

用户如果临时有事不过来了，客服可以在后台取消这个预约.

点击【更多操作】-->【取消】，点击之后该条记录的状态就会变成【用户取消状态】

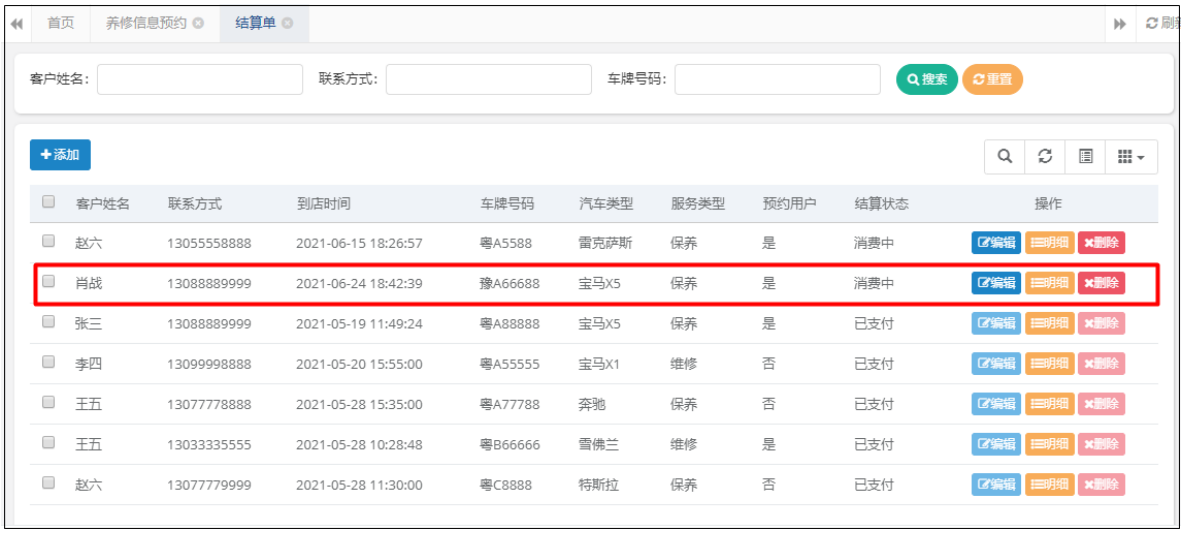
4.3.4 用户到店

用户实际到店的时候，我们需要在系统标记该用户已经到店了.此时记录的状态会变成【已到店】

4.3.5 生成结算单

用户到店之后如果有消费会生成结算单,结算单会记录用户消费哪些服务

在结算单列表中可以看到刚刚生成的结算单



我们可以点击明细进行服务项的添加,根据用户消费的情况添加对应的服务项。

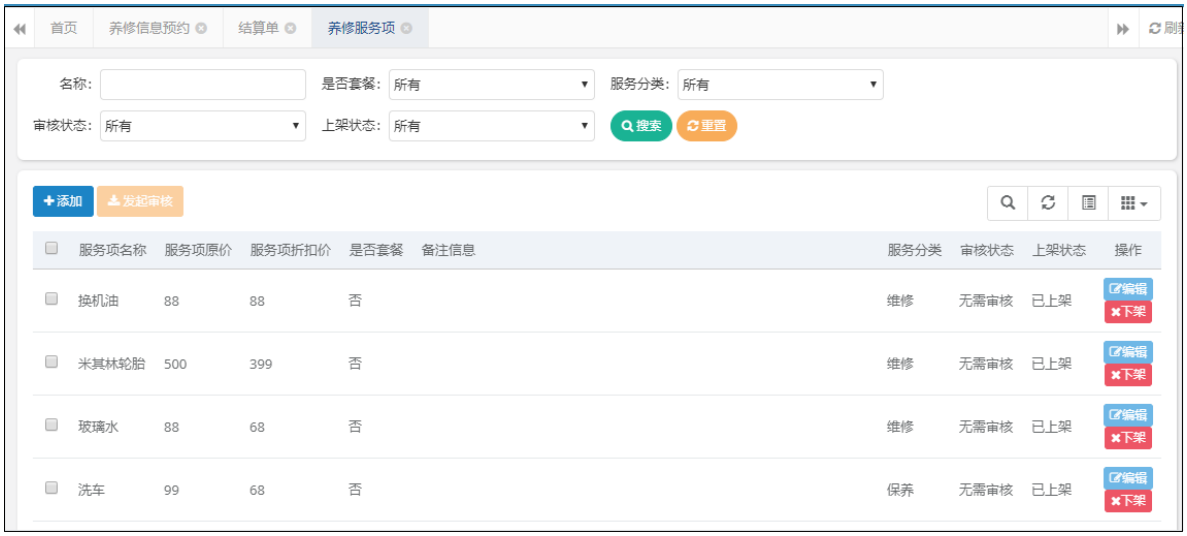
点击保存按钮临时保存目前的消费小.

点击确认支付,结算单状态变更为【已结算】，结算单就不能在添加服务项了.

点击确认支付之后，页面就变成如下:

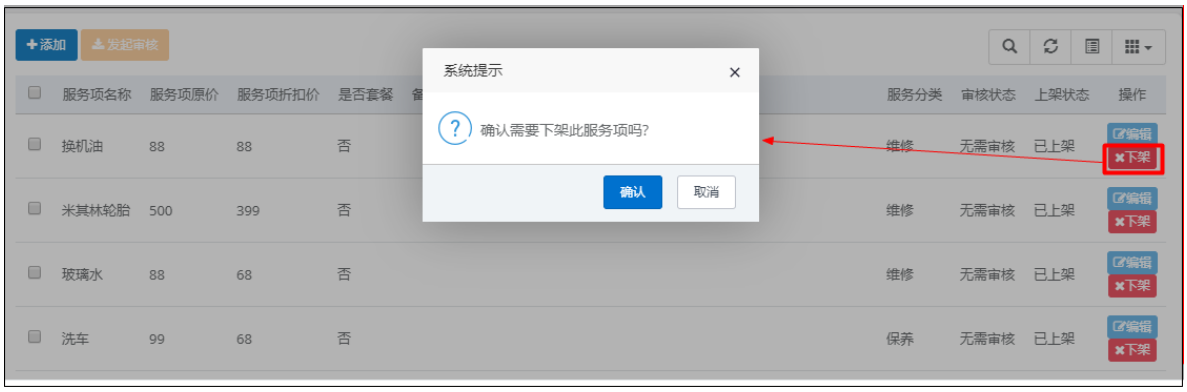
4.3.6 养修服务项列表

在结算单中添加的服务项就是在这里进行维护的.



4.3.7 服务项上架下架功能

服务项需要上架之后，在结算单明细页面中才能看到,如果下架之后,结算单明细中就看不到这个服务项了.

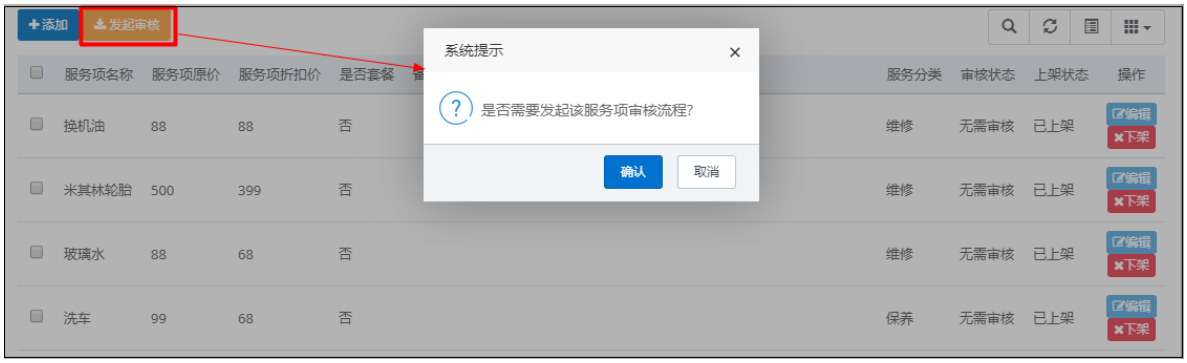


4.3.8 发起审核

服务项分为两种。套餐和非套餐.

套餐是不能直接上架的，需要审核通过才能上架.

审核是使用我们接下来要讲解的Activiti7来实现



五、集成Activiti7

5.1 添加依赖

在`ruoyi-business`模块的pom文件中添加SpringBoot集成Activiti7的依赖

```
<!--添加activiti和SpringBoot整合的依赖
    MyBatis版本会有冲突，所以需要排除-->
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring-boot-starter</artifactId>
    <version>7.0.0.SR1</version>
    <exclusions>
        <exclusion>
            <artifactId>mybatis</artifactId>
            <groupId>org.mybatis</groupId>
        </exclusion>
    </exclusions>
</dependency>
<!--activiti可以绘制流程的的依赖-->
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-image-generator</artifactId>
```

```
    <version>7.0.0.SR1</version>
</dependency>
```

5.2 添加配置信息

在`ruoyi-admin`项目的`application.yml`配置文件中添加Activiti7的配置信息

```
spring:
  activiti:
    database-schema-update: true
    db-history-used: true
    history-level: full
    check-process-definitions: false
    use-strong-uuids: false
```

- database-schema-update属性

  ```
  1.flase：默认值。activiti在启动时，对比数据库表中保存的版本，如果没有表或者版本不匹配，将抛出异常
  2.true： activiti会对数据库中所有表进行更新操作。如果表不存在，则自动创建
  3.create_drop： 在activiti启动时创建表，在关闭时删除表（必须手动关闭引擎，才能删除表）
  4.drop-create： 在activiti启动时删除原来的旧表，然后在创建新表（不需要手动关闭引擎）
  ```

- db-history-used

  ```
  检测历史表是否存在 activiti7默认没有开启数据库历史记录,true启动数据库历史记录
  ```

- history-level

  ```
  #记录历史等级 可配置的历史级别有none, activity, audit, full
  1.none：不保存任何的历史数据，因此，在流程执行过程中，这是最高效的。
  2.activity：级别高于none，保存流程实例与流程行为，其他数据不保存。
  3.audit：除activity级别会保存的数据外，还会保存全部的流程任务及其属性。audit为history的默认值。
  4.full：保存历史数据的最高级别，除了会保存audit级别的数据外，还会保存其他全部流程相关的细节数据，包括一些流程参数等。
  ```

- check-process-definitions

  ```
  #校验流程文件，默认校验resources下的processes文件夹里的流程文件
  ```

- use-strong-uuids

  ```
  是否使用UUID作为主键生成策略
  ```

5.3 排除Spring Security配置

Activiti7默认和Spring Security集成了,但是我们的项目中使用的是Shiro,所以我们需要在项目中排除掉Spring Security的自动装配配置,否则我们的登录页会被覆盖

在`ruoyi-admin`项目的`RuoYiApplication`添加如下信息:

```
package com.ruoyi;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.actuate.autoconfigure.security.servlet.ManagementWebSec
urityAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;
import
org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguratio
n;

/**
 * 启动程序
 */
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class,
        SecurityAutoConfiguration.class,
        ManagementWebSecurityAutoConfiguration.class
})
public class RuoYiApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(RuoYiApplication.class, args);
        System.out.println("启动成功");
    }
}
```

运行项目，如果在数据库中出现25张act_开头的表说明集成是没问题的.

六、流程定义功能

6.1 需求分析

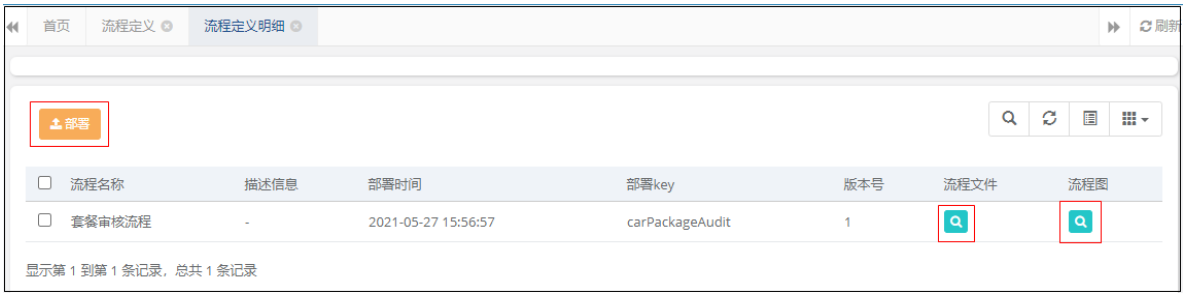
6.1.1 流程定义页面

我们系统中会有很多的工作流程,我们需要有个地方对这些工作流程进行统一的管理(部署/删除/挂起)

需要完成的功能:
1.设计表结构
2.列表页面展示
3.高级查询功能

6.1.2 部署列表页面

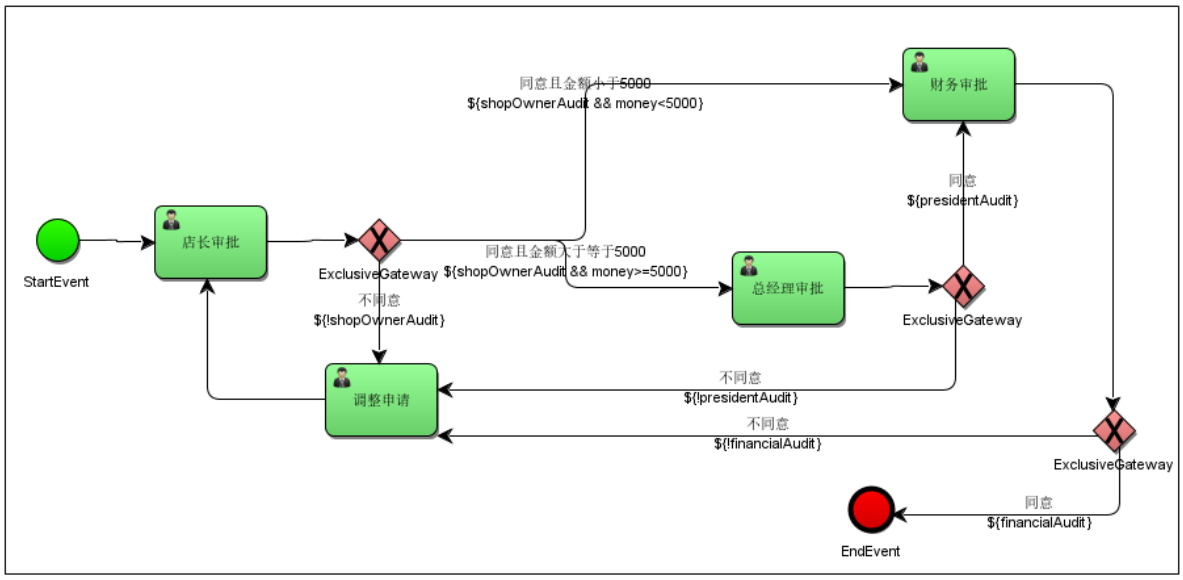
当点击部署列表,可以看到这个流程究竟部署了几个版本,每个版本部署的属性等等.



需要完成的功能:
1.用户某条记录的【部署列表】,打开新的Tab页,然后出对应流程所有的部署信息.(按部署时间逆序)
2.点击部署,可以实现流程的部署
3.查看流程图png
4.查看流程文件xml

6.1.3 节点人员页面

这个是我们需要完成的套餐审核流程



从上图可视,我们有四个节点是需要用户进行审核操作的,这些节点的审核用户我们是不能写死在流程定义中的.

所以我们需要做人员列表功能，定义这个流程中的每一个节点由哪些用户进行审核.

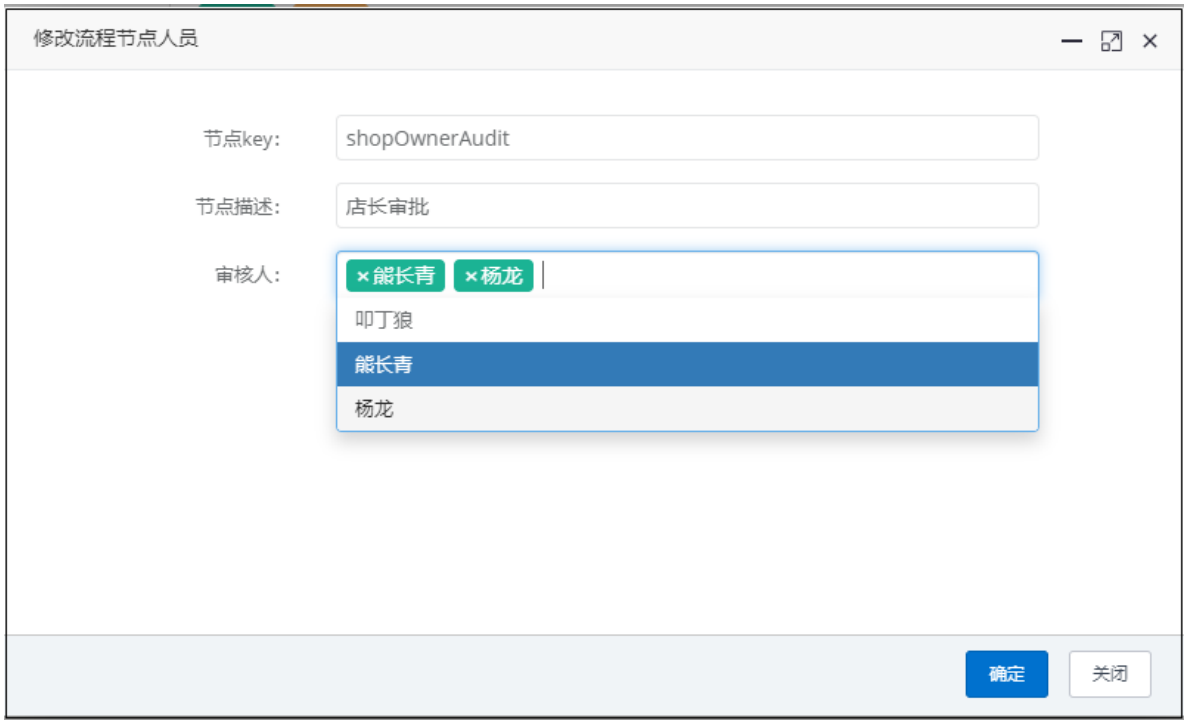
- 节点定义如下:

property	value	property	value	property	value	property	value
Id	shopOwnerAudit	Id	financialAudit	Id	presidentAudit	Id	reApply
Name	店长审批	Name	财务审批	Name	总经理审批	Name	调整申请
Documentation		Documentation		Documentation		Documentation	
Asynchronous	☐	Asynchronous	☐	Asynchronous	☐	Asynchronous	☐
Exclusive	☑	Exclusive	☑	Exclusive	☑	Exclusive	☑
⊞ Multi Instance		⊞ Multi Instance		⊞ Multi Instance		⊞ Multi Instance	
Assignee		Assignee		Assignee		Assignee	

- 节点人员列表

到时候流程流转到某个节点时，我们就可以获取这个节点的id,然后通过这个id从数据库中查询到对应的审核人员列表.

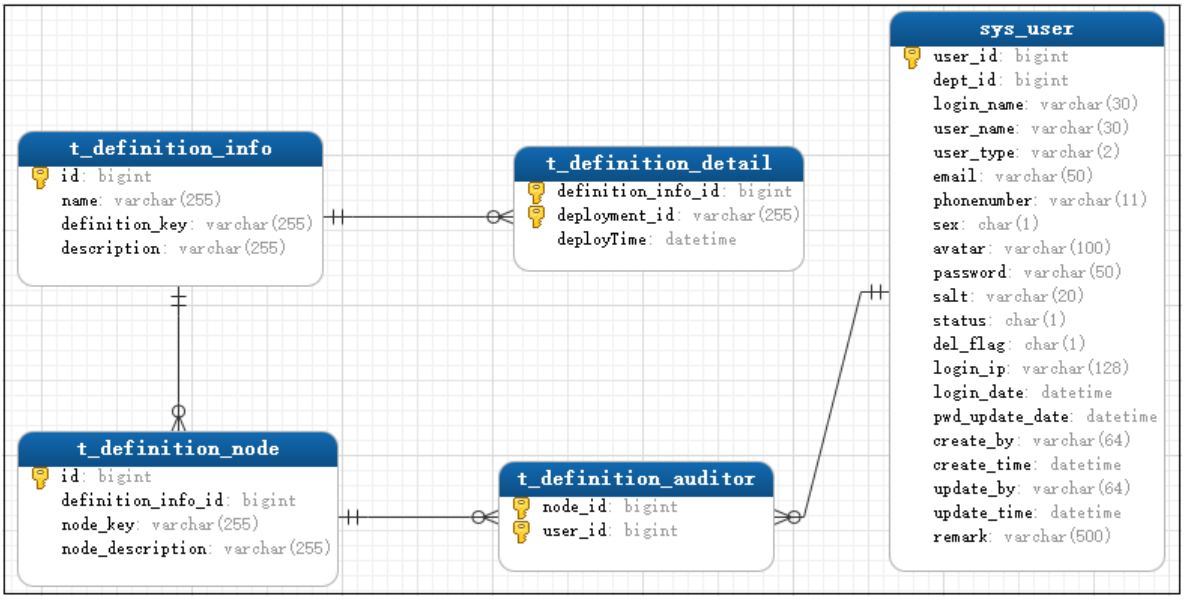
- 编辑功能，每个节点是可以选择多个用户的.



需要完成的功能:
1.根据流程的Id查询改流程的节点信息
2.新增/编辑的时候可以对审核人进行多选操作
3.删除功能

6.2 表结构设计

同学们自行思考,完成上述功能,需要几张表?每张表之间的关联是怎么样的?



- t_definition_info 流程定义表

字段	描述
id	主键
name	流程名称
definition_key	流程key
description	流程描述

```
DROP TABLE IF EXISTS `t_definition_info`;
CREATE TABLE `t_definition_info` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL COMMENT '流程名称',
  `definition_key` varchar(255) DEFAULT NULL COMMENT '流程key',
  `description` varchar(255) DEFAULT NULL COMMENT '流程描述',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 COMMENT='流程定义';

INSERT INTO `t_definition_info` VALUES ('1', '套餐审核流程', 'carPackageAudit',
'设置套餐项需要店长和财务进行审核');
```

- t_definition_detail 流程定义明细

字段	描述
definition_info_id	流程定义ID
deployment_id	部署ID
deploy_time	部署时间

```
CREATE TABLE `t_definition_detail` (
  `definition_info_id` bigint(20) NOT NULL COMMENT '流程定义id',
  `deployment_id` varchar(255) NOT NULL COMMENT '流程部署id',
  `deploy_time` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '部署时间',
  PRIMARY KEY (`definition_info_id`,`deployment_id`),
  CONSTRAINT `t_definition_detail_ibfk_1` FOREIGN KEY (`definition_info_id`)
REFERENCES `ry`.`t_definition_info` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='流程定义明细';
```

- t_definition_node 流程定义节点信息

字段	描述
id	主键ID
definition_info_id	流程ID
node_key	节点的key
node_description	节点的描述

```
CREATE TABLE `t_definition_node` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `definition_info_id` bigint(20) DEFAULT NULL COMMENT '流程定义ID',
  `node_key` varchar(255) DEFAULT NULL COMMENT '节点key',
  `node_description` varchar(255) DEFAULT NULL COMMENT '节点描述',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 COMMENT='流程定义节点信息';
```

- t_definition_auditor 节点审核人

字段	描述
node_id	节点ID
user_id	用户ID

```
CREATE TABLE `t_definition_auditor` (
  `node_id` bigint(20) NOT NULL COMMENT '节点ID',
  `user_id` bigint(20) NOT NULL COMMENT '用户ID',
  PRIMARY KEY (`node_id`,`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='节点审核人';
```

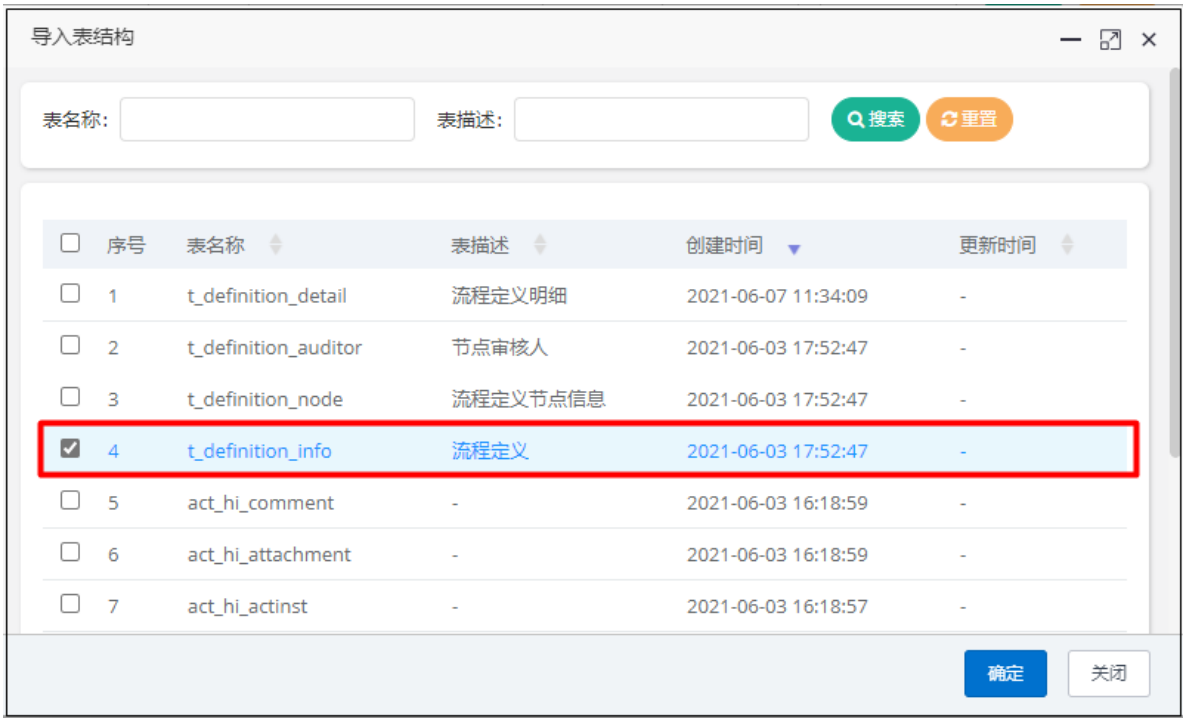
6.3 代码逻辑实现

6.3.1 流程定义功能实现

- 代码生成器生成基础代码

在代码生成页面，点击`导入`按钮,选择我们需要生成表

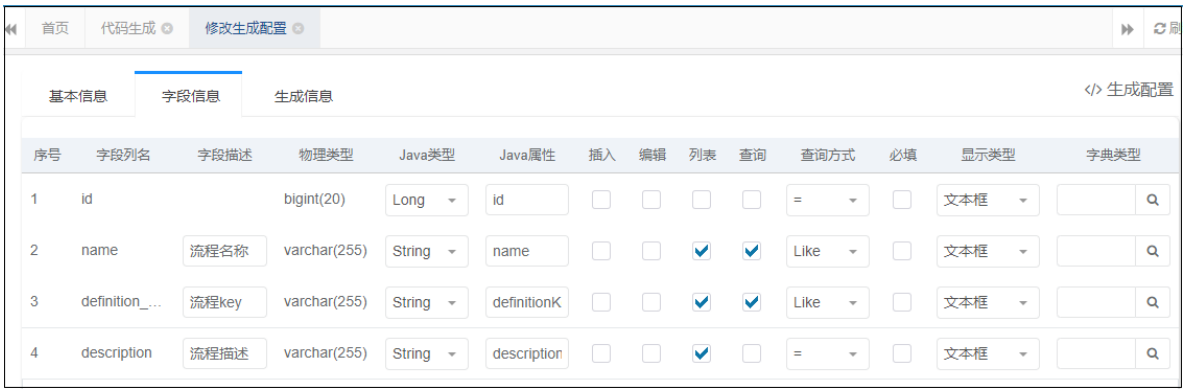
选择需要导入的表.



导入之后效果如下,需要进行进行编辑操作

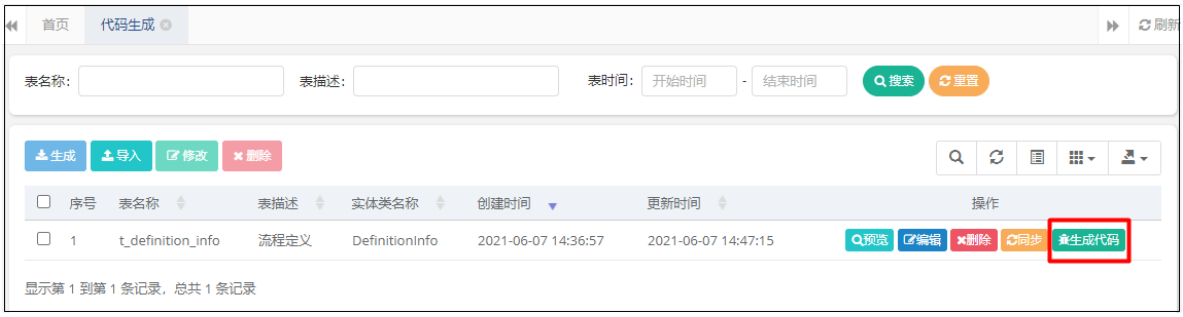
基本信息中根据情况进行修改.

根据情况选择是否需要插入、更新、查询等操作



根据情况在`生成信息`中修改

在列表中点击`生成代码`



将下载好的代码导入到我们的项目中.因为流程定义中我们并不需要新增和编辑，所以我们可以把`add.html`和`edit.html`删除掉,对应的代码也可以删除掉.

在`definitionInfo.html`页面的操作中增加`部署列表`和`人员列表`按钮

```
actions.push('<a class="btn btn-success btn-xs " href="javascript:void(0)"
onclick="openDetail(\'' + row.id + '\')"><i class="fa fa-bars"></i>部署列表</a>
');

actions.push('<a class="btn btn-success btn-xs " href="javascript:void(0)"
onclick="openNode(\'' + row.id + '\')"><i class="fa fa-user"></i>人员列表</a> ');
```

增加对应的处理事件

```
function openDetail(id){
    $.modal.openTab("流程定义明细","/business/definitionDetail/"+definitionId);
}
function openNode(id){
    $.modal.openTab("节点人员列表","/business/definitionNode/"+definitionId);
}
```

6.3.2 部署列表功能实现

6.3.2.1 添加基础代码

使用代码生成器生成部署列表代码

修改`DefinitionDetail.java`实体内容

```
public class DefinitionDetail extends BaseEntity {
    private static final long serialVersionUID = 1L;
    //流程定义ID
    private DefinitionInfo definitionInfo;
    //流程部署ID
    private String deploymentId;
    //部署时间
    private Date deployTime;
```

```
    //描述信息
    private String description;
    //部署的key
    private String deployKey;
    //版本号
    private int version;
}
```

因为部署列表不需要进行新增和编辑操作,所以对应的`edit.html`和`add.html`就不拷贝到项目中了.

在`DefinitionDetailController.java`中修改控制器方法,内容如下:

```
@RequiresPermissions("business:definitionDetail:view")
@GetMapping("/{definitionId}")
public String definitionDetail(@PathVariable("definitionId") String
definitionId,ModelMap mmap) {
    mmap.put("definitionId",definitionId);
    return prefix + "/definitionDetail";
}
```

6.3.2.2 部署功能

我们首先来完成部署的功能，我们需要使用到前端文件上传插件`jasny-bootstrap`.

插件官网地址

```
https://www.jasny.net/bootstrap/components/#fileinput
```

添加对应的CSS和JS文件

```
<th:block th:include="include :: jasny-bootstrap-css" />
<th:block th:include="include :: jasny-bootstrap-js" />
```

需要在页面中添加插件的html代码

```
<div id="uploadBtn" class="fileinput fileinput-new" data-provides="fileinput">
      <span class="btn btn-outline-secondary btn-file">
        <span class="fileinput-new">
            <a class="btn btn-warning btn-sm" ><i class="fa fa-upload">
</i> 部署</a>
        </span>
        <span class="fileinput-exists">Change</span>
        <input type="file" name="processDefinition" multiple>
      </span>
      <span class="fileinput-filename"></span>
      <a href="#" class="close fileinput-exists" data-dismiss="fileinput"
style="float: none">&times;</a>
</div>
```

然后需要对这个按钮进行监听操作,监听到事件之后我们就需要通过Ajax实现文件上传

```
参考资料:https://blog.csdn.net/qq_42944520/article/details/84572509
```

```
var definitionId = [[${definitionId}]];//thymeleaf获取值的语法
```

```
var prefix = ctx + "business/definitionDetail";
$(function() {
    var options = ...;
    $.table.init(options);
    initUploadBtn();
});
function initUploadBtn(){
    $('#uploadBtn').on('change.bs.fileinput', function (e) {
        // 处理自己的业务
        var formdata = new FormData();
        //获取文件上传的dom对象
        formdata.append("processDefinition", $('input[type=file]')[0].files[0]);
        formdata.append("definitionId", definitionId);
        $.ajax({
            url: prefix + '/upload',
            data: formdata,
            type: "post",
            processData: false,
            contentType: false,
            success: function(result) {
                $('#uploadBtn').fileinput('reset');     // 重置
                $.operate.ajaxSuccess(result);
            }
        })
    });
}
```

在控制器`DefinitionDetailController.java`中添加文件上传的代码

```
@RequiresPermissions("business:definitionDetail:upload")
@PostMapping("/upload")
@Log(title = "流程定义明细", businessType = BusinessType.UPDATE)
@ResponseBody
public AjaxResult upload(@RequestParam("processDefinition") MultipartFile file,
Long definitionId)
{
    if(file!=null){
        String originalFilename = file.getOriginalFilename();
        String extName =
originalFilename.substring(originalFilename.lastIndexOf(".")+1);
        if("bpmn".equalsIgnoreCase(extName) || "zip".equalsIgnoreCase(extName)){
            try {
                String fileName = FileUploadUtils.upload(file);
                definitionDetailService.deployProcessDefinition(definitionId,
RuoYiConfig.getProfile()
+fileName.substring(Constants.RESOURCE_PREFIX.length()+1));
            } catch (IOException e) {
                e.printStackTrace();
                return AjaxResult.error("上传流程定义文件失败！");
            }
            return AjaxResult.success();
        }else{
            return AjaxResult.error("流程定义文件仅支持 bpmn 和 zip 格式！");
        }
    }else{
        return AjaxResult.error("不允许上传空文件！");
    }
```

```
}
```

在`DefinitionDetailServiceImpl.java`中处理对应的业务逻辑

```
@Override
@Transactional
public void deployProcessDefinition(Long definitionId, String filePath) {
    //Activiti7流程部署
    Deployment deployment = processService.deploy(filePath);
    //插入流程定义明细
    DefinitionDetail detail = new DefinitionDetail();
    DefinitionInfo definitionInfo = new DefinitionInfo();
    definitionInfo.setId(definitionId);
    detail.setDefinitionInfo(definitionInfo);
    detail.setDeploymentId(deployment.getId());
    definitionDetailMapper.insertDefinitionDetail(detail);
}
```

`processService`中的deploy方法

```
@Override
public Deployment deploy(String filePath) throws IOException {
    Deployment deploy = null;
    if (filePath.endsWith("zip")) {
        ZipInputStream zipInputStream = new ZipInputStream(new
FileInputStream(filePath));
        deploy = repositoryService.createDeployment()
                .addZipInputStream(zipInputStream)
                .deploy();
    } else if (filePath.endsWith("bpmn")) {
        deploy = repositoryService.createDeployment()
                .addInputStream(filePath,new FileInputStream(filePath))
                .deploy();
    }
    return deploy;
}
```

修改`DefinitionDetailMapper.xml`内容

```
<insert id="insertDefinitionDetail" parameterType="DefinitionDetail">
    insert into t_definition_detail
  (definition_info_id,deployment_id)
  values
  (#{definitionInfo.id},#{deploymentId})
</insert>
```

当完成这些功能之后，我们进行测试，但是会提示我们上传失败.后台错误信息

```
Caused by: com.ruoyi.common.exception.file.InvalidExtensionException: filename :
[carPackage-audit.bpmn], extension : [bpmn], allowed extension : [[bmp, gif,
jpg, jpeg, png, doc, docx, xls, xlsx, ppt, pptx, html, htm, txt, rar, zip, gz,
bz2, mp4, avi, rmvb, pdf]]
    at
com.ruoyi.common.utils.file.FileUploadUtils.assertAllowed(FileUploadUtils.java:1
95)
    at
com.ruoyi.common.utils.file.FileUploadUtils.upload(FileUploadUtils.java:109)
    at
com.ruoyi.common.utils.file.FileUploadUtils.upload(FileUploadUtils.java:59)
    ... 109 more
```

因为默认框架中并没有支持`bpmn格式`，所以我们需要添加上. 找到`MimeTypeUtils.java`

```
public static final String[] DEFAULT_ALLOWED_EXTENSION = {
        // 图片
        "bmp", "gif", "jpg", "jpeg", "png",
        // word excel powerpoint
        "doc", "docx", "xls", "xlsx", "ppt", "pptx", "html", "htm", "txt",
        // 压缩文件
        "rar", "zip", "gz", "bz2",
        // 视频格式
        "mp4", "avi", "rmvb",
        // pdf
        "pdf",
        //流程定义文件
        "bpmn"
};
```

6.3.2.3 列表功能

完成文件上传之后,我们就需要来把列表展示一下.

需要把表`t_definition_detail`和表`t_definition_info`进行关联,然后按照部署时间进行降序排列.

请同学们完成这个SQL

在`DefinitionDetailServiceImpl.java`中需要把流程描述信息、版本信息、流程key关联查询出来

```
@Override
public List<DefinitionDetail> selectDefinitionDetailList(DefinitionDetail
definitionDetail) {
    List<DefinitionDetail> definitionDetailList =
definitionDetailMapper.selectDefinitionDetailList(definitionDetail);
    //查询出对应的描述，版本，和业务Key
    for(DefinitionDetail detail:definitionDetailList){
        ProcessDefinition processDefinition =
processService.selectProcessDefinitionByDeploymentId(detail.getDeploymentId());
        detail.setDescription(processDefinition.getDescription());
        detail.setVersion(processDefinition.getVersion());
        detail.setDeployKey(processDefinition.getKey());
    }
    return definitionDetailList;
}
```

在`ProcessServiceImpl.java`中添加方法

```
@Override
public ProcessDefinition selectProcessDefinitionByDeploymentId(String
deploymentId) {
    return repositoryService
            .createProcessDefinitionQuery()
            .deploymentId(deploymentId)
            .singleResult();
}
```

需要在页面中设置回显字段

```
var options = {
    url: prefix + "/list",
    modalName: "流程定义明细",
    queryParams:queryParams,
    columns: [{
        checkbox: true
    },
    {
        field: 'deploymentId',
        title: '部署ID',
        visible: false
    },
    {
        field: 'definitionInfo.name',
        title: '流程名称'
    },
    {
        field: 'description',
        title: '描述信息',
        formatter: function(value, row, index) {
            return $.table.tooltip(value, 5);;
        }
    },
    {
        field: 'deployTime',
        title: '部署时间'
    },
    {
        field: 'deployKey',
        title: '部署key'
    },
    {
        field: 'version',
        title: '版本号'
    },
    {
        field: 'resourceName',
        title: '流程文件',
        formatter:function(value, row, index) {
            var url = prefix+"/readResource?
deployId="+row.deploymentId+"&type=xml";
            return '<a class="btn btn-info btn-xs " href="' + url + '"
target="_blank"><i class="fa fa-search"></i></a> ';
```

```
        }
    },
    {
        field: 'diagramResourceName',
        title: '流程图',
        formatter:function(value, row, index) {
            var url = prefix+"/readResource?
deployId="+row.deploymentId+"&type=png";
            return '<a class="btn btn-info btn-xs " href="' + url + '"
target="_blank"><i class="fa fa-search"></i></a> ';
        }
    }]
};
```

6.2.3.4 查看流程图

1.如果查看xml的流程文件,我们可以直接使用RepositoryService获取就可以了.

2.如果查看png的流程图,我们可以使用 `activiti-image-generator` 插件根据xml文件生成png文件.

- 我们需要引入依赖

```
<!--activiti可以绘制流程的的依赖-->
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-image-generator</artifactId>
    <version>7.0.0.SR1</version>
</dependency>
```

- 在控制器中逻辑如下:

```
@RequestMapping(value = "/readResource")
public void readResource(String deployId, String type, HttpServletResponse
response)
        throws Exception {
    ProcessDefinition processDefinition =
processService.selectProcessDefinitionByDeploymentId(deployId);
    InputStream inputStream = null;
    if("xml".equals(type)){
        inputStream =
processService.getResourceAsStream(processDefinition.getDeploymentId(),
processDefinition.getResourceName());
    }else if("png".equals(type)){
        inputStream =
processService.getProcessImage(processDefinition.getId(),
Collections.EMPTY_LIST,Collections.EMPTY_LIST);
    }
    IOUtils.copy(inputStream,response.getOutputStream());
}
```

- 在ProcessService中业务逻辑如下

```
@Override
public InputStream getResourceAsStream(String deploymentId, String
resourceName) {
    return repositoryService.getResourceAsStream(deploymentId,resourceName);
```

```
}

@Override
public InputStream getProcessImage(String processDefinitionId,
                                   List<String> highLightedActivities,
                                   List<String> highLightedFlows) {
    BpmnModel model = repositoryService.getBpmnModel(processDefinitionId);
    ProcessDiagramGenerator generator = new
DefaultProcessDiagramGenerator();
    //generateDiagram(流程模型,需要高亮的节点,需要高亮的线条,后面三个参数都表示是字体)
    InputStream inputStream = generator.generateDiagram(model,
highLightedActivities, highLightedFlows,"宋体","宋体","宋体");
    return inputStream;
}
```

6.3.3 节点人员功能实现

6.3.3.1 添加基础代码

使用代码生成器完成,不重复演示.

6.3.3.2 新增节点

我们需要在新增的时候,可以给新增的节点设置对应的审核人(可以多选的操作).

考虑到每个节点的审核人个数不会太多,所以我们采取多选下拉框的方式.

我们使用项目中已经有的插件`select2`,因为项目中已经存在此功能,所以我们参考来完成即可.

select2参考文档

- 添加`select2`的CSS和JS引用

```
<th:block th:include="include :: select2-css" />
<th:block th:include="include :: select2-js" />
```

- 添加对应的html代码

```
<div class="form-group">
    <label class="col-sm-3 control-label">审核人：</label>
    <div class="col-sm-8">
        <select id="auditorId" class="form-control select2-multiple"
multiple>
            <option th:each="auditor:${auditors}"
th:value="${auditor.userId}" th:text="${auditor.userName}"></option>
        </select>
    </div>
</div>
```

- 后台需要查询出所有用户出来

```
/**
 * 新增流程定义节点信息
 */
@GetMapping("/add")
public String add(ModelMap mmap)
{
    List<SysUser> auditors = sysUserService.listAllAuditors();
    mmap.put("auditors",auditors);
    return prefix + "/add";
}
```

- 保存

  前端代码需要把多选框的内容一并提交.

```
var definitionInfoId = [[${definitionInfoId}]];
var prefix = ctx + "business/definitionNode"
$("#form-definitionNode-add").validate({
    focusCleanup: true
});

function submitHandler() {
    if ($.validate.form()) {
        var data = $("#form-definitionNode-add").serializeArray();
        var auditorIds = $.form.selectSelects("auditorId");
        data.push({"name": "auditorIds", "value": postIds});
        data.push({"name": "definitionInfoId", "value": definitionInfoId});
        $.operate.save(prefix + "/add", data);
    }
}
```

我们需要接受这个id集合信息，所以我们在`DefinitionNode.java`增加一个字段用户接受集合信息

```
public class DefinitionNode extends BaseEntity
{
    private static final long serialVersionUID = 1L;

    /** $column.columnComment */
    private Long id;

    /** 流程定义ID */
    private Long definitionInfoId;

    /** 节点key */
    @Excel(name = "节点key")
    private String nodeKey;

    /** 节点描述 */
    @Excel(name = "节点描述")
    private String nodeDescription;

    /** 审核人集合 */
    private Long[] auditorIds;
}
```

别忘记提供setter/getter方法

在后台我们需要去维护节点和审核人之间的关系.

```
/**
 * 新增流程定义节点信息
 *
 * @param definitionNode 流程定义节点信息
 * @return 结果
 */
@Override
@Transactional
public int insertDefinitionNode(DefinitionNode definitionNode)
{
    int count = definitionNodeMapper.insertDefinitionNode(definitionNode);
    definitionNodeMapper.insertRelation(definitionNode);
    return count;
}
```

同学们，可以思考一下.insertRelation这个SQL语句应该怎么写?需要用到什么标签?

```
<insert id="insertRelation">
    insert into t_definition_auditor(node_id,user_id) values
    <foreach collection="auditorIds" separator="," item="item">
        (#{id},#{item})
    </foreach>
</insert>
```

6.3.3.3 编辑节点

点击编辑的时候，需要回显之前选择的审核人.

回显功能实现:

- edit.html页面添加`select2`的CSS和JS引用

```
<th:block th:include="include :: select2-css" />
<th:block th:include="include :: select2-js" />
```

- 添加html

```
<div class="form-group">
    <label class="col-sm-3 control-label">审核人：</label>
    <div class="col-sm-8">
        <select id="auditorId" class="form-control select2-multiple"
multiple>
            <option th:each="auditor:${auditors}"
th:value="${auditor.userId}" th:text="${auditor.userName}"></option>
        </select>
    </div>
</div>
```

- 通过JS控制回显

```
var selectedAuditorIds = JSON.parse([[${selectedAuditorIds}]]);
var prefix = ctx + "business/definitionNode";
$(function(){
    $("#auditorId").val(selectedAuditorIds).trigger('change');;
})
$("#form-definitionNode-edit").validate({
    focusCleanup: true
});

function submitHandler() {
    if ($.validate.form()) {
        var data = $("#form-definitionNode-add").serializeArray();
        var auditorIds = $.form.selectSelects("auditorId");
        data.push({"name": "auditorIds", "value": auditorIds});
        $.operate.save(prefix + "/edit", data);
    }
}
```

- 后台返回数据

```
/**
 * 修改流程定义节点信息
 */
@GetMapping("/edit/{id}")
public String edit(@PathVariable("id") Long id, ModelMap mmap)
{
    List<SysUser> auditors = sysUserService.listAllAuditors();
    mmap.put("auditors",auditors);
    List<Long> selectedAuditorIds =
definitionNodeService.querySelectedAuditorIdsByNodeId(id);
    //注意要转成JSON字符串.
    mmap.put("selectedAuditorIds", JSON.toJSONString(selectedAuditorIds));
    DefinitionNode definitionNode =
definitionNodeService.selectDefinitionNodeById(id);
    mmap.put("definitionNode", definitionNode);
    return prefix + "/edit";
}
```

编辑功能实现:

- 保持完之后后台需要处理中间关系.先删除关系，然后再重新插入关系.

```
/**
 * 修改流程定义节点信息
 *
 * @param definitionNode 流程定义节点信息
 * @return 结果
 */
@Override
@Transactional
public int updateDefinitionNode(DefinitionNode definitionNode)
{
    definitionNodeMapper.deleteRelation(definitionNode.getId());
    int count = definitionNodeMapper.updateDefinitionNode(definitionNode);
    if(definitionNode.getAuditorIds()!=null &&
definitionNode.getAuditorIds().length>0){
```

```
            definitionNodeMapper.insertRelation(definitionNode);
        }
        return count;
    }
```

七、审核列表功能

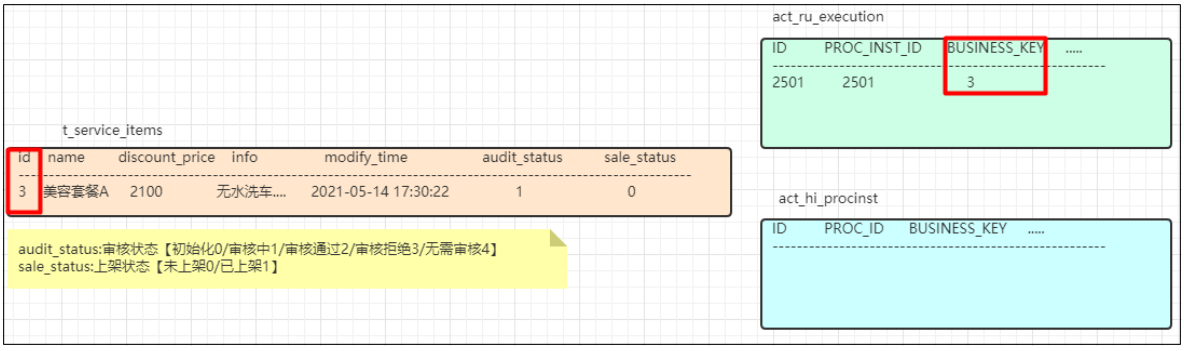
7.1 需求分析

在养修服务项中如果是套餐项的话,需要审核之后才能上架.所以我们需要先来完成这个审核功能.

关于审核状态的补充说明:

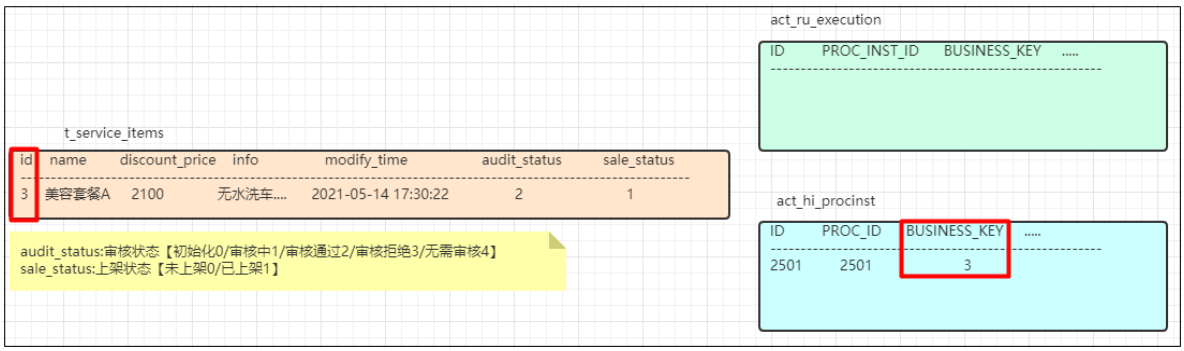
- 如果是非套餐,audit_status默认状态为4(无需审核),此时是可以直接上架商品的.
- 如果是套餐,audit_status默认状态0(初始化),此时是不能上架,需要审核通过才能上架

如果养修维护项是套餐,如果已经审核通过上架后.然后下架对套餐内容进行修改,这时候audit_status会变成0(初始化),需要重新审核才能重新上架.

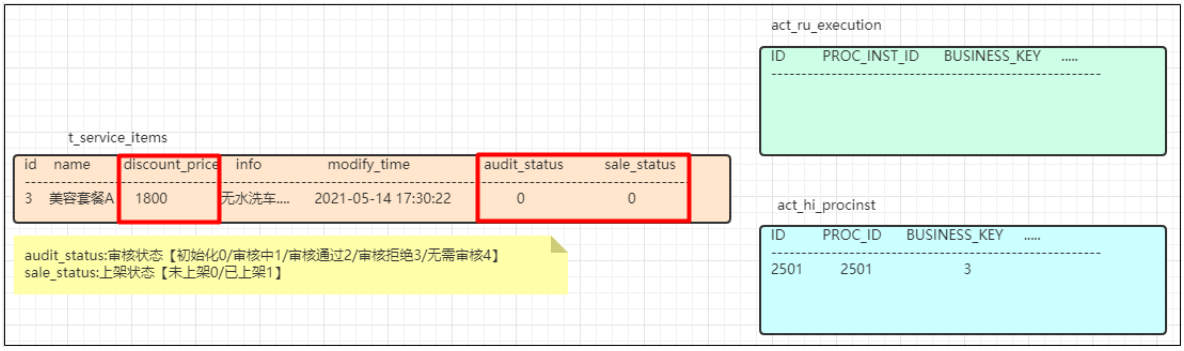


同学们思考一下,这样的方案可行吗？我们接着继续往后看.

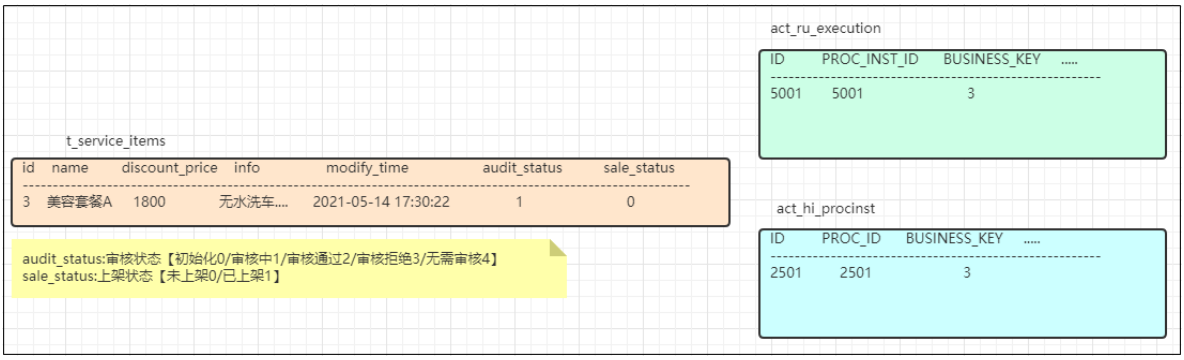
当流程适合完结束之后,在历史表中就会记录之前的流程信息.



假设我们对美容套餐A进行下架，然后进行价格的修改. 审核状态会被修改为0(初始化),需要重新审核才能重新上架.

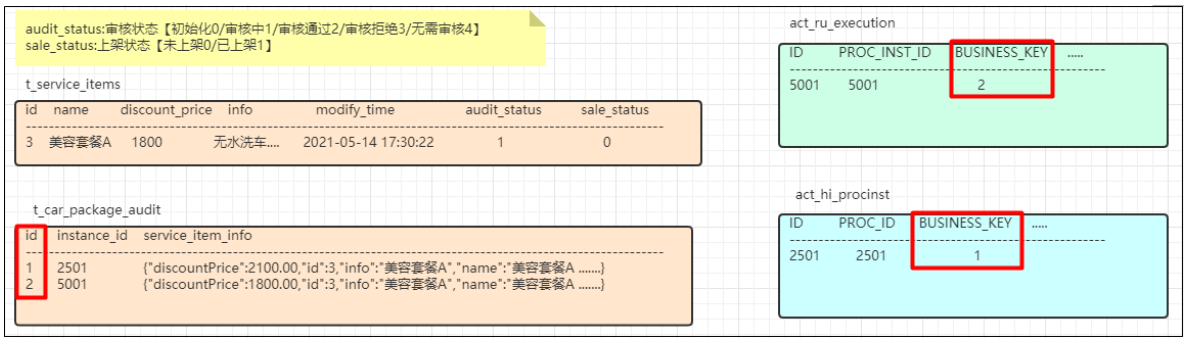


当我们发起审核之后.状态图如下.



因为我们当时`BUSINESS_KEY`存的是养修服务项的ID,那么历史表和运行时表都存储ID=3这套记录,但是这条记录的价格已经发生改变了.那么我们查看审核历史,查看业务数据的时候,关联出来的服务明细项就不对了,并不是当时审核时候的信息了(历史信息表中养修服务项的价格是2100元)，现在看到的是1800元.

所以我们不能这样设计.所以我们需要新增一张表,来存储发起审核那一刻养修明细项的具体内容，然后用这张表的ID作为`BUSINESS_KEY`.



当发起审核之后,在审核列表页面显示效果如下:

```
需要完成的功能：
1.撤销此次流程申请
2.进度查看
3.审批历史
```

7.2 表结构设计

- t_car_package_audit 汽车套餐审核记录表

字段	描述
id	主键
create_by	流程发起人
service_item_infoy	流程发起时间
service_item_info	服务项信息(JSON格式.)
process_definition_id	流程定义ID
instance_id	流程实例ID
auditors	当前节点审核人
status	状态【进行中0/审核拒绝1/审核通过2/审核撤销3】

```
CREATE TABLE `t_car_package_audit` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `create_by` bigint(20) DEFAULT NULL COMMENT '创建人',
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `service_item_info` varchar(4000) DEFAULT NULL COMMENT '服务项信息(JSON格式)',
  `status` tinyint(1) DEFAULT '0' COMMENT '状态【进行中0/审核拒绝1/审核通过2/审核撤销
3】',
  `instance_id` varchar(255) DEFAULT NULL COMMENT '流程实例ID',
  `auditors` varchar(4000) DEFAULT NULL COMMENT '当前节点审核人',
  `process_definition_id` varchar(255) DEFAULT NULL COMMENT '流程定义ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8 COMMENT='审核列表';
```

7.3 代码逻辑实现

7.3.1 基础代码生成

我们前面都是使用代码生成器生成代码的,但是有些公司是没有代码生成器的,所以我们接下来的操作就不使用代码生成器,自己把基础代码补充完整.

- 审核列表实体

```
public class CarPackageAudit extends BaseEntity {
    public static final String DEFINITION_KEY = "carPackageAudit";//汽车套餐流程
key
    public static final Integer STATUS_IN_ROGRESS = 0;//审核中
    public static final Integer STATUS_REJECT = 1;//审核拒绝
    public static final Integer STATUS_PASS = 2;//审核通过
```

```
    public static final Integer STATUS_CANCEL = 3;//审核撤销
    private static final long serialVersionUID = 1L;
    /** $column.columnComment */
    private Long id;
    /** 审核服务项ID */
    private ServiceItem serviceItem;
    /** 服务项信息(审核完成才存储，JSON格式.) */
    private String serviceItemInfo;
    /** 状态【审核中0/审核拒绝1/审核通过/审核撤销】 */
    private Integer status;
    /** 流程定义ID */
    private String processDefinitionId;
    /** 流程实例ID */
    private String instanceId;
    /** 任务ID */
    private String taskId;
    /** 任务名称 */
    private String taskName;
    /** 申请人 */
    private String createByName;
    /** 当前节点审核人**/
    private String auditors;
}
```

- 创建基础的文件

```
CarPackageAuditMapper.xml
CarPackageAuditMapper.java
ICarPackageAuditService.java
CarPackageAuditServiceImpl.java
CarPackageAuditController.java
```

页面等完成列表功能的时候再建立.

7.3.2 发起审核功能实现

我们需要在养修服务项页面中发起审核功能.

- 需要在`ServiceItemController.java`中接收前台的请求

```
@RequiresPermissions("business:serviceItem:startAudit")
@Log(title = "养修服务项", businessType = BusinessType.UPDATE)
@PostMapping( "/startAudit")
@ResponseBody
public AjaxResult startAudit(Long id){
    serviceItemService.startAudit(id);
    return success();
}
```

同学们思考一下,审核的逻辑有哪些?

```
1.根据ID查询养修服务项
2.判断养修服务项的状态(只有审核状态为初始化才能发起审核)
3.更新养修服务项状态为审核中.
4.插入审核列表信息
5.启动流程实例(把参数携带过去)
6.根据节点设置处理人
7.更新审核列表信息(审核人名称集合,流程实例ID)
```

- 在`ServiceItemServiceImpl.java`中添加审核逻辑

```
@Override
@Transactional
public void startAudit(Long id) {
    ServiceItem serviceItem = serviceItemMapper.selectServiceItemById(id);
    if(!serviceItem.getAuditStatus().equals(ServiceItem.AUDITSTATUS_INIT)){
        throw new BusinessException("非法操作,只有初始化状态服务项才能发起审核");
    }
    serviceItem.setAuditStatus(ServiceItem.AUDITSTATUS_AUDITING);
    //状态更新为审核中
    serviceItemMapper.updateServiceItem(serviceItem);
    CarPackageAudit carPackageAudit = new CarPackageAudit();
    carPackageAudit.setServiceItemInfo(JSON.toJSONString(serviceItem));
    carPackageAudit.setCreateBy(ShiroUtils.getUserId().toString());
    carPackageAuditService.insertCarPackageAudit(carPackageAudit);
    //启动流程
    Map<String, Object> variables = new HashMap<>();
    variables.put("money",serviceItem.getDiscountPrice().longValue());
    String businessKey = carPackageAudit.getId().toString();
    ProcessInstance instance = processService

.startProcessInstanceByKey(CarPackageAudit.DEFINITION_KEY,businessKey,variables)
;
    //查询当前任务节点
    Task currentTask = processService.getTaskByInstanceId(instance.getId());
    //获取任务节点名称
    String taskDefinitionKey = currentTask.getTaskDefinitionKey();
    //根据节点的key查询对应的审核人集合
    List<SysUser> auditors = definitionNodeService
            .queryAuditorsByTaskDefinitionKey(taskDefinitionKey);
    List<String> auditorsName = new ArrayList<>();
    for(SysUser sysUser:auditors){
        processService

.addCandidateUser(currentTask.getId(),sysUser.getUserId().toString());
        auditorsName.add(sysUser.getUserName());
    }
    //更新审核列表对象
    carPackageAudit.setProcessDefinitionId(instance.getProcessDefinitionId());
    carPackageAudit.setAuditors(JSON.toJSONString(auditorsName));
    carPackageAudit.setInstanceId(instance.getId());
    carPackageAuditService.updateCarPackageAudit(carPackageAudit);
}
```

7.3.3 列表功能实现

完成审核功之后,我们需要列表中展示出来.

- 编写控制器方法,跳转到指定页面

```
@RequiresPermissions("business:carPackageAudit:view")
@GetMapping()
public String carMaintenanceInfo()
{
    return prefix + "/carPackageAudit";
}
```

- 页面根据情况拷贝其他页面修改一下.
- 编写控制器方法,实现数据的查询

```
/**
 * 查询套餐审核列表
 */
@RequiresPermissions("business:carPackageAudit:list")
@PostMapping("/list")
@ResponseBody
public TableDataInfo list(CarPackageAudit carPackageAudit)
{
    startPage();
    carPackageAudit.setCreateBy(ShiroUtils.getUserId().toString());
    List<CarPackageAudit> list =
carPackageAuditService.selectCarPackageAuditList(carPackageAudit);
    return getDataTable(list);
}
```

- 业务层查询数据

```
@Override
public List<CarPackageAudit> selectCarPackageAuditList(CarPackageAudit
carPackageAudit) {
    List<CarPackageAudit> carMaintenanceInfos =
carPackageAuditMapper.selectCarPackageAuditList(carPackageAudit);
    Task task = null;
    for(CarPackageAudit audit:carMaintenanceInfos){
        audit.setServiceItem(JSON.parseObject(audit.getServiceItemInfo(),
ServiceItem.class));
        //根据流程实例id查询当前任务
        task = processService.getTaskByInstanceId(audit.getInstanceId());
        if(task!=null){
            audit.setTaskId(task.getId());
            audit.setTaskName(task.getName());
        }else{
            audit.setTaskName("已结束");
        }
    }
    return carMaintenanceInfos;
}
```

- 实现状态的回显

```
var statusDatas = [[${@dict.getType('cpa_status')}]];
```

```
$(function() {
    var options = {
        ...
        columns: [
        {
            field: 'status',
            title: '审核状态',
            formatter: function(value, row, index) {
                return $.table.selectDictLabel(statusDatas, value);
            }
        }]
    };
    $.table.init(options);
});
```

- 实现高级查询功能，在页面中增加按日期进行查询的功能.

```
<div class="select-list">
    <ul>
        <li class="select-time">
            <label>创建时间：</label>
            <input type="text" class="time-input" id="startTime" placeholder="开始时间" name="params[beginCreateTime]"/>
            <span>-</span>
            <input type="text" class="time-input" id="endTime" placeholder="结束时间" name="params[endCreateTime]"/>
        </li>
        <li>
            <a class="btn btn-primary btn-rounded btn-sm" onclick="$.table.search()"><i class="fa fa-search"></i> 搜索</a>
            <a class="btn btn-warning btn-rounded btn-sm" onclick="$.form.reset()"><i class="fa fa-refresh"></i> 重置</a>
        </li>
    </ul>
</div>
```

在Mapper.xml中增加查询条件

```
<if test="params.beginCreateTime != null and params.beginCreateTime != '' and params.endCreateTime != null and params.endCreateTime != ''">
    and create_time between #{params.beginCreateTime} and #{params.endCreateTime}
</if>
```

- 页面添加按钮

```
var actions = [];
    actions.push('<a class="btn btn-warning btn-xs" href="javascript:void(0)"
onclick="showHistoryDialog(\'' + row.instanceId + '\')"><i class="fa fa-list">
</i> 审批历史</a> ');
    actions.push('<a class="btn btn-info btn-xs" href="javascript:void(0)"
onclick="showProcessImgDialog(\'' + row.id + '\')"><i class="fa fa-image"></i>
进度查看</a> ');
    if (row.taskName.indexOf('已结束') === -1) {
        actions.push('<a class="btn btn-danger btn-xs" href="javascript:void(0)"
onclick="cancelApply(\'' + row.instanceId + '\',\'' + row.createBy + '\')"><i
class="fa fa-times"></i> 撤销</a> ');
    }
    return actions.join('');
```

7.3.4 进度查看功能实现

- 添加对应的JS方法,查看图片

```
function showProcessImgDialog(id) {
     var url = prefix + '/processImg/' + instanceId;
     $.modal.open("查看流程图", url);
}
```

- 后台控制器逻辑

```
@RequestMapping(value = "/processImg/{id}")
public void processImg(@PathVariable("id") String id, HttpServletResponse
response)
        throws Exception {
    CarPackageAudit carPackageAudit =
carPackageAuditService.getCarPackageAudit(id);
    InputStream inputStream=null;
    //如果状态为审核中或者审核拒绝，说明还正在流程中
    if(carPackageAudit.getStatus().equals(CarPackageAudit.STATUS_IN_ROGRESS) ||
            carPackageAudit.getStatus().equals(CarPackageAudit.STATUS_REJECT)
    ){
        //获取活动节点
        List<String> activeActivityIds =
processService.getActiveActivityIds(carPackageAudit.getInstanceId());
        inputStream =
processService.getProcessImage(carPackageAudit.getProcessDefinitionId(),activeAc
tivityIds,Collections.EMPTY_LIST);
    }else{
        inputStream =
processService.getProcessImage(carPackageAudit.getProcessDefinitionId(),Collecti
ons.EMPTY_LIST,Collections.EMPTY_LIST);
    }
    IOUtils.copy(inputStream,response.getOutputStream());
}
```

7.3.5 撤销功能实现

- 添加对应的JS方法,撤销审核流程.

```
function cancelApply(instanceId) {
    $.modal.confirm("确认要撤销申请吗?", function() {
        var url = prefix + "/cancelApply";
        var data = { "instanceId": instanceId};
        $.operate.submit(url, "post", "json", data);
    });
}
```

- 思考一下，当我们撤销的时候需要修改哪些数据?

```
1.根据流程实例ID查询流程实例对象
2.通过流程实例获取业务key
3.根据业务key查询审核对象，更新状态为撤销状态,审核人设置为空
4.获取服务项,更新审核状态为初始化
5.删除流程实例
```

- 业务逻辑处理

```
@Override
@Transactional
public void cancelApply(String instanceId) {
    //根据流程实例ID查询流程实例对象
    ProcessInstance instance =
processService.getProcessInstanceById(instanceId);
    //通过流程实例获取业务key
    String businessKey = instance.getBusinessKey();
    //根据业务key查询审核对象，更新状态为撤销状态
    CarPackageAudit carPackageAudit =
carPackageAuditMapper.getCarPackageAudit(businessKey);
    carPackageAudit.setStatus(CarPackageAudit.STATUS_CANCEL);
    carPackageAudit.setAuditors("");
    carPackageAuditMapper.updateCarPackageAudit(carPackageAudit);
    //获取服务项,更新审核状态为初始化
    ServiceItem serviceItem =
JSON.parseObject(carPackageAudit.getServiceItemInfo(),ServiceItem.class);
    serviceItem.setAuditStatus(ServiceItem.AUDITSTATUS_INIT);
    serviceItemService.updateServiceItemNoCondition(serviceItem);
    //执行此方法后未审批的任务 act_ru_task 会被删除，流程历史 act_hi_taskinst 不会被删
除，并且流程历史的状态为finished完成
    //删除流程实例
    processService.deleteProcessInstance(instanceId,"用户撤销");
}
```

7.3.6 审核历史功能实现

这个功能,我们留到后面再做.

八、我的待办功能

8.1 需求分析

当我们发起流程之后,根据流程图流转到下个节点中.每个节点都有对应的候选人.

此时我们需要根据当前登录用户,查询出他的待办任务集合.

当审核拒绝之后,发起人可以重新修改表单信息并重新申请

需要完成的功能:
1.审批功能
2.审批历史
3.进度查看(前面已经完成)
4.修改表单
5.重新申请

8.2 代码逻辑实现

8.2.1 基础代码生成

- 拷贝`carPackageAudit.html`并命名为`todoPage.html`
- 修改列表请求地址

```
<!DOCTYPE html>
<head>
    <th:block th:include="include :: header('我的待办')" />
</head>
<script th:inline="javascript">
    $(function() {
        var options = {
            url: prefix + "/todoList",
            modalName: "我的待办",
        };
        $.table.init(options);
    });
</script>
</html>
```

- 页面添加对应的按钮

```
var actions = [];
if(row.status==0){
    actions.push('<a class="btn btn-success btn-xs" href="javascript:void(0)"
onclick="showVerifyDialog(\'' + row.taskId + '\', \'' + row.taskName + '\')"><i
class="fa fa-edit"></i> 审批</a> ');
}else{
    actions.push('<a class="btn btn-success btn-xs" href="javascript:void(0)"
onclick="openServiceItemsEditPage(\'' + row.id + '\')"><i class="fa fa-edit">
</i> 修改表单</a> ');
    actions.push('<a class="btn btn-success btn-xs" href="javascript:void(0)"
onclick="reApply(\'' + row.taskId + '\', \'' + row.id + '\')"><i class="fa fa-
edit"></i> 重新申请</a> ');
}
actions.push('<a class="btn btn-warning btn-xs" href="javascript:void(0)"
onclick="showHistoryDialog(\'' + row.instanceId + '\')"><i class="fa fa-list">
</i> 审批历史</a> ');
actions.push('<a class="btn btn-info btn-xs" href="javascript:void(0)"
onclick="showProcessImgDialog(\'' + row.id + '\')"><i class="fa fa-image"></i>
进度查看</a> ');
return actions.join('');
```

8.2.2 列表功能实现

- 编写控制器方法,跳转到待办任务页面

```
@GetMapping("/todoPage")
public String todoPage()
{
    return prefix + "/todoPage";
}
```

- 编写控制器列表方法.

```
@PostMapping("/todoList")
@ResponseBody
public TableDataInfo todoList(CarPackageAudit carPackageAudit)
{
    List<CarPackageAudit> list =
carPackageAuditService.findTodoList(carPackageAudit);
    return getDataTable(list);
}
```

- 编写业务层查询待办任务集合方法

```
@Override
public List<CarPackageAudit> findTodoList(CarPackageAudit carPackageAudit) {
    //1.查询当前登录用户关于汽车审核所有的待办任务总数
    long count =
processService.selectTodoTaskCount(CarPackageAudit.DEFINITION_KEY,
ShiroUtils.getUserId().toString());
    if(count>0){
        PageDomain pageDomain = TableSupport.buildPageRequest();
        Integer pageNum = pageDomain.getPageNum();
```

```
        Integer pageSize = pageDomain.getPageSize();
         //2.查询当前登录用户关于汽车审核所有的待办任务集合
        List<Task> taskList = processService
                .selectTodoTaskList(CarPackageAudit.DEFINITION_KEY,
                        ShiroUtils.getUserId().toString(),
                        (pageNum-1)*pageSize,
                        pageSize);
        //3.遍历待办任务,查询对应的CarPackAgeAudit
        List<CarPackageAudit> resultList = new ArrayList<>();
        CarPackageAudit audit;
        for (Task task : taskList) {
            ProcessInstance processInstance = processService
                    .getProcessInstanceById(task.getProcessInstanceId());
            String businessKey = processInstance.getBusinessKey();
            audit = carPackageAuditMapper.getCarPackageAudit(businessKey);

 audit.setServiceItem(JSON.parseObject(audit.getServiceItemInfo(),ServiceItem.cl
ass));
            audit.setTaskName(task.getName());
            audit.setTaskId(task.getId());
            resultList.add(audit);
        }
        //封装成Page对象返回
        Page<CarPackageAudit> list = new Page<>();
        list.setTotal(count);
        list.setPageNum(pageNum);
        list.setPageSize(pageSize);
        list.addAll(resultList);
        return list;
    }else{
        //如果总数为0,返回空集合回去
        return Collections.EMPTY_LIST;
    }
}
```

```
@Override
public long selectTodoTaskCount(String definitionKey, String candidateUser) {
    return taskService.createTaskQuery()
            .processDefinitionKey(definitionKey)
            .taskCandidateUser(candidateUser)
            .count();
}
```

```
@Override
public List<Task> selectTodoTaskList(String definitionKey, String
candidateUser,Integer firstResult,Integer pageSize) {

    return taskService.createTaskQuery()
            .processDefinitionKey(definitionKey)
            .taskCandidateUser(candidateUser)
            .orderByTaskCreateTime()
            .desc()
            .listPage(firstResult,pageSize);
}
```

启动之后,进入列表页面会出现如下的错误

```
Caused by:
org.springframework.security.core.userdetails.UsernameNotFoundException: 1
```

原因是因为Activiti在代码中强耦合了SpringScurity,在使用

```
taskService.taskCandidateUser(candidateUser)
```

会调用SpringScurity中的UserDetailsService类型的bean中的loadUserByUsername方法.然后是找不到的,所以报错了.我们需要调整的是,写一个类实现UserDetailsService然后重写loadUserByUsername方法.

```
package com.ruoyi.business.config;

import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import java.util.Collections;

/**
 * Created by wolfcode
 */
@Component
public class SelfUserDetailsServiceImpl implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        return new User(username,"", Collections.EMPTY_LIST);
    }
}
```

8.2.3 审核功能实现

在待办任务中，我们需要对任务进行【审核通过】/【审核拒绝】的操作，同时需要给这次操作添加对应的审核批注.

- 在页面中增加按钮点击事件

```
function showVerifyDialog(taskId, taskName) {
    var url = prefix + "/showVerifyDialog/" + taskId;
    $.modal.open(taskName, url);
}
```

- 在控制器方法中实现页面跳转逻辑

```
@RequestMapping("/showVerifyDialog/{taskId}")
public String showVerifyDialog(@PathVariable("taskId") String taskId,ModelMap
mmap){
    mmap.put("taskId",taskId);
    return prefix+"/taskVerify";
}
```

- 新增页面`taskVerify.html`

```
<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org" >
<head>
    <th:block th:include="include :: header('审批')" />
</head>
<body class="white-bg">
    <div class="wrapper wrapper-content animated fadeInRight ibox-content">
        <form class="form-horizontal m" id="form-edit">
            <div class="form-group">
                <label class="col-sm-3 control-label">审批意见：</label>
                <div class="col-sm-8">
                    <select name="auditStatus" class="form-control m-b">
                        <option value="true">同意</option>
                        <option value="false">拒绝</option>
                    </select>
                </div>
            </div>
            <div class="form-group">
                <label class="col-sm-3 control-label">批注：</label>
                <div class="col-sm-8">
                    <textarea name="comment" class="form-control"></textarea>
                </div>
            </div>
        </form>
    </div>
    <th:block th:include="include :: footer" />
    <script th:inline="javascript">
        var prefix = "/business/carPackageAudit";
        $("#form-leave-edit").validate({
            focusCleanup: true
        });
        function submitHandler() {
            if ($.validate.form()) {
                var taskId = [[${taskId}]];
                $.operate.save(prefix + "/complete/" + taskId, $('#form-
edit').serialize());
            }
        }
    </script>
</body>
</html>
```

同学们,思考一下,审核有哪些逻辑?

```
1.根据任务Id查询任务对象
2.给任务添加批注,设置流程变量,认领任务，完成任务
3.查询流程实例下一个任务.
```

```
4.判断是否有下一个任务.
  4.1如果有下一个任务
     判断审核通过还是审核拒绝
     4.1.1如果是审核通过
          根据任务的key查询对应的审核人集合
          给任务设置候选人
     4.1.2如果是审核拒绝
          查询任务的发起人.
          给任务设置候选人(发起人)
          设置审核对象的状态为（拒绝）
     更新审核对象
  4.2如果没有下一个任务(流程已经结束,审核通过)
     修改审核对象的状态
     修改养修明细项的状态
```

```
@Override
@Transactional
public void complate(String taskId, String auditStatus, String comment) {
    boolean auditStatusBoolean = BooleanUtils.toBoolean(auditStatus);
    String commentStr = auditStatusBoolean?"【同意】":"【拒绝】";
    if(StringUtils.isNotEmpty(comment)){
        commentStr +=comment;
    }
    //1.根据任务Id查询任务对象
    Task task = processService.getTaskByTaskId(taskId);
    //这步需要放在前面
    ProcessInstance instance =
processService.getProcessInstanceById(task.getProcessInstanceId());
    //3.添加批注,设置流程变量,领取任务,完成任务

 processService.claimAndComplateTask(task,ShiroUtils.getUserId().toString(),audi
tStatusBoolean,commentStr);
    //4.完成任务后查询流程实例的下一个任务.
    Task nextTask =
processService.getTaskByInstanceId(task.getProcessInstanceId());

    CarPackageAudit audit =
carPackageAuditMapper.getCarPackageAudit(instance.getBusinessKey());
    //5.判断流程是否已经结束
    if(nextTask!=null){
        //5.1任务没有结束
        //需要指定下一个任务的候选人
        List<String> auditors = new ArrayList<>();
        //判断审核通过和是审核拒绝
        if(auditStatusBoolean){
            String taskDefinitionKey = nextTask.getTaskDefinitionKey();
            List<SysUser> auditorIdList =
definitionNodeService.queryAuditorsByTaskDefinitionKey(taskDefinitionKey);
            for(SysUser auditor:auditorIdList){

 processService.addCandidateUser(nextTask.getId(),auditor.getUserId().toString()
);
                auditors.add(auditor.getUserName());
            }
        }else{
            String userId = audit.getCreateBy();
            auditors.add(audit.getCreateByName());
```

```
            processService.addCandidateUser(nextTask.getId(),userId);
            audit.setStatus(CarPackageAudit.STATUS_REJECT);
        }
        audit.setAuditors(JSON.toJSONString(auditors));
        carPackageAuditMapper.updateCarPackageAudit(audit);
    }else{
        //5.2 任务结束
        //修改CarPackageAudit的状态,审核人字段设置为空
        audit.setAuditors("");
        audit.setStatus(CarPackageAudit.STATUS_PASS);
        carPackageAuditMapper.updateCarPackageAudit(audit);
        //修改养修明细项审核状态
        ServiceItem serviceItem =
JSON.parseObject(audit.getServiceItemInfo(),ServiceItem.class);
        serviceItem.setAuditStatus(ServiceItem.AUDITSTATUS_APPROVED);
        serviceItemService.updateServiceItemNoCondition(serviceItem);
    }
}
```

8.2.3 修改表单功能实现

如果流程审核拒绝之后，发起人可以通过重新修改养修明细项的细节，然后重新发起流程申请.

- 在页面中增加按钮点击事件

```
function openServiceItemsEditPage(carPackageAuditId){
    var url = prefix + "/openServiceItemsEditPage/" + carPackageAuditId;
    $.modal.open("修改表单", url);
}
```

- 控制器逻辑

```
@GetMapping("/openServiceItemsEditPage/{carPackageAuditId}")
public String edit(@PathVariable("carPackageAuditId") String carPackageAuditId,
ModelMap mmap)
{
    CarPackageAudit audit =
carPackageAuditService.getCarPackageAudit(carPackageAuditId);
    ServiceItem serviceItem =
JSON.parseObject(audit.getServiceItemInfo(),ServiceItem.class);
    mmap.put("serviceItem", serviceItem);
    mmap.put("carPackageAuditId", carPackageAuditId);
    return prefix + "/serviceItemEdit";
}
```

- 拷贝 `serviceItem/edit.html` 到目录 `carPackageAudit` 下，文件命名为 `serviceItemEdit.html`
- 页面中增加隐藏域carPackageAuditId

```
<input name="carPackageAuditId"  th:value="${carPackageAuditId}" type="hidden">
```

- 修改页面的请求地址

```
var prefix = ctx + "business/carPackageAudit";
$("#form-serviceItem-edit").validate({
    focusCleanup: true
});

function submitHandler() {
    if ($.validate.form()) {
        $.operate.save(prefix + "/serviceItemUpdate", $('#form-serviceItem-
edit').serialize());
    }
}
```

- 后台控制器逻辑

```
@PostMapping("/serviceItemUpdate")
@ResponseBody
public AjaxResult editSave(String carPackageAuditId,ServiceItem serviceItem)
{
    carPackageAuditService.updateServiceItem(carPackageAuditId,serviceItem);
    return success();
}
```

```
@Override
@Transactional
public void updateServiceItem(String carPackageAuditId, ServiceItem serviceItem)
{
    serviceItemService.updateServiceItemNoCondition(serviceItem);
    CarPackageAudit carPackageAudit =
carPackageAuditMapper.getCarPackageAudit(carPackageAuditId);
    carPackageAudit.setServiceItemInfo(JSON.toJSONString(serviceItem));
    carPackageAuditMapper.updateCarPackageAudit(carPackageAudit);
}
```

8.2.4 重新申请功能实现

如果流程被审核拒绝之后,运行重新调整养修服务项的内容,然后重新发起审核流程.

- 在页面中增加按钮点击事件

```
function reApply(taskId,carPackageAuditId){
    $.modal.confirm("确认需要重新发起申请吗", function() {
        $.operate.post(prefix + "/reApply", { "taskId": taskId,
"carPackageAuditId": carPackageAuditId });
    })
}
```

- 控制器逻辑

```
@PostMapping( "/reApply")
@ResponseBody
public AjaxResult reApply(String taskId,String carPackageAuditId) {
    carPackageAuditService.reApply(taskId,carPackageAuditId);
    return success();
}
```

同学们思考一下,重新申请逻辑?

```
1.根据任务ID查询任务对象
2.重新设置流程变量
3.领取任务并完成任务
4.完成任务后查询流程实例的下一个任务.
5.需要指定下一个任务的候选人
6.更新审核对象CarPackageAudit(更新状态和审核人集合)
```

- 业务层逻辑代码

```
@Override
public void reApply(String taskId, String carPackageAuditId) {
    //1.根据任务ID查询任务对象
    Task task = processService.getTaskByTaskId(taskId);
    //2.重新设置流程变量
    CarPackageAudit carPackageAudit =
carPackageAuditMapper.getCarPackageAudit(carPackageAuditId);
    ServiceItem serviceItem =
JSON.parseObject(carPackageAudit.getServiceItemInfo(),ServiceItem.class);

 processService.setVariable(taskId,"money",serviceItem.getDiscountPrice().longVa
lue());
    //3.领取任务并完成任务

 processService.claimAndComplateTask(task,ShiroUtils.getUserId().toString(),true
,"重新申请");
    //4.完成任务后查询流程实例的下一个任务.
    Task nextTask =
processService.getTaskByInstanceId(task.getProcessInstanceId());
    ProcessInstance instance =
processService.getProcessInstanceById(task.getProcessInstanceId());
    //5.需要指定下一个任务的候选人
    List<String> auditors = new ArrayList<>();
    String taskDefinitionKey = nextTask.getTaskDefinitionKey();
    List<SysUser> auditorIdList =
definitionNodeService.queryAuditorsByTaskDefinitionKey(taskDefinitionKey);
    for(SysUser auditor:auditorIdList){

 processService.addCandidateUser(nextTask.getId(),auditor.getUserId().toString()
);
        auditors.add(auditor.getUserName());
    }
    //6.更新审核对象CarPackageAudit
    carPackageAudit.setStatus(CarPackageAudit.STATUS_IN_ROGRESS);
    carPackageAudit.setAuditors(JSON.toJSONString(auditors));
    carPackageAuditMapper.updateCarPackageAudit(carPackageAudit);
}
```

8.2.5 进度查看功能实现

- 在页面中增加按钮点击事件

```
function showProcessImgDialog(id) {
    var url = prefix + '/processImg/' + id;
    $.modal.open("查看流程图", url);
}
```

8.2.6 审核历史功能实现

这个功能,我们留到后面再做.

九、我的已办功能

9.1 需求分析

当我们审核完成之后,我们可以再历史信息中查看到自己审核过哪些任务.


```
需要完成的功能:
1.我的已办列表
2.审核历史功能
3.进度查看功能
```

9.2 代码逻辑实现

9.2.1 基础代码生成

- 拷贝`carPackageAudit.html`并命名为`donePage.html`
- 修改列表请求地址

```
<!DOCTYPE html>
<head>
    <th:block th:include="include :: header('我的已办')" />
</head>
<script th:inline="javascript">
    $(function() {
        var options = {
            url: prefix + "/doneList",
            modalName: "我的已办",
        };
        $.table.init(options);
    });
</script>
</html>
```

- 添加按钮

```
var actions = [];
actions.push('<a class="btn btn-warning btn-xs" href="javascript:void(0)"
onclick="showHistoryDialog(\'' + row.id + '\')"><i class="fa fa-list"></i> 审批历
史</a> ');
actions.push('<a class="btn btn-info btn-xs" href="javascript:void(0)"
onclick="showProcessImgDialog(\'' + row.instanceId + '\')"><i class="fa fa-
image"></i> 进度查看</a> ');
return actions.join('');
```

- 页面列表中增加`已办任务名称`和`任务完成时间`两项展示项

```
{
    field: 'taskName',
    title: '已办任务名称'
},
{
    field: 'doneTime',
    title: '任务完成时间'
}
```

- 在`CarPackageAudit.java`中增加字段`doneTime`

```
/**任务完成时间**/
private Date doneTime;
```

9.2.2 我的已办列表功能

- 编写控制器方法,跳转到已办任务页面

```
@GetMapping("/donePage")
public String donePage()
{
    return prefix + "/donePage";
}
```

- 编写控制器列表方法.

```
@PostMapping("/doneList")
@ResponseBody
public TableDataInfo doneList(CarPackageAudit carPackageAudit)
{
    List<CarPackageAudit> list =
carPackageAuditService.findDoneList(carPackageAudit);
    return getDataTable(list);
}
```

- 业务层方法

```
@Override
public List<CarPackageAudit> findDoneList(CarPackageAudit carPackageAudit) {
```

```
    long count =
processService.selectDoneTaskCount(CarPackageAudit.DEFINITION_KEY,ShiroUtils.get
UserId().toString());
    if(count>0){
        PageDomain pageDomain = TableSupport.buildPageRequest();
        Integer pageNum = pageDomain.getPageNum();
        Integer pageSize = pageDomain.getPageSize();
        List<HistoricTaskInstance> taskList = processService
                .selectDoneTaskList(CarPackageAudit.DEFINITION_KEY,
                        ShiroUtils.getUserId().toString(),
                        (pageNum-1)*pageSize,
                        pageSize);
        //3.遍历待办任务,查询对应的CarPackAgeAudit
        List<CarPackageAudit> resultList = new ArrayList<>();
        CarPackageAudit audit;
        for (HistoricTaskInstance task : taskList) {
            HistoricProcessInstance historicProcessInstance = processService

.getHistoricProcessInstanceById(task.getProcessInstanceId());
            String businessKey = historicProcessInstance.getBusinessKey();
            audit = carPackageAuditMapper.getCarPackageAudit(businessKey);

 audit.setServiceItem(JSON.parseObject(audit.getServiceItemInfo(),ServiceItem.cl
ass));
            audit.setTaskName(task.getName());
            audit.setDoneTime(task.getEndTime());
            audit.setTaskId(task.getId());
            resultList.add(audit);
        }
        Page<CarPackageAudit> list = new Page<>();
        list.setTotal(count);
        list.setPageNum(pageNum);
        list.setPageSize(pageSize);
        list.addAll(resultList);
        return list;
    }else{
        return Collections.EMPTY_LIST;
    }
}
```

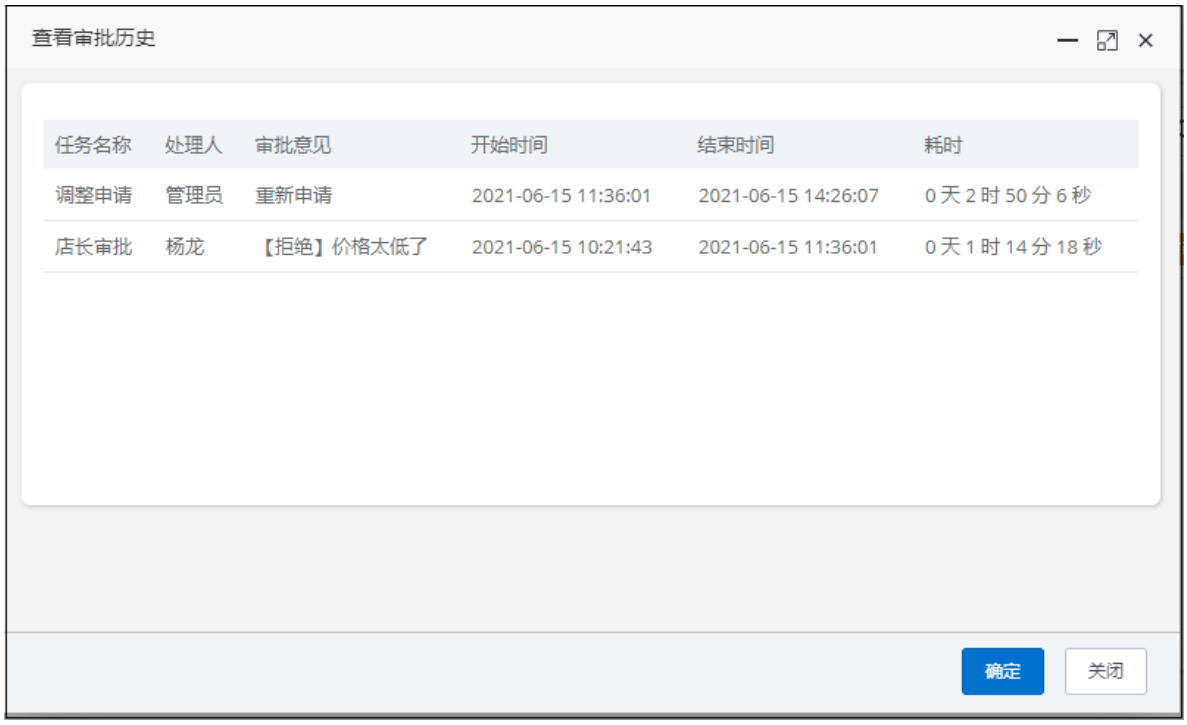
```
@Override
public long selectDoneTaskCount(String definitionKey, String userId) {
    return historyService
            .createHistoricTaskInstanceQuery()
            .processDefinitionKey(definitionKey)
            .taskAssignee(userId)
            .finished()
            .count();
}
```

```
@Override
public List<HistoricTaskInstance> selectDoneTaskList(String definitionKey,
String userId, Integer firstResult, Integer pageSize) {
    return historyService
            .createHistoricTaskInstanceQuery()
            .processDefinitionKey(definitionKey)
            .taskAssignee(userId)
            .finished()
            .orderByHistoricTaskInstanceEndTime()
            .desc()
            .listPage(firstResult,pageSize);
}
```

```
@Override
public HistoricProcessInstance getHistoricProcessInstanceById(String
processInstanceId) {
    return historyService
            .createHistoricProcessInstanceQuery()
            .processInstanceId(processInstanceId)
            .singleResult();
}
```

9.2.3 审核历史功能

我们可以点击审核历史看到历史审核人信息以及审核批注信息



- 在页面中增加按钮点击事件

```
/* 查看审批历史 */
function showHistoryDialog(instanceId) {
    var url = prefix + '/historyList/' + instanceId;
    $.modal.open("查看审批历史", url);
}
```

- 编写控制器逻辑，加载审核历史窗口

```
/**
 * 加载审批历史弹窗
 */
@GetMapping("/historyList/{instanceId}")
public String historyList(@PathVariable("instanceId") String instanceId,
ModelMap mmap)
{
    mmap.put("instanceId", instanceId);
    return prefix + "/historyList";
}
```

- 新建`historyList.html`,内容如下

```
<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org" >
<head>
    <th:block th:include="include :: header('审批历史')" />
</head>

<body class="gray-bg">
    <div class="container-div">
        <div class="row">
            <div class="col-sm-12 select-table table-striped">
                <table id="bootstrap-table"></table>
            </div>
        </div>
    </div>
    <th:block th:include="include :: footer" />
    <script th:inline="javascript">
        var prefix = ctx + "business/carPackageAudit";
        var instanceId=[[${instanceId}]];
        $(function() {
            var options = {
                url: prefix + "/listHistory?instanceId="+instanceId,
                queryParams: queryParams,
                sortName: "createTime",
                sortOrder: "desc",
                modalName: "审批历史",
                showSearch: false,
                showRefresh: false,
                showToggle: false,
                showColumns: false,
                clickToSelect: false,
                rememberSelected: false,
                pagination: false,
                columns: [{
                    field: 'activityId',
                    title: '活动ID',
                    visible: false
                },
                {
                    field: 'activityName',
                    title: '任务名称'
                },
```

```
                {
                    field: 'assigneeName',
                    title: '处理人'
                },
                {
                    field: 'comment',
                    title: '审批意见'
                },
                {
                    field: 'startTime',
                    title: '开始时间'
                },
                {
                    field: 'endTime',
                    title: '结束时间'
                },
                {
                    field: 'durationInMillis',
                    title: '耗时',
                    formatter: function(value, row, index) {
                        if (!value) return '未知';
                        return formatTotalDateSub(value / 1000);
                    }
                }]
            };
            $.table.init(options);
        });
        /**
         * 计算出相差天数
         * @param secondSub
         */
        function formatTotalDateSub (secondSub) {
            var days = Math.floor(secondSub / (24 * 3600));      // 计算出小时数
            var leave1 = secondSub % (24*3600) ;                 // 计算天数后剩余的
毫秒数
            var hours = Math.floor(leave1 / 3600);               // 计算相差分钟数
            var leave2 = leave1 % (3600);                        // 计算小时数后剩余
的毫秒数
            var minutes = Math.floor(leave2 / 60);               // 计算相差秒数
            var leave3 = leave2 % 60;                            // 计算分钟数后剩余
的毫秒数
            var seconds = Math.round(leave3);
            return days + " 天 " + hours + " 时 " + minutes + " 分 " + seconds +
' 秒';
        }
    </script>
</body>
</html>
```

- 新建实体，用于列表展示

```
public class HistoricActivity extends HistoricActivityInstanceEntityImpl {
    /** 审批批注 */
    private String comment;
    /** 办理人姓名 */
    private String assigneeName;
}
```

- 控制器逻辑

```
@PostMapping("/listHistory")
@ResponseBody
public TableDataInfo listHistory(String instanceId) {
    List<HistoricActivity> list =
carPackageAuditService.selectHistoryList(instanceId);
    return getDataTable(list);
}
```

- 业务层逻辑

```
@Override
public List<HistoricActivity> selectHistoryList(String instanceId) {
    //查询集合
    List<HistoricActivityInstance> instanceList = processService
            .selectHistoryTaskList(instanceId);
    List<HistoricActivity> resultList = new ArrayList<>();
    //遍历集合
    for (HistoricActivityInstance instance : instanceList) {
        HistoricActivity historicActivity = new HistoricActivity();
        //拷贝属性
        BeanUtils.copyProperties(instance, historicActivity);
        //查询审核批注

 historicActivity.setComment(processService.getTaskComment(instance.getTaskId())
);
        String assignee = instance.getAssignee();
        if(StringUtils.isNotEmpty(assignee)){
            SysUser sysUser =
sysUserMapper.selectUserById(Long.parseLong(assignee));
            if (sysUser != null) {
                //设置审核人姓名
                historicActivity.setAssigneeName(sysUser.getUserName());
            }
        }
        resultList.add(historicActivity);
    }
    return resultList;
}
```

```
@Override
public List<HistoricActivityInstance> selectHistoryTaskList(String instanceId) {

    return historyService.createHistoricActivityInstanceQuery()
            .processInstanceId(instanceId)
            .activityType("userTask")
            .finished()
```

```
            .orderByHistoricActivityInstanceStartTime()
            .desc()
            .list();
}

@Override
public String getTaskComment(String taskId) {
    List<Comment> taskComments = taskService.getTaskComments(taskId, "comment");
    if(taskComments!=null && taskComments.size()>0){
        return taskComments.get(0).getFullMessage();
    }else{
        return "";
    }
}
```

9.2.4 进度查看功能

前面已完成，参考前面即可.