程序开始!!!

```
@using LongQin.Models
@using LongQin.Infrastructures
@model LongQin.Models.FlowDesigner
@{
    ViewBag.Title = "流程设计器";
}
<blockquote class="layui-elem-quote">
    <span class="layui-breadcrumb">
        <a href="/FlowDesigner/FlowDesign/Index">流程设计器</a>
    </span>
</blockquote>
<div class="layui-fluid">
    <form class="layui-form layui-card" id="mainForm" lay-filter="mainForm">
        <input type="text" name="e2ent_id" value="{$data.id}" hidden="">
        <input type="hidden" name="flowId" id="flowId" value="@Model.FlowId" hidden="">
        <div class="layui-card-body">
            <div class="layui-row layui-col-space30">
                <div class="layui-col-md9" style="padding-top: 0;">
                    <div class="layui-card-header">
                        <div>
                            <button type="button" class="layui-btn layui-btn-sm
layui-btn-node" onclick="mflow.util.addrect()"><i class="layui-icon
layui-icon-component"></i>添加节点</button>
                            <button type="button" class="layui-btn layui-btn-sm
layui-btn-link" onclick="mflow.util.addpath()"><i class="layui-icon
layui-icon-link"></i>添加连线</button>
                            <div class="layui-form-mid layui-text-em"><i
class="layui-icon layui-icon-about"></i>选中元素后按Delete键删除</div>
                        </div>
                    </div>
                    <!-- // 流程设计区域 -->
                    <div id="flowBuilder" style="width: 100%;height:800px"></div>
                </div>
                <div class="layui-col-md3" style="padding-top: 0;">
                    <div class="layui-tab layui-tab-brief">
                        <ul class="layui-tab-title">
                            <li class="layui-this">元素属性</li>
                            <li>流程属性</li>
                        </ul>

                        <div class="layui-tab-content" id="layui-form-attribute">
                            <div class="layui-tab-item layui-form layui-show"
```

```
id="ElementPropertie" lay-filter="ElementPropertie"></div>
                            <div class="layui-tab-item">
                                <div class="layui-form-item">
                                    <label class="layui-form-label"><font
color="red">*</font>流程名</label>
                                    <div class="layui-input-inline">
                                        <input type="text" id="flowName" name="FlowName"
class="layui-input" value="@Model.FlowName" lay-verify="required|max" lay-reqtext="请填
写流程名" lay-max="25" lay-pretext="流程名" placeholder="请填写流程名">
                                    </div>
                                </div>
                                <div class="layui-form-item">
                                    <label class="layui-form-label"><font
color="red">*</font>流程类别</label>
                                    <div class="layui-input-inline">
                                        <select id="sort" lay-verify="required"
lay-reqtext="请选择一个流程类别">
                                            <option value="">请选择一个流程类别
</option>
                                            <option value="1">考勤类</option>
                                            <option value="2">行政类</option>
                                            <option value="3">业务类</option>
                                        </select>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>

        <div class="layui-footer layui-form-footer">
            <button class="layui-btn layui-btn-flow-commit" lay-filter="submitIframe"
type="button" lay-submit>提交</button>
        </div>
    </form>
</div>

<link rel="stylesheet" href="~/Content/formdesigner/module/formDesign/formdesign.css">
<!-- // 加载font-awesome图标 -->
<link href="~/Content/formdesigner/css/font-awesome.css" rel="stylesheet"
type="text/css" />
<style type="text/css">
```

```
    #layui-form-attribute .layui-form-label {
        width: 70px !important;
    }
</style>
<script type="text/javascript"
src="@Url.Content("~/Content/flowdesigner/raphael-min.js")"></script>
<script type="text/javascript"
src="@Url.Content("~/Content/flowdesigner/raphael-flow.js")"></script>

<script>
    var save_url = "@Url.Action("Save", "FlowDesign", new { area = "FlowDesigner" })";
    var pager;
    var mflow;
</script>

<script>
    /*$(function () {
        var v = $("#flowBuilder").width();
        var e = $("#flowBuilder").height();
        pager = new Raphael(document.getElementById("flowBuilder"), v, e);
        mflow = $.Flow.createNew(pager);
        mflow.init();
    });*/


    layui.use(['form', 'treeSelect'], function () {
        var form = layui.form;
        var treeSelect = layui.treeSelect;

        if ("@Model.FlowSort" != "0"){
            $("#sort").val("@Model.FlowSort");
        }
        form.render();
        var flowId = "@Model.FlowId";
        var v = $("#flowBuilder").width();
        var e = $("#flowBuilder").height();
        pager = new Raphael(document.getElementById("flowBuilder"), v, e);
        mflow = $.Flow.createNew(pager, form, treeSelect);
        if (flowId == "0") {
            mflow.init();
        }
        else {
            $.ajax({
                url: "/FlowDesigner/FlowDesign/GetFlowJson",    //后台数据请求地址
```

```
                type: "get",
                data: { flowId: flowId },
                async: false,
                success: function (result) {
                    if (result) {
                        var r = { data: eval(result) };
                        mflow.init(r);
                    }
                }
            });
        }

        $('body').on('click', '.layui-btn-flow-commit', function (e) {

            mflow.util.save();
        })
    })
</script>
```

程序结束!!!

程序开始!!!

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>LongQin</title>
    <link href="~/Content/layui/css/layui.css" rel="stylesheet" />
    <link href="~/Content/css/main.css" rel="stylesheet" />
    <link href="~/Content/layui/numberInput/css/theme.css" rel="stylesheet" />
    <link rel="icon" href="../../lq.png">
    <script src="~/Content/themes/blueskin/js/jquery-2.0.3.min.js"></script>
    <script src="~/Content/js/jquery.form.js"></script>
    <script src="~/Content/layui/layui.js"></script>
    <script src="~/Content/layui/treeTable.js"></script>
    <script src="~/Content/layui/treeSelect.js"></script>
    <script src="~/Content/layui/numberInput/numberInput.js"></script>

</head>
```

```
<body>
    <div class="layui-fluid">
        <form class="layui-form layui-card">
            <input type="hidden" id="id" name="Id" value="">
            <input type="hidden" id="jsonData" name="JsonData" value="">
            <input type="hidden" id="layerIndex" name="layerIndex" value="">
            <div class="layui-card-body">
                <div class="layui-row layui-col-space30">
                    <div class="layui-col-md2" style="padding-top: 0;">
                        <div class="layui-tab layui-tab-brief">
                            <ul class="layui-tab-title">
                                <li class="layui-this">表单组件</li>
                                <!--<li>表单模板</li>-->
                            </ul>
                            <div                    class="layui-tab-content"
id="layui-form-attribute">
                                <div      class="layui-tab-item      layui-form
layui-show">
                                    <div class="component">
                                        <div class="head">表单组件</div>
                                        <div            class="component-group"
id="sort_1">
                                            <ol           data-tag="input"><div
class="icon"><i class="layui-icon layui-icon-layer"></i></div><div class="name">
单行输入</div></ol>
                                            <ol        data-tag="textarea"><div
class="icon"><i     class="layui-icon     layui-icon-align-left"></i></div><div
class="name">多行输入</div></ol>
                                            <ol           data-tag="radio"><div
class="icon"><i class="layui-icon layui-icon-radio"></i></div><div class="name">
单选框</div></ol>
                                            <ol        data-tag="checkbox"><div
class="icon"><i class="layui-icon layui-icon-table"></i></div><div class="name">
多选框</div></ol>
                                            <ol          data-tag="select"><div
class="icon"><i class="layui-icon layui-icon-print"></i></div><div class="name">
下拉框</div></ol>
                                            <ol            data-tag="date"><div
class="icon"><i class="layui-icon layui-icon-time"></i></div><div class="name">
日期组件</div></ol>
                                            <ol     data-tag="colorpicker"><div
class="icon"><i class="layui-icon layui-icon-theme"></i></div><div class="name">
颜色选择器</div></ol>
                                            <ol          data-tag="slider"><div
```

```
class="icon"><i       class="layui-icon       layui-icon-slider"></i></div><div
class="name">滑块</div></ol>
                                        <ol            data-tag="rate"><div
class="icon"><i     class="layui-icon     layui-icon-rate-solid"></i></div><div
class="name">评分</div></ol>
                                        <ol          data-tag="switch"><div
class="icon"><i    class="layui-icon    layui-icon-switch"><k></k></i></div><div
class="name">开关</div></ol>
                                        <ol        data-tag="cascader"><div
class="icon"><i class="layui-icon layui-icon-cols"></i></div><div class="name">
级联选择器</div></ol>
                                        <ol          data-tag="editor"><div
class="icon"><i class="layui-icon layui-icon-form"></i></div><div class="name">
富文本</div></ol>
                                        <ol          data-tag="upload"><div
class="icon"><i       class="layui-icon       layui-icon-upload"></i></div><div
class="name">文件上传</div></ol>
                                        <ol            data-tag="tags"><div
class="icon"><i class="layui-icon fa-instagram"></i></div><div class="name">标签
选择器</div></ol>
                                        <!--<ol        data-tag="json"><div
class="icon"><i class="layui-icon fa-bars"></i></div><div class="name">JSON 组件
</div></ol>-->
                                    </div>
                                </div>

                                <div class="component">
                                    <div class="head">辅助组件</div>
                                    <div            class="component-group"
id="sort_2">
                                        <ol            data-tag="tips"><div
class="icon"><i class="layui-icon layui-icon-tips"></i></div><div class="name">
提示</div></ol>
                                        <!--<ol      data-tag="button"><div
class="icon"><i class="layui-icon layui-icon-layer"></i></div><div class="name">
按钮</div></ol>-->
                                        <ol            data-tag="note"><div
class="icon"><i class="layui-icon layui-icon-note"></i></div><div class="name">
便签</div></ol>
                                        <ol     data-tag="subtraction"><div
class="icon"><i    class="layui-icon    layui-icon-subtraction"></i></div><div
class="name">分割线</div></ol>
                                    </div>
                                </div>
```

```
                                    <div class="component">
                                        <div class="head">布局组件</div>
                                        <div            class="component-group"
id="sort_3">
                                            <!--<ol         data-tag="tab"><div
class="icon"><i  class="layui-icon  layui-icon-tabs  smallfont"></i></div><div
class="name">TAB 选项卡</div></ol>
                                            <ol            data-tag="grid"><div
class="icon"><i      class="layui-icon      layui-icon-layouts"></i></div><div
class="name">栅格</div></ol>-->
                                            <ol           data-tag="space"><div
class="icon"><i class="layui-icon layui-icon-more"></i></div><div class="name">
间距</div></ol>
                                        </div>
                                    </div>
                                </div>
                                <!-- // 加载远程表单模板 -->
                                <div                   id="layui-form-template"
class="layui-tab-item">
                                    <div id="item-list" class="item-list"></div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div class="layui-col-md8">
                        <div class="layui-card-header">
                            <div class="fr">
                                <button     type="button"     class="layui-btn
layui-btn-sm layui-btn-export"><i class="layui-icon layui-icon-export"></i>导出
</button>
                                <button     type="button"     class="layui-btn
layui-btn-sm layui-btn-import"><i class="layui-icon layui-icon-layer"></i>导入
</button>
                                <button     type="button"     class="layui-btn
layui-btn-sm         layui-btn-component">         <i         class="layui-icon
layui-icon-component"></i> 预览</button>
                                <button     type="button"     class="layui-btn
layui-btn-sm    layui-btn-danger    layui-form-clear"><i    class="layui-icon
layui-icon-delete"></i>清空</button>
                            </div>
                        </div>
                        <!-- // 表单设计区域 -->
                        <div id="formBuilder" style="width: 100%"></div>
                        <!-- // 表单隐藏域 -->
```

```
                    <textarea       id="formdesign"       name="formdesign"
hidden></textarea>
                </div>
                <div class="layui-col-md2" style="padding-top: 0;">
                    <div class="layui-tab layui-tab-brief">
                        <ul class="layui-tab-title">
                            <li class="layui-this">组件属性</ol>
                            <li>表单属性</ol>
                        </ul>

                        <div                        class="layui-tab-content"
id="layui-form-attribute">
                            <div      class="layui-tab-item      layui-form
layui-show" id="Propertie" lay-filter="Propertie"></div>
                            <div class="layui-tab-item">
                                <div class="layui-form-item">
                                    <label   class="layui-form-label"><font
color="red">*</font>表单 ID</label>
                                    <div class="layui-input-inline">
                                        <input    type="text"    id="formid"
name="formid" class="layui-input" value="" lay-verify="required|max|tablename"
lay-reqtext="请填写表单 ID" lay-max="25" lay-pretext="表单 ID" placeholder="请填
写表单 ID">
                                    </div>
                                </div>
                                <div class="layui-form-item">
                                    <label   class="layui-form-label"><font
color="red">*</font>表单名</label>
                                    <div class="layui-input-inline">
                                        <input   type="text"   id="formname"
name="formname"    class="layui-input"    value=""    lay-verify="required|max"
lay-reqtext="请填写表单名" lay-max="25" lay-pretext="表单名" placeholder="请填写
表单名">
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>

            <div class="layui-footer layui-form-footer">
                <button          class="layui-btn          layui-btn-subcommit"
```

```
lay-filter="submitIframe" type="button" lay-submit>提交</button>
            </div>
        </form>
    </div>

    <div class="layui-htmlview" style="display: none;">
        <textarea id="json-code"></textarea>
        <div class="layui-htmlbtn">
            <button  id="copy-code"  class="layui-btn  layui-hide"> 复 制 代 码
</button>
            <button  id="import-code"  class="layui-btn  layui-hide"> 导 入 数 据
</button>
        </div>
    </div>

    <link                                                    rel="stylesheet"
href="~/Content/formdesigner/module/formDesign/formdesign.css">
    <!-- // 全局加载第三方 JS -->
    <script src="~/Content/formdesigner/cascadata.js"></script>
    <script src="~/Content/formdesigner/tinymce/tinymce.min.js"></script>
    <!-- // 加载 font-awesome 图标 -->
    <link  href="~/Content/formdesigner/css/font-awesome.css"  rel="stylesheet"
type="text/css" />
    <script src="~/Content/formdesigner/Sortable/Sortable.js"></script>

    <script>
    var  commitUrl  =  "@Url.Action("set",  "formdesign",  new  {  area  =
"formdesigner" })";
    </script>

    <script>
        layui.config({
            version: true,
            base: '../../Content/formdesigner/module/'
        }).extend({
            cascader: 'cascader/cascader',
            tags: 'tags/tags',
            formDesign: 'formDesign/formDesign',
        }).use(['form', 'jquery', 'flow', 'formDesign', 'tags'], function () {
            var form = layui.form;
            var $ = layui.jquery;
            var tags = layui.tags;
            var formDesign = layui.formDesign;
```

```
            //自定义验证规则
            form.verify({
                max: function (value, item) {
                    var maxLen = item.getAttribute('lay-max');
                    var pretext = item.getAttribute('lay-pretext');
                    if (value.length > maxLen) {
                        return pretext + '不能大于' + maxLen + '个字符的长度';
                    }
                },
                min: function (value, item) {
                    var minLen = item.getAttribute('lay-min');
                    var pretext = item.getAttribute('lay-pretext');
                    if (value.length < minLen) {
                        return pretext + '至少' + minLen + '位';
                    }
                },
                tablename: [/^\w+$/, '表单 ID 只能由字母、数字或下划线组成']
            });

            var c = formDesign.render({
                elem: '#formBuilder'
                , eval: '#formdesign'
            });

            var data = $("#jsonData").val();
            if (data) {
                c.edit(data);
            }
        })
    </script>

</body>
</html>
```

程序结束!!!

程序开始!!!

```
(function ($) {
    var Flow = {
        createNew: function (pager, form, treeSelect) {
            var flow = {};
            flow.rectarr = {}; //节点集合
            flow.patharr = {}; //连线集合
```

```
flow.begin; //连线起点(添加连线时用到)
flow.tmp;  //临时点(为确定连线起点和终点)
flow.end;  //连线终点(添加连线时用到)
flow.forms;
flow.positions;
flow.users;
flow.config = {
    editable: true,
    lineHeight: 15,
    basePath: "",
    rect: {
        attr: {
            x: 300,
            y: 100,
            width: 100,
            height: 50,
            r: 5,
            fill: "90-#fff-#C0C0C0",
            stroke: "#000",
            "stroke-width": 3
        },
        data: { //节点属性数据
            id: 0,
            name: '新建节点',
            rectType: 0,
            formId: '',
            cooperation: 0,
            virtual: 0,
            beentrusted: 1,
            departmentId: 0,
            positionId: 0,
            userId: 0,
            userName: '',
            remark: '',
            isApproval: 0 // 是否审批节点，1-是，0-否
        },
        text: {
            text: "新建节点",
            "font-size": 12
        },
        margin: 5
    },
    path: {
        attr: {
```

```
path: {
    path: "M10 10L100 100",
    stroke: "#808080",
    fill: "none",
    "stroke-width": 4
},
arrow: {
    path: "M10 10L10 10",
    stroke: "#808080",
    fill: "#808080",
    "stroke-width": 4,
    radius: 4
},
fromDot: {
    width: 5,
    height: 5,
    stroke: "#fff",
    fill: "#000",
    cursor: "move",
    "stroke-width": 2
},
toDot: {
    width: 5,
    height: 5,
    stroke: "#fff",
    fill: "#000",
    cursor: "move",
    "stroke-width": 2
},
bigDot: {
    width: 5,
    height: 5,
    stroke: "#fff",
    fill: "#000",
    cursor: "move",
    "stroke-width": 2
},
smallDot: {
    width: 5,
    height: 5,
    stroke: "#fff",
    fill: "#000",
    cursor: "move",
    "stroke-width": 3
```

```
            }
        },
        data: { //连线属性数据
            id: 0,
            name: '',
            formId: '0',
            field: '', //条件字段
            operator: '', //条件符号，如‘=’
            operatorValue: '', //条件值
            remark: ''
        },
        text: {
            text: "",
            cursor: "move",
            background: "#000",
            "font-size": 12
        },
        textPos: {
            x: 0,
            y: -10
        }
    },
    restore: ""
};
flow.util = { //方法集
    isLine: function (g, f, e) {
        var d, c;
        if ((g.x - e.x) == 0) {
            d = 1;
        } else {
            d = (g.y - e.y) / (g.x - e.x);
        }
        c = (f.x - e.x) * d + e.y;
        if ((f.y - c) < 10 && (f.y - c) > -10) {
            f.y = c;
            return true;
        }
        return false;
    },
    center: function (d, c) {
        return {
            x: (d.x - c.x) / 2 + c.x,
            y: (d.y - c.y) / 2 + c.y
        }
```

```
},
nextId: (function () {
    var c = 0;
    return function () {
        return ++c;
    }
})(),
nextPathId: (function () {
    var c = 0;
    return function () {
        return ++c;
    }
})(),
connPoint: function (j, d) {
    var c = d,
    e = {
        x: j.x + j.width / 2,
        y: j.y + j.height / 2
    };
    var l = (e.y - c.y) / (e.x - c.x);
    l = isNaN(l) ? 0 : l;
    var k = j.height / j.width;
    var h = c.y < e.y ? -1 : 1,
    f = c.x < e.x ? -1 : 1,
    g,
    i;
    if (Math.abs(l) > k && h == -1) {
        g = e.y - j.height / 2;
        i = e.x + h * j.height / 2 / l;
    } else {
        if (Math.abs(l) > k && h == 1) {
            g = e.y + j.height / 2;
            i = e.x + h * j.height / 2 / l;
        } else {
            if (Math.abs(l) < k && f == -1) {
                g = e.y + f * j.width / 2 * l;
                i = e.x - j.width / 2;
            } else {
                if (Math.abs(l) < k && f == 1) {
                    g = e.y + j.width / 2 * l;
                    i = e.x + j.width / 2;
                }
            }
        }
```

```
    }
    return {
        x: i,
        y: g
    }
},
arrow: function (l, k, d) {
    var g = Math.atan2(l.y - k.y, k.x - l.x) * (180 / Math.PI);
    var h = k.x - d * Math.cos(g * (Math.PI / 180));
    var f = k.y + d * Math.sin(g * (Math.PI / 180));
    var e = h + d * Math.cos((g + 120) * (Math.PI / 180));
    var j = f - d * Math.sin((g + 120) * (Math.PI / 180));
    var c = h + d * Math.cos((g + 240) * (Math.PI / 180));
    var i = f - d * Math.sin((g + 240) * (Math.PI / 180));
    return [k, {
        x: e,
        y: j
    },
    {
        x: c,
        y: i
    }]
},
attr: function (ele, d) { //远程节点数据赋值到rect对象
    if (ele && d) {
        for (var p in d) {
            ele[p] = d[p];
        }
    }
},
addpath: function (c) { //添加连线
    if (flow.begin && flow.end) {
        if (!flow.util.checkpath(flow.begin, flow.end)) {
            var p = new flow.path(flow.begin, flow.end, c);
            flow.patharr[p.getId()] = p;

            //加载元素属性
            //flow.util.getPathPropertie(p);

            p.select();
        }
    }
},
getPathPropertie: function (p) {
```

```
                    $('#ElementPropertie').html('');
                    $('body').off('keyup', '#ElementPropertie #pathname');
                    var proHtml = '<div class="layui-form-item">';
                    proHtml += '<input type="hidden" value="' + p.getId() + '" id="' + 'pathid' + '">';
                    proHtml    +=    '<label    class="layui-form-label"><font color=\"red\">* </font>连线名称</label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml += '<input class="layui-input layui-keyup" value="' + p.attr({ path: 'name' }) + '" id="' + 'pathname' + '">';
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml  +=  '<label  class="layui-form-label"> 条 件 符 号 </label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml    +=    '<select    lay-filter="componentSelected" data-field="operator">';
                    var operatorType = [
                        {
                            title: '请选择',
                            value: '',
                        },
                        {
                            title: '=',
                            value: '=',
                        },
                        {
                            title: '!=',
                            value: '!=',
                        },
                        {
                            title: '>',
                            value: '>',
                        },
                        {
                            title: '<',
                            value: '<',
                        },
                        {
                            title: '>=',
                            value: '>=',
                        },
                        {
```

```
                        title: '<=',
                        value: '<=',
                    }
                ];
                for (let index = 0; index < operatorType.length; index++) {
                    const element = operatorType[index];
                    proHtml += '<option value="' + element.value + '"';
                    if (element.value == p.attr({ path: 'operator' })) {
                        proHtml += ' selected';
                    }
                    proHtml += '>' + element.title + '</option>';
                }
                proHtml += '</select>';
                proHtml += '</div></div>';

                proHtml += '<div class="layui-form-item">';
                proHtml  +=  '<label  class="layui-form-label">条件表单
</label>';
                proHtml += '<div class="layui-input-inline">';
                proHtml    +=    '<select    lay-filter="componentSelected"
name="formId" lay-search data-field="formId" id="formId"><option value="">请选择
</option>';
                if (flow.forms) {
                    for (let index = 0; index < flow.forms.length; index++) {
                        const element = flow.forms[index];
                        proHtml += '<option value="' + element.Id + '"';
                        if (element.Id == p.attr({ path: 'formId' })) {
                            proHtml += ' selected';
                        }
                        proHtml += '>' + element.FormName + '</option>';
                    }
                }
                proHtml += '</select>';
                proHtml += '</div></div>';

                proHtml += '<div class="layui-form-item">';
                proHtml  +=  '<label  class="layui-form-label">条件字段
</label>';
                proHtml += '<div class="layui-input-inline">';
                proHtml +='<select lay-filter="componentSelected" lay-search
data-field="field" id="field"><option value="">请选择</option>';
                var data = [];
                data.push({ 'label': '提交人', 'name': 'userId' });
                data.push({  'label':  '提交人职级',  'name':
```

```
'positionLevel' });
                    var fromFormId = p.attr({ path: 'formId' }) != '0' ?
p.attr({ path: 'formId' }) : p.from().attr('formId');
                    if (fromFormId) {
                        $.ajax({
                            url: "/FormDesigner/FormDesign/GetById",     //后台
数据请求地址
                            type: "get",
                            data: { id: fromFormId },
                            async: false,
                            success: function (slt) {
                                if (slt) {
                                    var formData = JSON.parse(slt).data;
                                    var                jsonData                =
JSON.parse(formData.JsonData);
                                    data = data.concat(jsonData);
                                    for (let index = 0; index < data.length;
index++) {
                                        const element = data[index];
                                        proHtml   +=   '<option   value="'   +
element.name + '"';
                                        if  (element.name  ==  p.attr({  path:
'field' })) {
                                            proHtml += ' selected';
                                        }
                                        proHtml  +=  '>'  +  element.label  +
'</option>';
                                    }
                                }
                            }
                        });
                    }
                    else {
                        for (let index = 0; index < data.length; index++) {
                            const element = data[index];
                            proHtml += '<option value="' + element.name + '"';
                            if (element.name == p.attr({ path: 'field' })) {
                                proHtml += ' selected';
                            }
                            proHtml += '>' + element.label + '</option>';
                        }
                    }
                    proHtml += '</select>';
                    proHtml += '</div></div>';
```

```
                    proHtml += '<div class="layui-form-item">';
                    proHtml += '<label class="layui-form-label">条件值</label>';
                    proHtml      +=      '<div      class="layui-input-inline"
id="operatorValueDiv">';
                    var selectFiled = p.attr({ path: 'field' });
                    if (selectFiled == 'positionLevel') {
                        proHtml  +=  '<select  lay-filter="componentSelected"
lay-search data-field="operatorValue">';
                        if (p.attr({ path: 'operatorValue' }) == 1) {
                            proHtml  +=  '<option  value="1"  selected> 基 层
</option><option value="2">中层</option><option value="3">高层</option>';
                        }
                        else if (p.attr({ path: 'operatorValue' }) == 2) {
                            proHtml += '<option value="1">基层</option><option
value="2" selected>中层</option><option value="3">高层</option>';
                        }
                        else if (p.attr({ path: 'operatorValue' }) == 2) {
                            proHtml += '<option value="1">基层</option><option
value="2">中层</option><option value="3" selected>高层</option>';
                        }
                        else {
                            proHtml += '<option value="1">基层</option><option
value="2">中层</option><option value="3">高层</option>';
                        }
                        proHtml += '</select>';
                    }
                    else if (selectFiled == 'userId') {
                        proHtml  +=  '<select  lay-filter="componentSelected"
lay-search data-field="operatorValue">';
                        if (flow.users) {
                            for (let index = 0; index < flow.users.length; index++)
{
                                const element = flow.users[index];
                                proHtml += '<option value="' + element.UserId +
'"';
                                if    (element.UserId    ==    p.attr({    path:
'operatorValue' })) {
                                    proHtml += ' selected';
                                }
                                proHtml += '>' + element.UserName + '</option>';
                            }
                        }
                        proHtml += '</select>';
```

```
                    }
                    else {
                        proHtml += '<input class="layui-input layui-keyup" value="' + p.attr({ path: 'operatorValue' }) + '" id="' + 'operatorValue' + '">';
                    }
                    proHtml += '</div></div>';

                    proHtml += '<div id="slideTest8" ></div>';
                    $('#ElementPropertie').html(proHtml);
                    form.render(null, 'ElementPropertie');

                    $('body').on('keyup', '#ElementPropertie #pathname', function (e) {
                        var id = $('#ElementPropertie #pathid').val();
                        //flow.patharr[id]['name'] = $(this).val();
                        flow.patharr[id].setattr('name', $(this).val());
                        flow.patharr[id].settext($(this).val());
                    })
                    $('body').on('keyup', '#ElementPropertie #operatorValue', function (e) {
                        var id = $('#ElementPropertie #pathid').val();
                        flow.patharr[id].setattr('operatorValue', $(this).val());
                    })
                    form.on('select(componentSelected)', function (data) {
                        var field = $(data.elem).data('field'), id = $('#ElementPropertie #pathid').val()
                            , element = flow.patharr[id];
                        element[field] = data.value;
                        element.setattr(field, data.value);

                        // 加载条件值选项
                        if (field == 'field') {
                            if (data.value == 'positionLevel') {
                                var subHtml = '<select lay-filter="componentSelected" lay-search data-field="operatorValue">';
                                if (p.attr({ path: 'operatorValue' }) == 1) {
                                    subHtml += '<option value="1" selected>基层</option><option value="2">中层</option><option value="3">高层</option>';
                                }
                                else if (p.attr({ path: 'operatorValue' }) == 2) {
                                    subHtml += '<option value="1">基层</option><option value="2" selected>中层</option><option value="3">高层
```

```
</option>';
                                }
                                else if (p.attr({ path: 'operatorValue' }) == 2)
{
                                    subHtml += '<option value="1"> 基 层
</option><option value="2"> 中 层 </option><option value="3" selected> 高 层
</option>';
                                }
                                else {
                                    subHtml += '<option value="1"> 基 层
</option><option value="2">中层</option><option value="3">高层</option>';
                                }
                                subHtml += '</select>';
                                $('#operatorValueDiv').html(subHtml);
                            }
                            else if (data.value == 'userId') {
                                var subHtml = '<select
lay-filter="componentSelected" lay-search data-field="operatorValue">';
                                if (flow.users) {
                                    for (let index = 0; index < flow.users.length;
index++) {
                                        const element = flow.users[index];
                                        subHtml += '<option value="' +
element.UserId + '"';
                                        if (element.UserId == p.attr({ path:
'operatorValue' })) {
                                            subHtml += ' selected';
                                        }
                                        subHtml += '>' + element.UserName +
'</option>';
                                    }
                                }
                                subHtml += '</select>';
                                $('#operatorValueDiv').html(subHtml);
                            }
                            else {
                                var subHtml = '<input class="layui-input
layui-keyup" value="' + p.attr({ path: 'operatorValue' }) + '" id="' +
'operatorValue' + '">';
                                $('#operatorValueDiv').html(subHtml);
                            }
                            form.render(null, 'ElementPropertie');
                        }
                        else if (field == 'formId') {
```

```
                            var subdata = [];
                            $('#field').html('');
                            var subHtml = '<option value="">请选择</option>';
                            subdata.push({ 'label': '提交人', 'name':
'userId' });
                            subdata.push({ 'label': '提交人职级', 'name':
'positionLevel' });
                            var fromFormId = data.value ? data.value :
p.from().attr('formId');
                            if (fromFormId) {
                                $.ajax({
                                    url:    "/FormDesigner/FormDesign/GetById",
//后台数据请求地址
                                    type: "get",
                                    data: { id: fromFormId },
                                    async: false,
                                    success: function (slt) {
                                        if (slt) {
                                            var formData = JSON.parse(slt).data;
                                            var            jsonData           =
JSON.parse(formData.JsonData);
                                            subdata = subdata.concat(jsonData);
                                            for (let index = 0; index <
subdata.length; index++) {
                                                const element = subdata[index];
                                                subHtml += '<option value="' +
element.name + '"';
                                                if (element.name == p.attr({ path:
'field' })) {
                                                    subHtml += ' selected';
                                                }
                                                subHtml += '>' + element.label +
'</option>';
                                            }
                                            $('#field').append($(subHtml));
                                        }
                                    }
                                });
                            }
                            else {
                                for (let index = 0; index < data.length; index++)
{
                                    const element = data[index];
                                    subHtml += '<option value="' + element.name +
```

```
'"';
                                    if (element.name == p.attr({path:'field'}))
{
                                        subHtml += ' selected';
                                    }
                                    subHtml +='>' + element.label + '</option>';
                                }
                                $('#field').append($(subHtml));
                            }
                            form.render('select');
                        }
                    })
                },
                checkpath: function (begin, end) { //检查连线是否存在
                    for (var p in flow.patharr) {
                        if (flow.patharr[p]) {
                            if ((flow.patharr[p].from().getId() == begin.getId()
&& flow.patharr[p].to().getId() == end.getId())
                                || (flow.patharr[p].from().getId() == end.getId()
&& flow.patharr[p].to().getId() == begin.getId())) {
                                return true;
                            }
                        }
                    }
                    return false;
                },
                addrect: function () { //添加节点
                    var p = new flow.rect();
                    flow.rectarr[p.getId()] = p;

                    //加载元素属性
                    //flow.util.getRectPropertie(p);

                    p.select();
                },
                getRectPropertie: function (p) {
                    $('#ElementPropertie').html('');
                    $('body').off('click', '.layui-btn-designer');
                    $('body').off('keyup', '#ElementPropertie #name');
                    var proHtml = '<div class="layui-form-item">';
                    proHtml += '<input type="hidden" value="' + p.getId() + '"
id="' + 'pid' + '">';
                    proHtml    +=    '<label    class="layui-form-label"><font
color=\"red\">* </font>节点名称</label>';
```

```
                    proHtml += '<div class="layui-input-inline">';
                    proHtml += '<input class="layui-input layui-keyup" value="' +
p.attr('name') + '" id="' + 'name' + '">';
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml    +=    '<label    class="layui-form-label"><font
color=\"red\">* </font>节点类型</label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml    +=    '<select    lay-filter="componentSelected"
data-field="rectType">';
                    var nodeType = [
                        {
                            title: '普通节点',
                            value: '0',
                        },
                        {
                            title: '分流节点',
                            value: '1',
                        },
                        {
                            title: '合流节点',
                            value: '2',
                        },
                        {
                            title: '分合流点',
                            value: '3',
                        }
                    ];
                    for (let index = 0; index < nodeType.length; index++) {
                        const element = nodeType[index];
                        proHtml += '<option value="' + element.value + '"';
                        if (element.value == p.attr('rectType')) {
                            proHtml += ' selected';
                        }
                        proHtml += '>' + element.title + '</option>';
                    }
                    proHtml += '</select>';
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml  +=  '<label  class="layui-form-label">是否审批
</label>';
                    proHtml += '<div class="layui-input-inline">';
```

```
                    if (p.attr('isApproval') == 0) {
                        proHtml += '<input type="radio" name="isApproval"
value="1" lay-filter="isApproval" title="是">';
                        proHtml += '<input type="radio" name="isApproval"
value="0" lay-filter="isApproval" title="否" checked>';
                    }
                    else {
                        proHtml += '<input type="radio" name="isApproval"
value="1" lay-filter="isApproval" title="是" checked>';
                        proHtml += '<input type="radio" name="isApproval"
value="0" lay-filter="isApproval" title="否">';
                    }
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item" id="nodeFormDiv">';
                    proHtml += '<label class="layui-form-label">节点表单
</label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml += '<select lay-filter="componentSelected"
name="formId" lay-search data-field="formId"><option value="">请选择</option>';
                    if (flow.forms) {
                        for (let index = 0; index < flow.forms.length; index++) {
                            const element = flow.forms[index];
                            proHtml += '<option value="' + element.Id + '"';
                            if (element.Id == p.attr('formId')) {
                                proHtml += ' selected';
                            }
                            proHtml += '>' + element.FormName + '</option>';
                        }
                    }
                    proHtml += '</select>';
                    proHtml += '<button type="button" class="layui-btn
layui-btn-sm layui-btn-designer">设计</button>';
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml += '<label class="layui-form-label">多人协作
</label>';
                    proHtml += '<div class="layui-input-inline">';
                    if (p.attr('cooperation') == 0) {
                        proHtml += '<input type="radio" name="cooperation"
value="1" lay-filter="cooperation" title="是">';
                        proHtml += '<input type="radio" name="cooperation"
value="0" lay-filter="cooperation" title="否" checked>';
```

```
                    }
                    else {
                        proHtml += '<input type="radio" name="cooperation" value="1" lay-filter="cooperation" title="是" checked>';
                        proHtml += '<input type="radio" name="cooperation" value="0" lay-filter="cooperation" title="否">';
                    }
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml += '<label class="layui-form-label">处理部门</label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml += '<input class="layui-input" lay-filter="departmentId" value="' + p.attr('departmentId') + '" id="' + 'departmentId' + '">';
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml += '<label class="layui-form-label">处理职位</label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml +='<select lay-filter="componentSelected" lay-search data-field="positionId"><option value="0">请选择</option>';
                    if (flow.positions) {
                        for (let index = 0; index < flow.positions.length; index++) {
                            const element = flow.positions[index];
                            proHtml += '<option value="' + element.PositionId + '"';
                            if (element.PositionId == p.attr('positionId')) {
                                proHtml += ' selected';
                            }
                            proHtml += '>' + element.PositionName + '</option>';
                        }
                    }
                    proHtml += '</select>';
                    proHtml += '</div></div>';

                    proHtml += '<div class="layui-form-item">';
                    proHtml +='<label class="layui-form-label">处理人</label>';
                    proHtml += '<div class="layui-input-inline">';
                    proHtml +='<select lay-filter="componentSelected" lay-search data-field="userId"><option value="0">请选择</option>';
```

```
                    if (flow.users) {
                        for (let index = 0; index < flow.users.length; index++) {
                            const element = flow.users[index];
                            proHtml += '<option value="' + element.UserId + '"';
                            if (element.UserId == p.attr('userId')) {
                                proHtml += ' selected';
                            }
                            proHtml += '>' + element.UserName + '</option>';
                        }
                    }
                    proHtml += '</select>';
                    proHtml += '</div></div>';

                    proHtml += '<div id="slideTest8" ></div>';
                    $('#ElementPropertie').html(proHtml);
                    form.render(null, 'ElementPropertie');

                    $('body').on('click', '.layui-btn-designer', function (e) {
                        layer.open({
                            type: 2
                            , content: '/FlowDesigner/FlowDesign/FormDesigner'
                            , area: 'auto'
                            , shade: false
                            , resize: true
                            , maxmin: false
                            , success: function (layero, index) {
                                layer.full(index);
                                var                 iframeWin                 =
window[layero.find('iframe')[0]['name']];
                                var elemMark = iframeWin.$('#layerIndex'); // 获
得 iframe 中某个输入框元素
                                elemMark.val(index);
                            }
                        });
                    })

                    treeSelect.render({
                        // 选择器
                        elem: '#departmentId',
                        // 异步获取下拉树需要显示的数据
                        data: '/System/Department/GetDepartmentTree',
                        // 异步加载方式：get/post，默认 get
                        type: 'get',
                        // 占位符
```

```
                placeholder: '处理部门',
                // 是否开启搜索功能：true/false，默认 false
                search: true,
                // 一些可定制的样式
                style: {
                    folder: {
                        enable: true
                    },
                    line: {
                        enable: true
                    }
                },
                // 点击节点回调
                click: function (d) {
                    var id = $('#ElementPropertie #pid').val()
                        , element = flow.rectarr[id];
                    element.setattr('departmentId', d.current.id);
                },
                // 加载完成后的回调函数
                success: function (d) {
                    //console.log(d);
                    // 选中节点,根据 id 筛选,一般修改时会有默认选中状态,
可以在这里设置
                    if ($("#departmentId").val() != '0') {
                        treeSelect.checkNode('departmentId',
$("#departmentId").val());
                    }
                }
            });

            $('body').on('keyup', '#ElementPropertie #name', function (e)
{
                var id = $('#ElementPropertie #pid').val();
                //flow.rectarr[id]['name'] = $(this).val();
                flow.rectarr[id].setattr('name', $(this).val());
                flow.rectarr[id].settext($(this).val());
            })
            form.on('radio(cooperation)', function (data) {
                var elem = data.elem; // 获得 radio 原始 DOM 对象
                var value = elem.value; // 获得 radio 值

                var id = $('#ElementPropertie #pid').val();
                flow.rectarr[id].setattr('cooperation', value);
            });
```

```
                form.on('radio(isApproval)', function (data) {
                    var elem = data.elem; // 获得 radio 原始 DOM 对象
                    var value = elem.value; // 获得 radio 值

                    var id = $('#ElementPropertie #pid').val();
                    flow.rectarr[id].setattr('isApproval', value);
                    var element = flow.rectarr[id];
                    if (value == 1) {
                        $('#nodeFormDiv').hide();
                        element['formId'] = 1;
                        element.setattr('formId', 1);
                    }
                    else {
                        $('#nodeFormDiv').show();
                        form.val('mainForm', { 'formId': '' });
                    }
                });
                form.on('select(componentSelected)', function (data) {
                    var field = $(data.elem).data('field'), id = $('#ElementPropertie #pid').val()
                        , element = flow.rectarr[id];
                    element[field] = data.value;
                    element.setattr(field, data.value);
                })
            },
            check: function () { //流程检查
                if (flow.patharr.length == 0 || flow.rectarr.length == 0) return false;
                return true;
            },
            save: function () { //保存
                var flowId = $("#flowId").val();
                var flowName = $("#flowName").val();
                if (!flowName) {
                    layer.msg("请输入流程名");
                    return;
                }
                if (flow.util.check()) {
                    var nodes = "";
                    for (var rect in flow.rectarr) {
                        if (flow.rectarr[rect]) {
                            nodes += flow.rectarr[rect].toJson() + ";";
                        }
                    }
```

```
                        if (nodes.substring(nodes.length - 1, nodes.length) == ";")
{
                            nodes = nodes.substring(0, nodes.length - 1);
                        }
                        var links = "";
                        for (var path in flow.patharr) {
                            if (flow.patharr[path]) {
                                links += flow.patharr[path].toJson() + ";";
                            }
                        }
                        if (links == "") {
                            layer.msg("请检查流程");
                            return;
                        }
                        if (links.substring(links.length - 1, links.length) == ";")
{
                            links = links.substring(0, links.length - 1);
                        }
                        var flowSort = $("#sort").val();
                        $.ajax({
                            url: "/FlowDesigner/FlowDesign/Save",    //后台数据
请求地址
                            type: "post",
                            data: { flowId: flowId, flowName: flowName, flowSort:
flowSort, nodes: nodes, links: links },
                            async: false,
                            success: function (result) {
                                if (result && result.success) {
                                    layer.msg("保存成功");
                                }
                                else {
                                    layer.msg("保存失败，请重试");
                                }
                            }
                        });
                    }
                },
                groupSeq: function (r) { //得到节点的序号
                    var beginNum = 0; //起点连线数量
                    var endNum = 0; //终点连线数量
                    for (var path in flow.patharr) {
                        if                (flow.patharr[path]                &&
flow.patharr[path].from().getId() == r.getId()) {
                            beginNum++;
```

```
                        }
                        if (flow.patharr[path] && flow.patharr[path].to().getId()
== r.getId()) {
                            endNum++;
                        }
                    }
                    if (beginNum > 0 && endNum == 0) { //起点
                        return 1;
                    }
                    else if (beginNum == 0 && endNum > 0) { //终点
                        return 9;
                    }
                    else {
                        return 2;
                    }
                }
            };
            flow.rect = function (rect) {
                var u = this;
                var nextId = flow.util.nextId();
                var g = "rect" + nextId;
                var a;
                if (rect) {
                    a = $.extend(true, {}, flow.config.rect, rect);
                }
                else {
                    a = flow.config.rect;
                    a.attr.y = 100 + (nextId - 1) * 120;
                }
                var t = pager.rect(a.attr.x, a.attr.y, a.attr.width, a.attr.height,
a.attr.r).attr(a.attr);
                flow.util.attr(t, a.data); //节点属性
                var f = pager.text(a.attr.x + a.attr.width / 2, a.attr.y +
a.attr.height / 2, a.text.text).attr(a.text);
                var n = pager.text(a.attr.x + 120, a.attr.y + 8, '').attr("fill",
"rgb(20, 165, 236)");
                var q = {
                    x: a.attr.x - a.margin,
                    y: a.attr.y - a.margin,
                    width: a.attr.width + a.margin * 2,
                    height: a.attr.height + a.margin * 2
                };
                var x, v;
                t.drag(function (r, o) {
```

```
    A(r, o)
},
function () {
    z()
},
function () {
    l()
});
f.drag(function (r, o) {
    A(r, o)
},
function () {
    z()
},
function () {
    l()
});
n.drag(function (r, o) {
    A(r, o)
},
function () {
    z()
},
function () {
    l()
});
var A = function (dx, dy) {
    var o = (x + dx);
    var G = (v + dy);
    q.x = o - a.margin;
    q.y = G - a.margin;
    B();
};
var z = function () {
    x = t.attr("x");
    v = t.attr("y");
    t.attr({
        opacity: 0.5
    });
    f.attr({
        opacity: 0.5
    });
};
var l = function () {
```

```
    t.attr({
        opacity: 1
    });
    f.attr({
        opacity: 1
    });
};

$([t.node, f.node]).bind("click", function (e) {
    if ($(pager).data("currNode") != u) {
        t.attr("fill", "90-#fff-#0b92d5");
        if (flow.begin) {
            if (flow.begin != u) {
                if (flow.end) {
                    if (flow.end != u) {
                        n.show();
                        n.attr("text", "[后继]");
                        flow.tmp = flow.end;
                        flow.end = u;
                    }
                }
                else {
                    n.show();
                    n.attr("text", "[后继]");
                    flow.end = u;
                }
            }
            else {
                if (flow.end) {
                    n.show();
                    n.attr("text", "[后继]");
                    flow.tmp = flow.end;
                    flow.end = u;
                }
            }
        }
        else {
            n.show();
            n.attr("text", "[前置]");
            flow.begin = u;
        }
        $(pager).trigger("click", u);
        $(pager).data("currNode", u);
        flow.util.getRectPropertie(flow.rectarr[g]);
```

```
    }
});

var j = function (o, r) {
    if (r.getId() != g) {
        t.attr("fill", "90-#fff-#C0C0C0");
        if (r.getId().substring(0, 4) == "rect") {
            if (flow.begin == u) {
                //终点非当前选中节点
                if (flow.tmp && flow.tmp.getId() != r.getId()) {
                    n.hide();
                    n.attr("text", '');
                }
            }
            else if (flow.tmp == u) { //终点改为起点
                n.show();
                n.attr("text", "[前置]");
                flow.begin = u;
            }
            else {
                n.hide();
                n.attr("text", '');
            }
        }
    }
};
$(pager).bind("click", j);

$([t.node, f.node]).bind("dblclick", function () {
    //alert(t['rectType'])
});

function B() {
    var F = q.x + a.margin,
    r = q.y + a.margin,
    G = q.width - a.margin * 2,
    o = q.height - a.margin * 2;
    t.attr({
        x: F,
        y: r,
        width: G,
        height: o
    });
    f.attr({
```

```
                    x: F + G / 2,
                    y: r + o / 2
                });
                n.attr({
                    x: F + 120,
                    y: r + 8
                });
                $(pager).trigger("rectresize", u)
            }
            this.toJson = function () {
                var seq = flow.util.groupSeq(u);
                var r = g + "," + Math.round(t.attr("x")) + "," +
Math.round(t.attr("y")) + "," + t["id"] + "," + t["name"]
                + "," + t["rectType"] + "," + t["formId"] + "," + t["virtual"]
+ "," + t["cooperation"]
                + "," + t["departmentId"] + "," + t["positionId"] + "," +
t["userId"] + "," + t["remark"] + "," + seq + "," + t["isApproval"];
                //                        var r = "{TmpID:'" + g + "',X:" +
Math.round(t.attr("x")) + ",Y:" + Math.round(t.attr("y")) + ",NodeID:"
                //                    + t["id"] + ",NodeName:'" + t["name"] +
"',NodeType:" + t["recttype"] + ",MainMenu:'" + t["mainmenu"]
                //                        + "',CopyMenu:'" + t["copymenu"] +
"',Virtual:'" + t["virtual"] + "',Cooperation:'" + t["cooperation"]
                //                    + "',Dept:'" + t["dept"] + "',Role:'" +
t["role"] + "',Post:'" + t["position"] + "',User:'" + t["userid"]
                //                      + "',Description:'" + t["remark"] +
"'";
                //                    r += "}";
                return r;
            };
            this.getBBox = function () {
                return q;
            };
            this.getId = function () {
                return g;
            };
            this.text = function () {
                return f.attr("text");
            };
            this.settext = function (text) {
                f.attr("text", text);
            };
            this.attr = function (o) {
                if (o) {
```

```
            return t[o];
        }
    };
    this.setattr = function (o, v) {
        if (o) {
            t[o] = v;
        }
    };
    this.remove = function () {
        t.remove();
        f.remove();
        n.remove();
    };
    this.select = function () {
        $([t.node, f.node]).trigger('click');
    };
};
flow.path = function (begin, end, path) {
    var v = this;
    var i,
    t,
    f,
    y,
    w,
    x;
    var a;
    if (path) {
        a = $.extend(true, {}, flow.config.path, path);
    }
    else {
        a = flow.config.path;
    }
    var h = a.textPos;
    var g = "path" + flow.util.nextPathId();
    //绘制连线上的点
    function p(G, H, D, L) {
        var F = this,
        M = G,
        r, o = D,
        O = L,
        K, I, N = H;
        switch (M) {
            case "from":
                r = pager.rect(H.x - a.attr.fromDot.width / 2, H.y -
```

```
a.attr.fromDot.height / 2, a.attr.fromDot.width,
a.attr.fromDot.height).attr(a.attr.fromDot);
                            break;
                        case "big":
                            r = pager.rect(H.x - a.attr.bigDot.width / 2, H.y -
a.attr.bigDot.height / 2, a.attr.bigDot.width,
a.attr.bigDot.height).attr(a.attr.bigDot);
                            break;
                        case "small":
                            r = pager.rect(H.x - a.attr.smallDot.width / 2, H.y -
a.attr.smallDot.height / 2, a.attr.smallDot.width,
a.attr.smallDot.height).attr(a.attr.smallDot);
                            break;
                        case "to":
                            r = pager.rect(H.x - a.attr.toDot.width / 2, H.y -
a.attr.toDot.height / 2, a.attr.toDot.width,
a.attr.toDot.height).attr(a.attr.toDot);
                            break
                    }
                    if (r && (M == "big" || M == "small")) {
                        r.drag(function (Q, P) { //拖动处理函数
                            C(Q, P)
                        },
                        function () { //拖动开始的处理函数
                            J()
                        },
                        function () { //拖动结束的处理函数
                            E()
                        });
                        var C = function (R, Q) {
                            var P = (K + R), S = (I + Q);
                            F.moveTo(P, S)
                        };
                        var J = function () {
                            if (M == "big") {
                                K = r.attr("x") + a.attr.bigDot.width / 2;
                                I = r.attr("y") + a.attr.bigDot.height / 2
                            }
                            if (M == "small") {
                                K = r.attr("x") + a.attr.smallDot.width / 2;
                                I = r.attr("y") + a.attr.smallDot.height / 2
                            }
                        };
                        var E = function () { }
```

```
}
this.type = function (P) {
    if (P) {
        M = P
    } else {
        return M
    }
};
this.node = function (P) {
    if (P) {
        r = P
    } else {
        return r
    }
};
this.left = function (P) {
    if (P) {
        o = P
    } else {
        return o
    }
};
this.right = function (P) {
    if (P) {
        O = P
    } else {
        return O
    }
};
this.remove = function () {
    o = null;
    O = null;
    r.remove()
};
this.pos = function (P) {
    if (P) {
        N = P;
        r.attr({
            x: N.x - r.attr("width") / 2,
            y: N.y - r.attr("height") / 2
        });
        return this
    } else {
        return N
```

```
                        }
                    };
                    this.moveTo = function (Q, T) {
                        this.pos({
                            x: Q,
                            y: T
                        });
                        switch (M) {
                            case "from":
                                if (O && O.right() && O.right().type() == "to") {
O.right().pos(flow.util.connPoint(end.getBBox(), N))
                                }
                                if (O && O.right()) {
                                    O.pos(flow.util.center(N, O.right().pos()))
                                }
                                break;
                            case "big":
                                if (O && O.right() && O.right().type() == "to") {
O.right().pos(flow.util.connPoint(end.getBBox(), N))
                                }
                                if (o && o.left() && o.left().type() == "from") {
o.left().pos(flow.util.connPoint(begin.getBBox(), N))
                                }
                                if (O && O.right()) {
                                    O.pos(flow.util.center(N, O.right().pos()))
                                }
                                if (o && o.left()) {
                                    o.pos(flow.util.center(N, o.left().pos()))
                                }
                                var S = {
                                    x: N.x,
                                    y: N.y
                                };
                                if     (flow.util.isLine(o.left().pos(),     S,
O.right().pos())) {
                                    M = "small";
                                    r.attr(a.attr.smallDot);
                                    this.pos(S);
                                    var P = o;
                                    o.left().right(o.right());
                                    o = o.left();
```

```
                                P.remove();
                                var R = O;
                                O.right().left(O.left());
                                O = O.right();
                                R.remove()
                            }
                            break;
                        case "small":
                            if (o && O && !flow.util.isLine(o.pos(), {
                                x: N.x,
                                y: N.y
                            }, O.pos())) {
                                M = "big";
                                r.attr(a.attr.bigDot);
                                var P = new p("small",
flow.util.center(o.pos(), N), o, o.right());
                                o.right(P);
                                o = P;
                                var R = new p("small",
flow.util.center(O.pos(), N), O.left(), O);
                                O.left(R);
                                O = R
                            }
                            break;
                        case "to":
                            if (o && o.left() && o.left().type() == "from") {

o.left().pos(flow.util.connPoint(begin.getBBox(), N))
                            }
                            if (o && o.left()) {
                                o.pos(flow.util.center(N, o.left().pos()))
                            }
                            break
                    }
                    m()
                }
            }
            function j() {
                var D, C, E = begin.getBBox(), //起点属性
                F = end.getBBox(), //终点属性
                r,
                o;
                r = flow.util.connPoint(E, {
                    x: F.x + F.width / 2,
```

```
                    y: F.y + F.height / 2
                });
                o = flow.util.connPoint(F, r);
                D = new p("from", r, null, new p("small", {
                    x: (r.x + o.x) / 2,
                    y: (r.y + o.y) / 2
                }));
                D.right().left(D);
                C = new p("to", o, D.right(), null);
                D.right().right(C);
                this.toPathString = function () {
                    if (!D) {
                        return ""
                    }
                    var J = D,
                    I = "M" + J.pos().x + " " + J.pos().y,
                    H = "";
                    while (J.right()) {
                        J = J.right();
                        I += "L" + J.pos().x + " " + J.pos().y
                    }
                    var G = flow.util.arrow(J.left().pos(), J.pos(),
a.attr.arrow.radius);
                    H = "M" + G[0].x + " " + G[0].y + "L" + G[1].x + " " + G[1].y
+ "L" + G[2].x + " " + G[2].y + "z";
                    return [I, H]
                };
                this.toJson = function () {
                    var G = "[", H = D;
                    while (H) {
                        if (H.type() == "big") {
                            G += "{x:" + Math.round(H.pos().x) + ",y:" +
Math.round(H.pos().y) + "},"
                        }
                        H = H.right()
                    }
                    if (G.substring(G.length - 1, G.length) == ",") {
                        G = G.substring(0, G.length - 1)
                    }
                    G += "]";
                    return G
                };
                this.restore = function (H) {
                    var I = H, J = D.right();
```

```
    for (var G = 0; G < I.length; G++) {
        J.moveTo(I[G].x, I[G].y);
        J.moveTo(I[G].x, I[G].y);
        J = J.right()
    }
    this.hide()
};
this.fromDot = function () {
    return D
};
this.toDot = function () {
    return C
};
this.midDot = function () {
    var H = D.right(), G = D.right().right();
    while (G.right() && G.right().right()) {
        G = G.right().right();
        H = H.right()
    }
    return H
};
this.show = function () {
    var G = D;
    while (G) {
        G.node().show();
        G = G.right()
    }
};
this.hide = function () {
    var G = D;
    while (G) {
        G.node().hide();
        G = G.right()
    }
};
this.remove = function () {
    var G = D;
    while (G) {
        if (G.right()) {
            G = G.right();
            G.left().remove()
        } else {
            G.remove();
            G = null
```

```
            }
        }
    }
}
i = pager.path(a.attr.path.path).attr(a.attr.path);
flow.util.attr(i, a.data);
t = pager.path(a.attr.arrow.path).attr(a.attr.arrow);
x = new j();
x.hide();
f = pager.text(0, 0, a.text.text).attr(a.text);
f.drag(function (r, o) {
    if (!flow.config.editable) {
        return
    }
    f.attr({
        x: y + r,
        y: w + o
    })
},
function () {
    y = f.attr("x");
    w = f.attr("y")
},
function () {
    var o = x.midDot().pos();
    h = {
        x: f.attr("x") - o.x,
        y: f.attr("y") - o.y
    }
});
m();
//连线点击事件
$([i.node, t.node, f.node]).bind("click", function () {
    $(pager).trigger("click", v);
    $(pager).data("currNode", v);

    flow.util.getPathPropertie(flow.patharr[g]);
    return false
});
//pager 点击事件
var l = function (r, C) {
    if (C && C.getId() == g) {
        x.show();
    } else {
```

```
                x.hide()
            }
        };
        $(pager).bind("click", l);
        //双击事件
        $([i.node, t.node, f.node]).bind("dblclick", function () {
            //flow.util.showPathAttr(i, f);
        });
        //删除节点事件（每条连线都会触发）
        var A = function (o, r) {
            if (!flow.config.editable) {
                return
            }
            if (r && (r.getId() == begin.getId() || r.getId() ==
end.getId())) {
                $(pager).trigger("removepath", v)
            }
        };
        $(pager).bind("removerect", A);
        var d = function (C, D) {
            if (begin && begin.getId() == D.getId()) {
                var o;
                if (x.fromDot().right().right().type() == "to") {
                    o = {
                        x: end.getBBox().x + end.getBBox().width / 2,
                        y: end.getBBox().y + end.getBBox().height / 2
                    }
                } else {
                    o = x.fromDot().right().right().pos()
                }
                var r = flow.util.connPoint(begin.getBBox(), o);
                x.fromDot().moveTo(r.x, r.y);
                m();
            }
            if (end && end.getId() == D.getId()) {
                var o;
                if (x.toDot().left().left().type() == "from") {
                    o = {
                        x: begin.getBBox().x + begin.getBBox().width / 2,
                        y: begin.getBBox().y + begin.getBBox().height / 2
                    }
                } else {
                    o = x.toDot().left().left().pos()
                }
```

```
                    var r = flow.util.connPoint(end.getBBox(), o);
                    x.toDot().moveTo(r.x, r.y);
                    m();
                }
            };
            $(pager).bind("rectresize", d);
            this.from = function () {
                return begin
            };
            this.to = function () {
                return end
            };
            this.toJson = function () {
                //                      var r = "{From:'" + begin.getId() +
"',To:'" + end.getId() + "',LinkName:'" + f.attr("text") + "',X:" + Math.round(h.x)
+ ",Y:" + Math.round(h.y) + ",Operator:'"
                //                                  + i["operatortext"] +
"',OperatorValue:'" + i["condition"] + "',Description:'" + i["remark"] + "'";
                //                      r += "}";
                var r = begin.getId() + "," + end.getId() + "," + i["name"] +
"," + Math.round(h.x) + "," + Math.round(h.y) + ","
                    + i["formId"] + "," + i["field"] + "," + i["operator"] +
"," + i["operatorValue"] + "," + i["remark"];
                return r;
            };
            this.restore = function (o) {
                var r = o;
                a = $.extend(true, a, o);
                x.restore(r.dots)
            };
            this.remove = function () {
                x.remove();
                i.remove();
                t.remove();
                f.remove();
                try {
                    $(pager).unbind("click", l)
                } catch (o) { }
                try {
                    $(pager).unbind("removerect", A)
                } catch (o) { }
                try {
                    $(pager).unbind("rectresize", d)
                } catch (o) { }
```

```
    };
    function m() {
        var r = x.toPathString(), o = x.midDot().pos();
        i.attr({
            path: r[0]
        });
        t.attr({
            path: r[1]
        });
        f.attr({
            x: o.x + h.x,
            y: o.y + h.y
        })
    }
    this.getId = function () {
        return g
    };
    this.text = function () {
        return f.attr("text")
    };
    this.attr = function (o) {
        if (o && o.path) {
            return i[o.path]
        }
        if (o && o.arrow) {
            return t[o.arrow]
        }
    }
    this.settext = function (text) {
        f.attr("text", text);
    };
    this.setattr = function (o, v) {
        if (o) {
            i[o] = v;
        }
    };
    this.select = function () {
        $([i.node, t.node, f.node]).trigger('click');
    };
};
flow.init = function (d) {
    //Delete 按键删除事件
    $(document).keydown(function (i) {
        if (i.keyCode == 46) {
```

```
                var j = $(pager).data("currNode");
                if (j) {
                    if (j.getId().substring(0, 4) == "rect") {
                        $(pager).trigger("removerect", j)
                    } else {
                        if (j.getId().substring(0, 4) == "path") {
                            $(pager).trigger("removepath", j)
                        }
                    }
                    $(pager).removeData("currNode");
                }
            }
        });
        //删除事件
        var w = function (c, i) {
            if (i.getId().substring(0, 4) == "rect") {
                flow.rectarr[i.getId()] = null;
                i.remove();
            } else {
                if (i.getId().substring(0, 4) == "path") {
                    flow.patharr[i.getId()] = null;
                    i.remove();
                }
            }
        };
        $(pager).bind("removepath", w);
        $(pager).bind("removerect", w);
        //初始化
        var z = {};
        if (d) {
            if (d.data.rects) {
                for (var s in d.data.rects) {
                    var r = new flow.rect(d.data.rects[s]);
                    z[d.data.rects[s].data.id] = r;
                    flow.rectarr[r.getId()] = r;
                }
            }
            if (d.data.paths) {
                for (var s in d.data.paths) {
                    var n = new flow.path(z[d.data.paths[s].from],
z[d.data.paths[s].to], d.data.paths[s]);
                    flow.patharr[n.getId()] = n;
                }
            }
```

```
        }

        //获取表单数据
        $.ajax({
            url: "/FormDesigner/FormDesign/GetFormList",     //后台数据
请求地址
            type: "get",
            data: { page: 1, limit: 100 },
            async: false,
            success: function (slt) {
                if (slt) {
                    var data = JSON.parse(slt);
                    flow.forms = data.data;
                }
                else {
                    //layer.msg(slt.message || '操作失败，请重试。');
                }
            }
        });

        //获取职位数据
        $.ajax({
            url: "/System/Position/GetPositionList",     //后台数据请求
地址
            type: "get",
            async: false,
            success: function (slt) {
                if (slt) {
                    var data = JSON.parse(slt);
                    flow.positions = data.data;
                }
                else {
                    //layer.msg(slt.message || '操作失败，请重试。');
                }
            }
        });

        //获取用户数据
        $.ajax({
            url: "/System/User/GetUserList",     //后台数据请求地址
            type: "get",
            data: { page: 1, limit: 500 },
            async: false,
            success: function (slt) {
```

```
                        if (slt) {
                            var data = JSON.parse(slt);
                            flow.users = data.data;
                        }
                        else {
                            //layer.msg(slt.message || '操作失败，请重试。');
                        }
                    }
                });
            };
            flow.resetForm = function (formName) {
                $.ajax({
                    url: "/FormDesigner/FormDesign/GetFormList",     //后台数据
请求地址
                    type: "get",
                    data: { page: 1, limit: 100 },
                    async: false,
                    success: function (slt) {
                        if (slt) {
                            var data = JSON.parse(slt);
                            flow.forms = data.data;
                            var pid = $("#pid").val();
                            for (let index = 0; index < flow.forms.length; index++)
{

                                const element = flow.forms[index];
                                if (element.FormName == formName) {
                                    flow.rectarr[pid].setattr('formId',
element.Id);

                                }
                            }
                            //加载元素属性
                            flow.util.getRectPropertie(flow.rectarr[pid]);
                        }
                        else {
                            //layer.msg(slt.message || '操作失败，请重试。');
                        }
                    }
                });
                $("")
            }
            return flow;
        }
    };
    $.Flow = Flow;
```

```
})(jQuery);
```

程序结束!!!

程序开始!!!

```
using LongQin.Attributes;
using LongQin.Common;
using LongQin.Configs;
using LongQin.Models;
using LongQin.Service;
using LongQin.Service.Base;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Web;
using System.Web.Mvc;
using System.Web.Script.Serialization;

namespace LongQin.Areas.System.Controllers
{
    [CheckLogin]
    public class WorkFlowController : ControllerBase
    {
        IWorkFlowService _workFlowService = AutofacService.Resolve<IWorkFlowService>();

        public ActionResult Start(int flowId = 0, string flowName = "")
        {
            ViewBag.flowId = flowId;
            ViewBag.flowName = flowName;
            return View();
        }

        [HttpGet]
        public string GetFlowBeginNodeForm(int flowId)
        {
            FormDesigner form = _workFlowService.GetFlowBeignNodeFormAsync(flowId);
            JavaScriptSerializer serializer = new JavaScriptSerializer();
            return serializer.Serialize(form);
        }
```

```
        [HttpPost]
        [Operation("流程处理")]
        [ValidateInput(false)]
        public JsonResult Deal(FormCollection collection)
        {
            int flowId = collection[0].ToInt32();
            int processId = collection[1].ToInt32();
            string tableName = collection[2];
            int isApproval = collection[3].ToInt32();
            bool isSave = collection[4].ToInt32() == 1 ? true : false; //是否暂存
            List<string> columns = new List<string>();
            List<string> values = new List<string>();
            for (int i = 5; i < collection.Count; i++)
            {
                string key = collection.AllKeys[i];
                if (key == "file") continue; //文件上传有两个input，屏蔽一个
                columns.Add(key);
                string val = collection[i];
                values.Add(val);
            }
            int direction = isApproval == 1 && collection[5] == "0" ? 2 : 1; // 前进-1
还是后退-2
            var result = new ResultBase();
            int data = _workFlowService.DealWorkAsync(flowId, processId, direction,
tableName, columns, values, LoginUser.UserId, LoginUser.OrganizationId, isSave);
            result.success = data > 0 ? true : false;
            result.data = data;
            return Json(result);
        }

        // 待办工作列表
        public ActionResult Backlog()
        {
            return View();
        }

        public string GetProcessList(string beginDate, string endDate, int status, int page,
int limit)
        {
            int userId = LoginUser.UserId;
            var list = _workFlowService.GetProcessListAsync(userId, beginDate, endDate,
status, page, limit);
            JavaScriptSerializer serializer = new JavaScriptSerializer();
```

```
            string str = serializer.Serialize(new
            {
                code = 0,
                msg = "",
                count = list.Total,
                data = list.Data
            });

            str = Regex.Replace(str, @"\\/Date\((\d+)\)\\/", match =>
            {

                DateTime dt = new DateTime(1970, 1, 1);
                dt = dt.AddMilliseconds(long.Parse(match.Groups[1].Value));
                dt = dt.ToLocalTime();
                return dt.ToString("yyyy-MM-dd HH:mm:ss");
            });
            return str;
        }

        public ActionResult DealWork(int workId, int processId, int flowId)
        {
            ViewBag.workId = workId;
            ViewBag.processId = processId;
            ViewBag.flowId = flowId;
            return View();
        }

        [HttpGet]
        public string GetFlowProcessForm(int processId)
        {
            FormDesigner form = _workFlowService.GetFlowProcessFormAsync(processId);
            if (form == null) return "";
            Object formData = _workFlowService.GetFlowProcessFormDataAsync(processId,
form.TableName);
            Dictionary<string, object> dic = new Dictionary<string, object>();
            dic.Add("form", form);
            dic.Add("formData", formData);
            JavaScriptSerializer serializer = new JavaScriptSerializer();
            return serializer.Serialize(dic);
        }

        [HttpGet]
        public string GetFlowProcessFormData(int processId, string tableName)
        {
```

```
            Object formData = _workFlowService.GetFlowProcessFormDataAsync(processId,
tableName);
            if (formData == null) return "";
            JavaScriptSerializer serializer = new JavaScriptSerializer();
            return serializer.Serialize(formData);
        }

        [HttpGet]
        public string GetWorkProcessFormListAsync(int workId)
        {
            List<Dictionary<string, object>> list =
_workFlowService.GetWorkProcessFormListAsync(workId);
            JavaScriptSerializer serializer = new JavaScriptSerializer();
            return serializer.Serialize(list);
        }

        [HttpPost]
        [Operation("工作转办")]
        public JsonResult WorkTranfer(int processId, string transferUser)
        {
            var result = new ResultBase();
            int data = _workFlowService.WorkTransferAsync(processId, transferUser,
LoginUser.UserId, LoginUser.OrganizationId);
            result.success = data > 0 ? true : false;
            result.data = data;
            return Json(result);
        }

        // 已办工作列表
        public ActionResult Completed()
        {
            return View();
        }

        // 工作明细
        public ActionResult Details(int workId, int processId)
        {
            ViewBag.workId = workId;
            ViewBag.processId = processId;
            return View();
        }

        public string GetWorkSteps(int workId, int page, int limit)
        {
```

```
            var list = _workFlowService.GetWorkStepsAsync(workId, page, limit);
            JavaScriptSerializer serializer = new JavaScriptSerializer();
            string str = serializer.Serialize(new
            {
                code = 0,
                msg = "",
                count = list.Total,
                data = list.Data
            });

            str = Regex.Replace(str, @"\\/Date\((\d+)\)\\/", match =>
            {

                DateTime dt = new DateTime(1970, 1, 1);
                dt = dt.AddMilliseconds(long.Parse(match.Groups[1].Value));
                dt = dt.ToLocalTime();
                return dt.ToString("yyyy-MM-dd HH:mm:ss");
            });
            return str;
        }

        [HttpPost]
        [Operation("上传文件")]
        public JsonResult UploadFile(string processId)
        {
            string fileName = "";
            var files = Request.Files;
            foreach (var key in files.AllKeys)
            {
                var file = Request.Files[key];
                string uploadResult = UploadHelper.Process(file.FileName,
file.InputStream);
                if (!string.IsNullOrEmpty(uploadResult))
                {
                    fileName = uploadResult;
                }
            }
            var result = new ResultBase();
            result.success = String.IsNullOrEmpty(fileName) ? false : true;
            result.data = fileName;
            return Json(result);
        }
    }
}
```

程序结束!!!

程序开始!!!

```
using LongQin.Repository;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LongQin.Models;
using LongQin.Common;
using System.Data;
using System.Transactions;
using System.Diagnostics;

namespace LongQin.Service
{
    public class WorkFlowService : ServiceBase, IWorkFlowService
    {
        IWorkFlowRepository                 _workFlowRepository                 =
AutofacRepository.Resolve<IWorkFlowRepository>();
        IUserRepository                     _userRepository                     =
AutofacRepository.Resolve<IUserRepository>();
        IDepartmentRepository               _departmentRepository               =
AutofacRepository.Resolve<IDepartmentRepository>();
        IPositionRepository                 _positionRepository                 =
AutofacRepository.Resolve<IPositionRepository>();
        IFormDesignerRepository             _formDesignerRepository             =
AutofacRepository.Resolve<IFormDesignerRepository>();

        public WorkFlowService()
        {
            base.AddDisposableObject(_workFlowRepository);
        }

        public FormDesigner GetFlowBeignNodeFormAsync(int flowId)
        {
            if (flowId <= 0)
            {
                throw new ArgumentException("id 错误");
            }
```

```
            return _workFlowRepository.GetFlowBeignNodeFormAsync(flowId);
        }

        // 处理用户提交的工作
        public int DealWorkAsync(int flowId, int processId, int action, string
tableName,
            List<string> columns, List<string> values, int submitter, int
organizationId, bool isSave)
        {
            using (TransactionScope scope = new TransactionScope()) // 开启事务
            {
                // 先处理表单数据，增加创建人、创建时间、组织机构、表单状态
                columns.Add("creator");
                values.Add(submitter.ToString());
                columns.Add("createTime");
                string format = "yyyy-MM-dd HH:mm:ss";
                values.Add(DateTime.Now.ToString(format));
                columns.Add("organizationId");
                values.Add(organizationId.ToString());
                columns.Add("status");
                values.Add("1");

                bool isSucceed = false;
                int result = 1;
                if (processId == 0)
                {
                    // 工作还没创建
                    // 先创建工作实例以及生成一条代办
                    FlowNode                        node                        =
_workFlowRepository.GetFlowBeignNodeAsync(flowId);
                    int nodeId = node.NodeId;
                    FlowWork work = new FlowWork();
                    work.FlowId = flowId;
                    work.Creator = submitter;
                    work.OrganizationId = organizationId;
                    int workId = _workFlowRepository.CreateFlowWorkAsync(work);
                    if (workId == 0) return -1;
                    FlowProcess process = new FlowProcess();
                    process.WorkId = workId;
                    process.NodeId = node.NodeId;
                    process.LinkId = 0;
                    process.SendingTo = submitter;
                    process.ProcessType = 1;
                    process.Submitter = submitter;
```

```
                    process.Status = 1;
                    process.OrganizationId = organizationId;
                    processId                                                    =
_workFlowRepository.CreateFlowProcessAsync(process);
                    if (processId == 0) return -1;
                    // 插入表单数据
                    isSucceed = DealFormData(workId, processId, nodeId, tableName,
columns, values, 0);
                    if (!isSucceed) return -1;
                    if (!isSave)
                    {
                        // 流程流转
                        result = ExcuteFlowAsync(flowId, workId, processId,
nodeId, action, columns, values, submitter, organizationId);
                    }
                    else
                    {
                        // 返回代办工作 ID
                        result = processId;
                    }
                }
                else
                {
                    FlowProcess                  flowProcess                  =
_workFlowRepository.GetProcessByIdAsync(processId);
                    // 插入表单数据
                    isSucceed = DealFormData(flowProcess.WorkId, processId,
flowProcess.NodeId, tableName, columns, values, flowProcess.FormDataId);
                    if (!isSucceed) return -1;
                    if (!isSave)
                    {
                        // 流程流转
                        result      =      ExcuteFlowAsync(flowProcess.FlowId,
flowProcess.WorkId, processId, flowProcess.NodeId, action, columns, values,
submitter, organizationId);
                    }
                }
                if (result > 0)
                {
                    scope.Complete();
                }
                return result;
            }
        }
```

```
        // 流程流转
        // 返回值：1-成功，-1 失败，-2 没找到处理人
        public int ExcuteFlowAsync(int flowId, int workId, int processId, int nodeId, int action, List<string> columns, List<string> values, int submitter, int organizationId)
        {
            bool isSucceed = false;
            // 插入操作步骤
            FlowStep step = new FlowStep();
            step.WorkId = workId;
            step.NodeId = nodeId;
            step.ProcessId = processId;
            step.Submitter = submitter;
            step.OrganizationId = organizationId;
            step.Action = action;
            isSucceed = _workFlowRepository.CreateFlowStepAsync(step);
            if (!isSucceed) return -1;
            if (action == 1) // 前进
            {
                // 获取当前节点
                FlowNode fromNode = _workFlowRepository.GetFlowNodeByIdAsync(nodeId);
                bool needCooperation = NeedCooperation(workId, fromNode);
                if (!needCooperation)
                {
                    // 无需多人协作，关闭当前节点所有待办
                    isSucceed = _workFlowRepository.CloseNodeProcessAsync(workId, nodeId);
                    if (!isSucceed) return -1;
                    // 获取节点连线
                    List<FlowLink> links = _workFlowRepository.GetFlowNodeLinksAsync(nodeId);
                    if (links.Count == 0)
                    {
                        // 没有后继节点，结束流程
                        isSucceed = _workFlowRepository.CloseWorkAsync(workId);
                        if (!isSucceed) return -1;
                        else return 1;
                    }
                    // 创建下个节点待办
                    // 判断是否是普通节点或者合流点（只有普通节点或合流点存在条件走向，或后继节点可能是合流点）
                    if (fromNode.NodeType == 0 || fromNode.NodeType == 2)
```

```
                    {
                        // 判断是否有条件
                        int toNodeId = 0;
                        int linkId = 0;
                        foreach (FlowLink link in links)
                        {
                            string field = link.Field;
                            string operatorName = link.Operator;
                            string operatorValue = link.OperatorValue;
                            if                    (!String.IsNullOrEmpty(field)
&& !String.IsNullOrEmpty(operatorName) && !String.IsNullOrEmpty(operatorValue))
                            {
                                int submitterCondition = submitter;
                                // 判断条件表单，不为0表示取其他节点表单
                                if (link.FormId != 0)
                                {
                                    FormDesigner         formDesigner         =
_formDesignerRepository.GetByIdAsync(link.FormId);
                                    columns                                   =
_workFlowRepository.GetTableColumnsAsync(formDesigner.TableName);
                                    values                                    =
_workFlowRepository.GetTableValueAsync(workId, formDesigner.TableName);
                                }
                                // 有条件，判断满足哪个条件
                                if (field == "userId")
                                {
                                    // 提交人
                                    switch (operatorName)
                                    {
                                        case "=":
                                            if         (operatorValue        ==
submitter.ToString())
                                            {
                                                toNodeId = link.ToNodeId;
                                                linkId = link.LinkId;
                                            }
                                            break;
                                        case "!=":
                                            if         (operatorValue        !=
submitter.ToString())
                                            {
                                                toNodeId = link.ToNodeId;
                                                linkId = link.LinkId;
                                            }
```

```
                                    break;
                                }
                            }
                            else if (field == "positionLevel")
                            {
                                // 提交人职级
                                // 获取提交人最高职级
                                int             positionLevel             =
_userRepository.GetUserPositionLevelAsync(submitter);
                                int val = operatorValue.ToInt32();
                                switch (operatorName)
                                {
                                    case "=":
                                        if (val == positionLevel)
                                        {
                                            toNodeId = link.ToNodeId;
                                            linkId = link.LinkId;
                                        }
                                        break;
                                    case "!=":
                                        if (val != positionLevel)
                                        {
                                            toNodeId = link.ToNodeId;
                                            linkId = link.LinkId;
                                        }
                                        break;
                                    case ">":
                                        if (positionLevel > val)
                                        {
                                            toNodeId = link.ToNodeId;
                                            linkId = link.LinkId;
                                        }
                                        break;
                                    case "<":
                                        if (positionLevel < val)
                                        {
                                            toNodeId = link.ToNodeId;
                                            linkId = link.LinkId;
                                        }
                                        break;
                                    case ">=":
                                        if (positionLevel >= val)
                                        {
                                            toNodeId = link.ToNodeId;
```

```
                                        linkId = link.LinkId;
                                    }
                                    break;
                                case "<=":
                                    if (positionLevel <= val)
                                    {
                                        toNodeId = link.ToNodeId;
                                        linkId = link.LinkId;
                                    }
                                    break;
                            }
                        }
                        else
                        {
                            if (columns == null || values == null ||
columns.Count != values.Count) continue;
                            for (int i = 0; i < columns.Count; i++)
                            {
                                string column = columns[i];
                                string value = values[i];
                                if (column == field)
                                {
                                    switch (operatorName)
                                    {
                                        case "=":
                                            if (operatorValue == value)
                                            {
                                                toNodeId              =
link.ToNodeId;
                                                linkId = link.LinkId;
                                            }
                                            break;
                                        case "!=":
                                            if (operatorValue != value)
                                            {
                                                toNodeId              =
link.ToNodeId;
                                                linkId = link.LinkId;
                                            }
                                            break;
                                        case ">":
                                            int     intFormValue     =
value.ToInt32();
                                            int    intOperatorValue    =
```

```
operatorValue.ToInt32();
                    if      (intFormValue      >
intOperatorValue)
                    {
                        toNodeId              =
link.ToNodeId;
                        linkId = link.LinkId;
                    }
                    break;
                case "<":
                    int     intFormValue1     =
value.ToInt32();
                    int   intOperatorValue1   =
operatorValue.ToInt32();
                    if     (intFormValue1     <
intOperatorValue1)
                    {
                        toNodeId              =
link.ToNodeId;
                        linkId = link.LinkId;
                    }
                    break;
                case ">=":
                    int     intFormValue2     =
value.ToInt32();
                    int   intOperatorValue2   =
operatorValue.ToInt32();
if (intFormValue2 >= intOperatorValue2)
                    {
                        toNodeId              =
link.ToNodeId;
                        linkId = link.LinkId;
                    }
                    break;
                case "<=":
                    int     intFormValue3     =
value.ToInt32();
                    int   intOperatorValue3   =
operatorValue.ToInt32();
                    if     (intFormValue3     <=
intOperatorValue3)
                    {
                        toNodeId              =
link.ToNodeId;
```

```
                                        linkId = link.LinkId;
                                    }
                                    break;
                                }
                            }
                        }
                    }
                }
                if (toNodeId == 0)
                {
                    // 没有条件，则默认取第一个线路
                    FlowLink link = links[0];
                    FlowNode toNode =
_workFlowRepository.GetFlowNodeByIdAsync(link.ToNodeId);
                    if (toNode.NodeType == 2 || toNode.NodeType == 3)
                    {
                        // 后继节点是合流点或者分合流点，则需判断前置节
点待办是否都已处理
                        int preNodeProcess =
_workFlowRepository.GetPreNodeProcessCountAsync(workId, link.ToNodeId);
                        if (preNodeProcess == 0)
                        {
                            // 前驱节点都已关闭，创建后继节点的待办，否
则不做处理
                            List<int> handlers =
GetHandlerAsync(link.ToNodeId, submitter);
                            if (handlers.Count == 0)
                            {
                                return -2;
                            }
                            foreach (int handler in handlers)
                            {
                                // 创建代办工作
                                FlowProcess process = new FlowProcess();
                                process.WorkId = workId;
                                process.NodeId = link.ToNodeId;
                                process.LinkId = link.LinkId;
                                process.SendingTo = handler;
                                process.ProcessType = 1;
                                process.Submitter = submitter;
                                process.OrganizationId = organizationId;
                                process.Status = 1;
                                int newProcessId =
```

```
_workFlowRepository.CreateFlowProcessAsync(process);
                                    if (newProcessId <= 0) return -1;
                                }
                            }
                        }
                        else
                        {
                            // 后继节点是普通或者分流点，则直接生成待办工作
                            List<int>              handlers              =
GetHandlerAsync(link.ToNodeId, submitter);
                            if (handlers.Count == 0)
                            {
                                return -2;
                            }
                            foreach (int handler in handlers)
                            {
                                // 创建代办工作
                                FlowProcess process = new FlowProcess();
                                process.WorkId = workId;
                                process.NodeId = link.ToNodeId;
                                process.LinkId = link.LinkId;
                                process.SendingTo = handler;
                                process.ProcessType = 1;
                                process.Submitter = submitter;
                                process.OrganizationId = organizationId;
                                process.Status = 1;
                                int             newProcessId             =
_workFlowRepository.CreateFlowProcessAsync(process);
                                if (newProcessId <= 0) return -1;
                            }
                        }
                    }
                    else
                    {
                        // 没有条件，则直接生成待办工作
                        List<int>  handlers  =  GetHandlerAsync(toNodeId,
submitter);
                        if (handlers.Count == 0)
                        {
                            return -2;
                        }
                        foreach (int handler in handlers)
                        {
                            // 创建代办工作
```

```
                                FlowProcess process = new FlowProcess();
                                process.WorkId = workId;
                                process.NodeId = toNodeId;
                                process.LinkId = linkId;
                                process.SendingTo = handler;
                                process.ProcessType = 1;
                                process.Submitter = submitter;
                                process.OrganizationId = organizationId;
                                process.Status = 1;
                                int newProcessId =
_workFlowRepository.CreateFlowProcessAsync(process);
                                if (newProcessId <= 0) return -1;
                            }
                        }
                    }
                    else if (fromNode.NodeType == 1 || fromNode.NodeType == 3)
                    {
                        // 分流节点或者分合流点
                        // 遍历后继节点分别创建待办工作
                        foreach (FlowLink link in links)
                        {
                            List<int> handlers = GetHandlerAsync(link.ToNodeId,
submitter);
                            if (handlers.Count == 0)
                            {
                                return -2;
                            }
                            foreach (int handler in handlers)
                            {
                                // 创建代办工作
                                FlowProcess process = new FlowProcess();
                                process.WorkId = workId;
                                process.NodeId = link.ToNodeId;
                                process.LinkId = link.LinkId;
                                process.SendingTo = handler;
                                process.ProcessType = 1;
                                process.Submitter = submitter;
                                process.OrganizationId = organizationId;
                                process.Status = 1;
                                int newProcessId =
_workFlowRepository.CreateFlowProcessAsync(process);
                                if (newProcessId <= 0) return -1;
                            }
                        }
```

```
                }
            }
            else
            {
                // 需多人协作，只关闭当前处理人待办
                isSucceed                                                    =
_workFlowRepository.CloseUserProcessAsync(processId);
                if (!isSucceed) return -1;
            }
        }
        else
        {
            // 作废所有已办和未办工作
            _workFlowRepository.DisableProcessAsync(workId);
            // 后退到开始节点
            FlowNode                        beginNode                        =
_workFlowRepository.GetFlowBeignNodeAsync(flowId);
            FlowProcess                     beginProcess                     =
_workFlowRepository.GetFlowNodeProcessAsync(workId, beginNode.NodeId);
            beginProcess.Status = 1;
            int                         newProcessId                         =
_workFlowRepository.CreateFlowProcessAsync(beginProcess);
            if (newProcessId <= 0) return -1;
            FlowWorkForm                        form                         =
_workFlowRepository.GetWorkFormAsync(beginProcess.ProcessId);
            // 删除原有关联
            _workFlowRepository.DeleteWorkFormAsync(form.Id);
            form.ProcessId = newProcessId;
            isSucceed = _workFlowRepository.InsertWorkFormAsync(form);
            if (!isSucceed) return -1;
        }
        return 1;
    }

    // 处理业务数据
    public bool DealFormData(int workId, int processId, int nodeId, string
tableName, List<string> columns, List<string> values, int formDataId)
    {
        if (formDataId == 0)
        {
            formDataId = _workFlowRepository.InsertFormDataAsync(tableName,
columns, values);
            if (formDataId > 0)
            {
```

```
                    FlowWorkForm flowWorkForm = new FlowWorkForm();
                    flowWorkForm.WorkId = workId;
                    flowWorkForm.ProcessId = processId;
                    flowWorkForm.NodeId = nodeId;
                    flowWorkForm.TableName = tableName;
                    flowWorkForm.FormDataId = formDataId;
                    return
_workFlowRepository.InsertWorkFormAsync(flowWorkForm);
                }
                else
                {
                    return false;
                }
            }
            else
            {
                return       _workFlowRepository.UpdateFormDataAsync(tableName,
columns, values, formDataId);
            }
        }

        // 是否需要多人协作（false 表示不需要或者是最后一人）
        private bool NeedCooperation(int workId, FlowNode node)
        {
            if (node.Cooperation == 1)
            {
                int                          processCount                          =
_workFlowRepository.GetNodeProcessCountAsync(workId, node.NodeId);
                if (processCount > 1)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
            else
            {
                return false;
            }
        }

        // 获取处理人
```

```
        public List<int> GetHandlerAsync(int toNodeId, int submitterId)
        {
            List<int> handlers = new List<int>();
            User submitter = _userRepository.GetByIdAsync(submitterId);
            FlowNode                          toNode                          =
_workFlowRepository.GetFlowNodeByIdAsync(toNodeId);
            // 先判断节点是否配置指定用户
            if (toNode.UserId != 0)
            {
                handlers.Add(toNode.UserId);
            }
            else if (toNode.PositionId != 0)
            {
                if (toNode.DepartmentId != 0)
                {
                    // 指定部门和职位
                    handlers                                                    =
_userRepository.GetUserByDeptAndPositionAsync(toNode.DepartmentId,
toNode.PositionId);
                }
                else
                {
                    // 只指定了职位,根据用户所属部门向上逐级查找
                    GetUserByPosition(toNode.PositionId, submitter.DepartmentId,
ref handlers);
                }
            }
            else if (toNode.DepartmentId != 0)
            {
                // 只指定了部门,根据用户职位向上逐级查找
                GetUserByDepartment(submitter.PositionId,  toNode.DepartmentId,
ref  handlers);
            }
            else
            {
                // 啥都没指定
                GetUserRecursion(submitter.PositionId,  submitter.DepartmentId,
ref handlers);
            }
            return handlers;
        }

        // 指定职位，根据用户所属部门向上逐级查找
        private void GetUserByPosition(int positionId, int departmentId, ref
```

```
List<int> handlers)
        {
            handlers                                                    =
_userRepository.GetUserByDeptAndPositionAsync(departmentId, positionId);
            if (handlers == null || handlers.Count == 0)
            {
                Department                          dept                    =
_departmentRepository.GetByIdAsync(departmentId);
                int parentId = dept.ParentId;
                if (parentId != 0)
                {
                    // 递归获取处理人
                    GetUserByPosition(positionId, parentId, ref handlers);
                }
            }
        }

        // 指定部门，根据用户所属职位向上逐级查找
        private void GetUserByDepartment(int positionId, int departmentId, ref
List<int> handlers)
        {
            Position position = _positionRepository.GetByIdAsync(positionId);
            int parentId = position.ParentId;
            if (parentId != 0)
            {
                handlers                                                    =
_userRepository.GetUserByDeptAndPositionAsync(departmentId, parentId);
                if (handlers == null || handlers.Count == 0)
                {
                    // 递归获取处理人
                    GetUserByDepartment(parentId, departmentId, ref handlers);
                }
            }
        }

        // 未指定部门和职位，根据用户所属职位和部门向上逐级查找
        private void GetUserRecursion(int positionId, int departmentId, ref
List<int> handlers)
        {
            Position position = _positionRepository.GetByIdAsync(positionId);
            if (position != null && position.ParentId != 0)
            {
                int parentId = position.ParentId;
                handlers                                                    =
```

```
_userRepository.GetUserByDeptAndPositionAsync(departmentId, parentId);
                if (handlers == null || handlers.Count == 0)
                {
                    // 递归部门获取处理人
                    GetUserByPosition(parentId, departmentId, ref handlers);
                    if (handlers == null || handlers.Count == 0)
                    {
                        // 递归职位和部门获取处理人
                        GetUserRecursion(parentId, departmentId, ref handlers);
                    }
                }
            }
        }

        // 获取待办工作列表
        public PageModel<Backlog> GetProcessListAsync(int userId, string
beginDate, string endDate, int status, int pageIndex, int pageSize)
        {
            return _workFlowRepository.GetProcessListAsync(userId, beginDate,
endDate, status, pageIndex, pageSize);
        }

        public FormDesigner GetFlowProcessFormAsync(int processId)
        {
            if (processId <= 0)
            {
                throw new ArgumentException("processId 错误");
            }

            return _workFlowRepository.GetFlowProcessFormAsync(processId);
        }

        public Object GetFlowProcessFormDataAsync(int processId, string
tableName)
        {
            if (processId <= 0)
            {
                throw new ArgumentException("processId 错误");
            }

            return _workFlowRepository.GetFlowProcessFormDataAsync(processId,
tableName);
        }
```

```
        // 获取已处理的表单列表
        public List<Dictionary<string, object>> GetWorkProcessFormListAsync(int workId)
        {
            if (workId <= 0)
            {
                throw new ArgumentException("workId 错误");
            }

            List<Dictionary<string, object>> result = new List<Dictionary<string, object>>();
            List<ProcessForm> list = _workFlowRepository.GetWorkProcessFormListAsync(workId);
            if (list != null)
            {
                foreach(ProcessForm item in list)
                {
                    item.SubmitTimeStr = item.SubmitTime == null ? "" : item.SubmitTime.ToString("yyyy-MM-dd HH:mm:ss");
                    object formData = _workFlowRepository.GetFlowProcessFormDataAsync(item.ProcessId, item.TableName);
                    Dictionary<string, object> dic = new Dictionary<string, object>();
                    dic.Add("form", item);
                    dic.Add("formData", formData);
                    result.Add(dic);
                }
            }

            return result;
        }

        // 处理用户提交的工作
        public int WorkTransferAsync(int processId, string transferUser, int submitter, int organizationId)
        {
            using (TransactionScope scope = new TransactionScope()) // 开启事务
            {
                bool isSucceed = false;
                int result = 1;
                FlowProcess flowProcess = _workFlowRepository.GetProcessByIdAsync(processId);
                // 插入操作步骤
```

```
                FlowStep step = new FlowStep();
                step.WorkId = flowProcess.WorkId;
                step.NodeId = flowProcess.NodeId;
                step.ProcessId = processId;
                step.Submitter = submitter;
                step.OrganizationId = organizationId;
                step.Action = 3; //转办
                isSucceed = _workFlowRepository.CreateFlowStepAsync(step);
                if (!isSucceed) return -1;
                isSucceed                                                    =
_workFlowRepository.CloseUserProcessAsync(processId);
                if (!isSucceed) return -1;
                flowProcess.Status = 1;
                string[] users = transferUser.Split(',');
                for(int i = 0; i < users.Length; i++)
                {
                    flowProcess.SendingTo = users[i].ToInt32();
                    result                                                   =
_workFlowRepository.CreateFlowProcessAsync(flowProcess);
                }
                if (result > 0)
                {
                    scope.Complete();
                }
                return result;
            }
        }

        // 获取待办工作列表
        public PageModel<FlowStep> GetWorkStepsAsync(int workId, int pageIndex,
int pageSize)
        {
            PageModel<FlowStep>                    result                    =
_workFlowRepository.GetWorkStepsAsync(workId, pageIndex, pageSize);
            for (int i = 0; i < result.Data.Count; i++)
            {
                TimeSpan      ts      =      result.Data[i].SubmitTime      -
result.Data[i].BeginTime;
                result.Data[i].StayTime = ts.Days + "天" + ts.Hours + "小时" +
ts.Minutes + "分" + ts.Seconds + "秒";
            }
            return result;
        }
    }
```

```
}
```

程序结束!!!

程序开始!!!

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LongQin.Models;
using Dapper;
using LongQin.Common;

namespace LongQin.Repository
{
    public class WorkFlowRepository : IWorkFlowRepository
    {
        public FormDesigner GetFlowBeignNodeFormAsync(int flowId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select d.* from [wf_node] s left join [des_form] d
on d.id = s.formId where s.flowId = @id and s.status=1 and s.groupseq = 1;";
                var list = conn.Query<FormDesigner>(sql, new { id = flowId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        public FlowNode GetFlowBeignNodeAsync(int flowId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select s.* from [wf_node] s where s.flowId = @id and
s.status=1 and s.groupseq = 1;";
                var list = conn.Query<FlowNode>(sql, new { id = flowId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        public int CreateFlowWorkAsync(FlowWork model)
        {
            using (var conn = DapperFactory.GetConnection())
```

```
            {
                var fields = model.ToFields(removeFields: new List<string>
{ "WorkId", "CreateTime", "CloseTime", "Status" });
                string sql = string.Format("insert into [wf_work] ({0}) values
({1});", string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                sql += ";select @@identity";
                return conn.ExecuteScalar<int>(sql, model);
            }
        }

        public int CreateFlowProcessAsync(FlowProcess model)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                var fields = model.ToFields(removeFields: new List<string>
{ "ProcessId", "ProcessType", "SubmitTime", "Flag", "FlowId", "FormDataId" });
                string sql = string.Format("insert into [wf_process] ({0}) values
({1});", string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                sql += ";select @@identity";
                return conn.ExecuteScalar<int>(sql, model);
            }
        }

        public bool CreateFlowStepAsync(FlowStep model)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                var fields = model.ToFields(removeFields: new List<string>
{"StepId", "SubmitTime", "NodeName", "SubmitterName", "BeginTime", "StayTime"});
                string sql = string.Format("insert into [wf_step] ({0}) values
({1});", string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                return conn.Execute(sql, model) > 0;
            }
        }

        public int InsertFormDataAsync(string tableName, List<string> columns,
List<string> values)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                for (int i = 0; i < values.Count; i++)
                {
                    if (values[i].Contains(","))
                    {
```

```
                    string[] arr = values[i].Split(",");
                    values[i] = "'" + string.Join("|", arr) + "'";
                }
                else
                {
                    values[i] = "'" + values[i] + "'";
                }
            }
            string sql = string.Format("insert into [" + tableName + "] ({0}) values ({1});", string.Join(",", columns), string.Join(",", values));
            sql += ";select @@identity";
            return conn.ExecuteScalar<int>(sql);
        }
    }

    public bool UpdateFormDataAsync(string tableName, List<string> columns, List<string> values, int formId)
    {
        using (var conn = DapperFactory.GetConnection())
        {
            var fieldList = new List<string>();
            for (int i = 0;i < columns.Count; i++)
            {
                fieldList.Add(string.Format("{0}='{1}'",        columns[i], values[i]));
            }

            string sql = string.Format("update [" + tableName + "] set {0} where id = {1}; ", string.Join(",", fieldList), formId);
            return conn.Execute(sql) > 0;
        }
    }

    // 流程表单关联
    public bool InsertWorkFormAsync(FlowWorkForm model)
    {
        using (var conn = DapperFactory.GetConnection())
        {
            var fields = model.ToFields(removeFields: new List<string> { "Id" });
            string sql = string.Format("insert into [wf_workform] ({0}) values ({1});", string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
            return conn.Execute(sql, model) > 0;
        }
```

```
        }

        public List<FlowLink> GetFlowNodeLinksAsync(int nodeId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select * from wf_link where fromnodeid = @nodeId and
status = 1;";
                var list = conn.Query<FlowLink>(sql, new { nodeId = nodeId });
                return list != null ? list.ToList() : null;
            }
        }

        public FlowNode GetFlowNodeByIdAsync(int nodeId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select * from wf_node where nodeId = @nodeId and status
= 1;";
                var list = conn.Query<FlowNode>(sql, new { nodeId = nodeId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        public bool CloseWorkAsync(int workId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("update [wf_work] set status = 0,
closeTime = getdate() where workId = {0};", workId);
                return conn.Execute(sql) > 0;
            }
        }

        public FlowProcess GetFlowNodeProcessAsync(int workId, int nodeId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select top 1 * from [wf_process] where workId = @workId
and nodeId = @nodeId order by processId desc;";
                var list = conn.Query<FlowProcess>(sql, new { workId = workId,
nodeId = nodeId });
                return list != null ? list.FirstOrDefault() : null;
            }
```

```
        }

        public FlowWorkForm GetWorkFormAsync(int processId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select * from [wf_workform] where processId = @processId;";
                var list = conn.Query<FlowWorkForm>(sql, new { processId = processId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        public bool DeleteWorkFormAsync(int id)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "delete from [wf_workform] where id = @id;";
                return conn.Execute(sql, new { Id = id }) > 0;
            }
        }

        public FlowProcess GetProcessByIdAsync(int processId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select p.*, w.formDataId, k.flowId from [wf_process] p left join [wf_workform] w on w.processId = p.processId left join [wf_work] k on k.workId = p.workId where p.processId = @processId;";
                var list = conn.Query<FlowProcess>(sql, new { processId = processId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        // 获取当前节点待办工作数量
        public int GetNodeProcessCountAsync(int workId, int nodeId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select COUNT(0) from [wf_process] where workId = @workId and status = 1 and nodeId = @nodeId;";
                var result = conn.ExecuteScalar<int>(sql, new { workId = workId,
```

```
nodeId = nodeId });
                return result;
            }
        }

        // 获取前置节点待办工作数量
        public int GetPreNodeProcessCountAsync(int workId, int nodeId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select COUNT(0) from [wf_process] where workId =
@workId and status = 1 and nodeId in (select fromNodeId from wf_link where toNodeId
= @nodeId and status = 1);";
                var result = conn.ExecuteScalar<int>(sql, new { workId = workId,
nodeId = nodeId });
                return result;
            }
        }

        public bool CloseNodeProcessAsync(int workId, int nodeId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("update [wf_process] set status = 0
where status = 1 and workId = {0} and nodeId = {1};", workId, nodeId);
                return conn.Execute(sql) > 0;
            }
        }

        public bool CloseUserProcessAsync(int processId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("update [wf_process] set status = 0
where processId = {0};", processId);
                return conn.Execute(sql) > 0;
            }
        }

        // 回退时禁用待办的工作
        public bool DisableProcessAsync(int workId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
```

```
                string sql = string.Format("update [wf_process] set status = 9
where workId = {0};", workId);
                return conn.Execute(sql) > 0;
            }
        }

        // 获取用户待办/已办工作列表
        public PageModel<Backlog> GetProcessListAsync(int userId, string
beginDate, string endDate, int status, int pageIndex, int pageSize)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string whereSql = " and p.status = " + status;
                whereSql += string.IsNullOrEmpty(beginDate) ? "" : " and
p.submitTime >= '" + beginDate + "'";
                whereSql += string.IsNullOrEmpty(endDate) ? "" : " and p.submitTime
<= '" + endDate + "'";
                string countSql = @"select count(1) from [wf_process] p where
p.sendingTo=@userId" + whereSql;
                int total = conn.ExecuteScalar<int>(countSql, new { userId =
userId });
                if (total == 0)
                {
                    return new PageModel<Backlog>();
                }

                string sql = string.Format(@"select * from (select p.*, n.nodeName,
n.formId, u1.nickName as submitterName, w.creator, u2.nickName as creatorName,
w.createTime, d.departmentName,
                    f.flowId, f.flowName, ROW_NUMBER() over (Order by p.processId
desc) as RowNumber from [wf_process] p
                    left join [wf_node] n on n.nodeId = p.nodeId
                    left join [sys_user] u1 on u1.userId = p.submitter
                    left join [wf_work] w on w.workId = p.workId
                    left join [wf_flow] f on f.flowId = w.flowId
                    left join [sys_user] u2 on u2.userId = w.creator
                    left join [sys_department] d on d.departmentId =
u2.departmentId
                    where p.sendingTo=@userId {0}) as b where RowNumber between {1}
and {2};", whereSql, ((pageIndex - 1) * pageSize) + 1, pageIndex * pageSize);
                var list = conn.Query<Backlog>(sql, new { userId = userId });

                return new PageModel<Backlog>
                {
```

```
                    Total = total,
                    Data = list != null ? list.ToList() : null
                };
            }
        }

        public FormDesigner GetFlowProcessFormAsync(int processId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select d.* from [wf_process] p left join [wf_node] s on s.nodeId= p.nodeId left join [des_form] d on d.id = s.formId where p.processId = @processId;";
                var list = conn.Query<FormDesigner>(sql, new { processId = processId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        public Object GetFlowProcessFormDataAsync(int processId, string tableName)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("select s.* from [{0}] s left join wf_workform f on f.formDataId = s.id where f.processId = @processId;", tableName);
                var list = conn.Query<Object>(sql, new { processId = processId });
                return list != null ? list.FirstOrDefault() : null;
            }
        }

        // 获取流程工作表单集合
        public List<ProcessForm> GetWorkProcessFormListAsync(int workId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = @"select p.processId, u.nickName as submitterName, t.submitTime, d.* from [wf_process] p left join [wf_node] s on s.nodeId= p.nodeId left join [des_form] d on d.id = s.formId
                    left join [wf_workform] w on w.processId = p.processId
                    left join [wf_step] t on t.processId = p.processId
                    left join [sys_user] u on t.submitter = u.userId
                    where p.workId = @workId and p.status = 0 and w.id is not null;";
                var list = conn.Query<ProcessForm>(sql, new { workId = workId });
```

```
                return list != null ? list.ToList() : null;
            }
        }

        // 获取工作历史记录
        public PageModel<FlowStep> GetWorkStepsAsync(int workId, int pageIndex,
int pageSize)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string countSql = @"select count(1) from [wf_step] s where
s.workId=@workId";
                int total = conn.ExecuteScalar<int>(countSql, new { workId =
workId });
                if (total == 0)
                {
                    return new PageModel<FlowStep>();
                }

                string sql = string.Format(@"select * from (select s.*, n.nodeName,
u.nickName as submitterName, p.submitTime as beginTime,
                    ROW_NUMBER() over (Order by s.stepId desc) as RowNumber from
[wf_step] s
                    left join [wf_node] n on n.nodeId = s.nodeId
                    left join [sys_user] u on u.userId = s.submitter
                    left join [wf_process] p on s.processId = p.processId
                    where s.workId=@workId) as b where RowNumber between {0} and
{1};", ((pageIndex - 1) * pageSize) + 1, pageIndex * pageSize);
                var list = conn.Query<FlowStep>(sql, new { workId = workId });

                return new PageModel<FlowStep>
                {
                    Total = total,
                    Data = list != null ? list.ToList() : null
                };
            }
        }

        // 获取表列集合
        public List<string> GetTableColumnsAsync(string tableName)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("select name from syscolumns where id
```

```
= object_id('{0}');", tableName);
                var list = conn.Query<string>(sql);
                return list != null ? list.ToList() : null;
            }
        }

        // 获取表数据集合
        public List<string> GetTableValueAsync(int workId, string tableName)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("select top 1 s.* from [{0}] s left join
wf_workform f on f.formDataId = s.id where f.workId = {1} and f.tableName = '{0}'
order by id desc;", tableName, workId);
                var data = conn.QueryFirst(sql);
                var fields = data as IDictionary<string, object>;
                List<string> result = new List<string>();
                fields.ForEach(item                                          =>
result.Add(item.Value.ToStringOrDefault()));
                return result;
            }
        }
    }
}
```

程序结束!!!

程序开始!!!

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LongQin.Models;
using Dapper;

namespace LongQin.Repository
{
    public class FlowDesignerRepository : IFlowDesignerRepository
    {
        public int CreateFlowAsync(FlowDesigner model)
        {
            using (var conn = DapperFactory.GetConnection())
```

```
            {
                var fields = model.ToFields(removeFields: new List<string> { "FlowId",
"CreateTime" });
                string sql = string.Format("insert into [wf_flow] ({0}) values ({1});",
string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                sql += ";select @@identity";
                return conn.ExecuteScalar<int>(sql, model);
            }
        }

        public PageModel<FlowDesigner> GetFlowListAsync(int organizationId, int pageIndex,
int pageSize)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string countSql = string.Format(@"select count(1) from [wf_flow] where
OrganizationId = {0} and Status=1;", organizationId);
                int total = conn.ExecuteScalar<int>(countSql);
                if (total == 0)
                {
                    return new PageModel<FlowDesigner>();
                }

                string sql = string.Format(@"select * from (select *, ROW_NUMBER() over
(Order by FlowId desc) as RowNumber from [wf_flow] where OrganizationId = {0} and Status=1)
as b where RowNumber between {1};", organizationId, ((pageIndex - 1) * pageSize) + 1 + "
and " + pageIndex * pageSize);
                var list = conn.Query<FlowDesigner>(sql);

                return new PageModel<FlowDesigner>
                {
                    Total = total,
                    Data = list != null ? list.ToList() : null
                };
            }
        }

        public FlowDesigner GetFlowByIdAsync(int id)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "select * from [wf_flow] where flowid = @id and Status=1;";
                var list = conn.Query<FlowDesigner>(sql, new { id = id });
                return list != null ? list.FirstOrDefault() : null;
```

```
        }
    }

    public bool UpdateFlowAsync(FlowDesigner model)
    {
        using (var conn = DapperFactory.GetConnection())
        {
            var fields = model.ToFields(removeFields: new List<string>
            {
                "FlowId",
                "CreateTime",
                "Creator",
                "OrganizationId"
            });

            if (fields == null || fields.Count == 0)
            {
                return false;
            }

            var fieldList = new List<string>();
            foreach (var field in fields)
            {
                fieldList.Add(string.Format("{0}=@{0}", field));
            }

            string sql = string.Format("update [wf_flow] set {0} where flowid=@FlowId;",
string.Join(",", fieldList));
            return conn.Execute(sql, model) > 0;
        }
    }

    public bool DeleteFlowAsync(int id)
    {
        using (var conn = DapperFactory.GetConnection())
        {
            string sql = "update [wf_flow] set Status=0 where flowid=@FlowId;";
            return conn.Execute(sql, new { FlowId = id }) > 0;
        }
    }

    public int CreateNodeAsync(FlowNode model)
    {
        using (var conn = DapperFactory.GetConnection())
```

```
            {
                var fields = model.ToFields(removeFields: new List<string> { "NodeId",
"FormName", "CreateTime", "Gid" });
                string sql = string.Format("insert into [wf_node] ({0}) values ({1});",
string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                sql += ";select @@identity";
                return conn.ExecuteScalar<int>(sql, model);
            }
        }

        public bool UpdateNodeAsync(FlowNode model)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                var fields = model.ToFields(removeFields: new List<string>
                {
                    "NodeId",
                    "FormName",
                    "CreateTime",
                    "Creator",
                    "OrganizationId",
                    "Gid"
                });

                if (fields == null || fields.Count == 0)
                {
                    return false;
                }

                var fieldList = new List<string>();
                foreach (var field in fields)
                {
                    fieldList.Add(string.Format("{0}=@{0}", field));
                }
                fieldList.Add("Status=1");

                string sql = string.Format("update [wf_node] set {0} where nodeid=@NodeId;",
string.Join(",", fieldList));
                return conn.Execute(sql, model) > 0;
            }
        }

        public bool DeleteNodeAsync(int flowId)
        {
```

```
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "update [wf_node] set Status=0 where FlowId=@FlowId;";
                return conn.Execute(sql, new { FlowId = flowId }) > 0;
            }
        }

        public List<FlowNode> GetFlowNodesAsync(int flowId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format(@"select * from [wf_node] where FlowId = {0} and
Status=1;", flowId);
                var list = conn.Query<FlowNode>(sql).ToList();
                return list;
            }
        }

        public bool DeleteLinkAsync(int flowId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "delete from [wf_link] where FlowId=@FlowId;";
                return conn.Execute(sql, new { FlowId = flowId }) > 0;
            }
        }

        public bool CreateLinkAsync(FlowLink model)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                var fields = model.ToFields(removeFields: new List<string> { "LinkId",
"CreateTime" });
                string sql = string.Format("insert into [wf_link] ({0}) values ({1});",
string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                return conn.Execute(sql, model) > 0;
            }
        }

        public List<FlowLink> GetFlowLinksAsync(int flowId)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format(@"select * from [wf_link] where FlowId = {0} and
```

```
Status=1;", flowId);
                var list = conn.Query<FlowLink>(sql).ToList();
                return list;
            }
        }
    }
}
```

程序结束!!!

程序开始!!!

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LongQin.Models;
using Dapper;

namespace LongQin.Repository
{
    public class FormDesignerRepository : IFormDesignerRepository
    {
        public bool CreateTableAsync(string tableName, List<TableColumn> list)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string columns = "";
                string descriptions = "";
                for (int i = 0; i < list.Count; i++)
                {
                    TableColumn column = list[i];
                    columns += "[" + column.ColumnName + "] " + column.ColumnType + " " +
column.IsNull + ",";
                    descriptions += "EXEC sys.sp_addextendedproperty
@name=N'MS_Description', @value=N'" + column.Description + "' ,
@level0type=N'SCHEMA',@level0name=N'dbo', @level1type=N'TABLE',@level1name=N'" +
tableName + "', @level2type=N'COLUMN',@level2name=N'" + column.ColumnName + "'";
                }
                columns += "[creator] " + "[int]" + " " + "NULL" + ",";
                columns += "[createTime] " + "[datetime]" + " " + "NULL" + ",";
                columns += "[organizationId] " + "[int]" + " " + "NULL" + ",";
                columns += "[status] " + "[tinyint]" + " " + "NULL" + ",";
```

```
                string sql = string.Format("CREATE TABLE [{0}]([id][int] IDENTITY(1, 1) NOT NULL,{1}CONSTRAINT[PK_{0}] PRIMARY KEY CLUSTERED([id] ASC)WITH(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON[PRIMARY]) ON[PRIMARY]", tableName, columns);
                sql += descriptions;
                return conn.Execute(sql) > 0;
            }
        }

        public bool CreateFormAsync(FormDesigner model)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                var fields = model.ToFields(removeFields: new List<string> { "Id", "CreateTime", "TableColumns" });
                if (fields == null || fields.Count == 0)
                {
                    return false;
                }

                string sql = string.Format("insert into [des_form] ({0}) values ({1});", string.Join(",", fields), string.Join(",", fields.Select(n => "@" + n)));
                return conn.Execute(sql, model) > 0;
            }
        }

        public PageModel<FormDesigner> GetListAsync(int organizationId, int pageIndex, int pageSize)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string countSql = string.Format(@"select count(1) from [des_form] where OrganizationId = {0} and Status=1;", organizationId);
                int total = conn.ExecuteScalar<int>(countSql);
                if (total == 0)
                {
                    return new PageModel<FormDesigner>();
                }

                string sql = string.Format(@"select * from (select *, ROW_NUMBER() over (Order by Id desc) as RowNumber from [des_form] where OrganizationId = {0} and Status=1) as b where RowNumber between {1};", organizationId, ((pageIndex - 1) * pageSize) + 1 + " and " + pageIndex * pageSize);
                var list = conn.Query<FormDesigner>(sql);
```

```
            return new PageModel<FormDesigner>
            {
                Total = total,
                Data = list != null ? list.ToList() : null
            };
        }
    }

    public FormDesigner GetByIdAsync(int id)
    {
        using (var conn = DapperFactory.GetConnection())
        {
            string sql = "select * from [des_form] where id = @id and Status=1;";
            var list = conn.Query<FormDesigner>(sql, new { id = id });
            return list != null ? list.FirstOrDefault() : null;
        }
    }

    public bool UpdateAsync(FormDesigner model)
    {
        using (var conn = DapperFactory.GetConnection())
        {
            var fields = model.ToFields(removeFields: new List<string>
            {
                "id",
                "CreateTime",
                "Status",
                "Creator",
                "OrganizationId",
                "TableColumns"
            });

            if (fields == null || fields.Count == 0)
            {
                return false;
            }

            var fieldList = new List<string>();
            foreach (var field in fields)
            {
                fieldList.Add(string.Format("{0}=@{0}", field));
            }
```

```
                string sql = string.Format("update [des_form] set {0} where id=@Id;",
string.Join(",", fieldList));
                return conn.Execute(sql, model) > 0;
            }
        }

        public bool DeleteAsync(int id)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = "update [des_form] set Status=0 where id=@Id;";
                return conn.Execute(sql, new { Id = id }) > 0;
            }
        }

        public bool DeleteTableAsync(string tableName)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("drop table {0};", tableName);
                return conn.Execute(sql) > 0;
            }
        }

        public bool InsertFormDataAsync(string tableName, List<string> columns,
List<string> values)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string sql = string.Format("insert into [" + tableName + "] ({0}) values
({1});", string.Join(",", columns), string.Join(",", values));
                return conn.Execute(sql) > 0;
            }
        }

        public int GetTableCountAsync(string tableName)
        {
            using (var conn = DapperFactory.GetConnection())
            {
                string countSql = string.Format("select count(1) from sysobjects where id
= object_id('{0}');", tableName);
                return conn.ExecuteScalar<int>(countSql);
            }
        }
```

```
    }
}
```

程序结束!!!

程序开始!!!

```
using LongQin.Repository;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using LongQin.Models;
using LongQin.Common;
using System.Data;

namespace LongQin.Service
{
    public class FlowDesignerService : ServiceBase, IFlowDesignerService
    {
        IFlowDesignerRepository _flowDesignerRepository =
AutofacRepository.Resolve<IFlowDesignerRepository>();

        public FlowDesignerService()
        {
            base.AddDisposableObject(_flowDesignerRepository);
        }

        public PageModel<FlowDesigner> GetFlowListAsync(int organizationId, int pageIndex,
int pageSize)
        {
            return _flowDesignerRepository.GetFlowListAsync(organizationId, pageIndex,
pageSize);
        }

        public FlowDesigner GetFlowByIdAsync(int flowId)
        {
            if (flowId <= 0)
            {
                throw new ArgumentException("id错误");
            }

            return _flowDesignerRepository.GetFlowByIdAsync(flowId);
```

```
        }

        public string GetFlowJson(int flowId)
        {
            var nodes = _flowDesignerRepository.GetFlowNodesAsync(flowId);
            var links = _flowDesignerRepository.GetFlowLinksAsync(flowId);
            StringBuilder sb = new StringBuilder("({rects:{");
            if (nodes != null && nodes.Count != 0)
            {
                int i = 0;
                foreach (FlowNode node in nodes)
                {
                    sb.Append("rect" + i + ":{data:{\"id\":\"" + node.NodeId +
"\",\"name\":\"" + node.NodeName + "\",\"rectType\":\"" + node.NodeType
                        + "\",\"formId\":\"" + node.FormId + "\",\"cooperation\":\"" +
node.Cooperation
                        + "\",\"virtual\":\"" + node.Virtual + "\",\"departmentId\":\"" +
node.DepartmentId
                        + "\",\"positionId\":\"" + node.PositionId + "\",\"userId\":\"" +
node.UserId
                        + "\",\"remark\":\"" + node.Description + "\"}, attr:{x:" +
node.PositionX + ",y:" + node.PositionY
                        + "}, text:{\"text\":\"" + node.NodeName + "\"}}");
                    sb.Append(",");
                    i++;
                }
                sb = sb.Remove(sb.Length - 1, 1);
            }
            sb.Append("},paths:{");
            if (links != null && links.Count != 0)
            {
                int j = 0;
                foreach (FlowLink link in links)
                {
                    sb.Append("path" + j + ":{from:" + link.FromNodeId + ",to:" +
link.ToNodeId + ",data:{\"id\":\"" + link.LinkId + "\",\"name\":\"" + link.LinkName
                        + "\",\"formId\":\"" + link.FormId + "\",\"field\":\"" +
link.Field + "\",\"operator\":\"" + link.Operator + "\",\"operatorValue\":\"" +
link.OperatorValue + "\",\"remark\":\"" + link.Description
                        + "\"}, text:{\"text\":\"" + link.LinkName + "\"}, textPos:{x:" +
link.PositionX + ",y:" + link.PositionY + "}}");
                    sb.Append(",");
                    j++;
                }
```

```
            sb = sb.Remove(sb.Length - 1, 1);
        }
        sb.Append("}})");
        return sb.ToString();
    }

    public bool SaveAsync(FlowDesigner model, string nodes, string links)
    {
        bool result = false;
        if (model.FlowId == 0)
        {
            model.FlowId = _flowDesignerRepository.CreateFlowAsync(model);
            if (model.FlowId > 0)
            {
                result = true;
            }
        }
        else
        {
            result = _flowDesignerRepository.UpdateFlowAsync(model);
        }
        if (result)
        {
            _flowDesignerRepository.DeleteNodeAsync(model.FlowId);
            _flowDesignerRepository.DeleteLinkAsync(model.FlowId);
            Dictionary<string, int> nodeDics = new Dictionary<string, int>();
            string[] nodeArr = nodes.Split(';');
            for (int i = 0; i < nodeArr.Length; i++)
            {
                string[] node = nodeArr[i].Split(',');
                FlowNode flowNode = new FlowNode();
                flowNode.Gid = node[0];
                flowNode.PositionX = node[1].ToInt32();
                flowNode.PositionY = node[2].ToInt32();
                flowNode.NodeId = node[3].ToInt32();
                flowNode.NodeName = node[4];
                flowNode.NodeType = node[5].ToInt32();
                flowNode.FormId = node[6].ToInt32();
                flowNode.Virtual = node[7].ToInt32();
                flowNode.Cooperation = node[8].ToInt32();
                flowNode.DepartmentId = node[9].ToInt32();
                flowNode.PositionId = node[10].ToInt32();
                flowNode.UserId = node[11].ToInt32();
                flowNode.Description = node[12];
```

```
                    flowNode.Groupseq = node[13].ToInt32();
                    flowNode.IsApproval = node[14].ToInt32();
                    flowNode.FlowId = model.FlowId;
                    flowNode.Creator = model.Creator;
                    flowNode.OrganizationId = model.OrganizationId;
                    if (flowNode.NodeId == 0)
                    {
                        flowNode.NodeId =
_flowDesignerRepository.CreateNodeAsync(flowNode);
                    }
                    else
                    {
                        _flowDesignerRepository.UpdateNodeAsync(flowNode);
                    }
                    nodeDics.Add(flowNode.Gid, flowNode.NodeId);
                }
                string[] linkArr = links.Split(';');
                for (int j = 0; j < linkArr.Length; j++)
                {
                    string[] link = linkArr[j].Split(',');
                    FlowLink flowLink = new FlowLink();
                    int fromNodeId = 0;
                    nodeDics.TryGetValue(link[0], out fromNodeId);
                    flowLink.FromNodeId = fromNodeId;
                    int toNodeId = 0;
                    nodeDics.TryGetValue(link[1], out toNodeId);
                    flowLink.ToNodeId = toNodeId;
                    flowLink.LinkName = link[2];
                    flowLink.PositionX = link[3].ToInt32();
                    flowLink.PositionY = link[4].ToInt32();
                    flowLink.FormId = link[5].ToInt32();
                    flowLink.Field = link[6];
                    flowLink.Operator = link[7];
                    flowLink.OperatorValue = link[8];
                    flowLink.Description = link[9];
                    flowLink.FlowId = model.FlowId;
                    flowLink.Creator = model.Creator;
                    flowLink.OrganizationId = model.OrganizationId;
                    _flowDesignerRepository.CreateLinkAsync(flowLink);
                }
            }
            return result;
        }
```

```
        public bool DeleteFlowAsync(int flowId)
        {
            if (flowId <= 0)
            {
                throw new ArgumentException("id错误");
            }

            return _flowDesignerRepository.DeleteFlowAsync(flowId);
        }
    }
}
```

程序结束!!!