北京泰合佳通信息技术有限公司

HTML5 工作流
详细设计说明书

魏远阳
2015-11-10

修订记录

时间	版本号	修订内容	修订人
2015 年 11 月 10 日	1.0	起草	魏远阳
2015 年 11 月 10 日	1.1	创建	彭文超

审核记录

时间	版本号	审核结果	审核人

目 录

1 范围

1.1 缩略语

本工作流采用 HTML5+bootstrap+angularjs 实现，支持子流程、领单、挂起等操作；支持虚拟节点；支持节假日工作日计算；流程与业务完全分离，流程引擎不负责任何业务操作。

1.2 文档概述

本文档的流程设计由广州分公司开发人员彭文超编写，系统框架由广州分公司架构组人员李国雄、黄庆鑫等人完成，流程引擎由北京研发人员魏远阳编写。

2 引用文档

序号	文档名称	文档编号	作者/修订者	发布日期	出版单位	备注
1	jquery.jsPlumb-1.7.2					
2	iscroll-zoom.js					
3						
4						

3 遵循原则

该详细设计说明书主要遵循《工作流引擎数据库设计文档 V0.1.pdm》来编写。在符合客户需求的基础上，设计出适用于开发人员开发的文档。

3.1 设计原则

- 模块设计原则

  面向对象设计原则

- 接口设计原则

  共享外部数据、参数表达形式和传递方式、层次调用关系等。

3.2 命名规则

1. 命名方法

   Pascal：所有单词第一个字母大写，其他字母小写。

   camelCase：除了第一个单词，所有单词第一个字母大写，其他字母小写。

   匈牙利命名法：通过在变量名之前增加小写字母的符号前缀，以标识变量的属性、类型、作用域等参数。

2. 建议使用命名方法

   类、方法、接口、属性、全局变量、只读变量等用Pascal命名法。

   局部变量、参数名称等用camel命名法。

   实体属性、私有变量、常量、静态变量等用匈牙利命名法。

3. 类

   示例：Customer

4. 方法

   使用动词-名词的方法来命名对给定对象执行的特定操作，如CalculateInvoiceTotal()

5. 变量

   用有意义的，描述性的词语来命名变量，不要使用单个字母的变量像i,n,x等。用于循环迭代的变量除外：

```
for(int i=0;i<count;i++)
{
    //To do some...
}
```

6. 常量

   常量全用大写字母命名，用下划线分割单词。

   例如：const int INT_MAX_LENGTH=80;

7. 注释，类、方法、变量、属性、事件、委托名称必须添加注释

   1>单行注释 "//"

   2>多行注释"/* */"

3>文档注释"///"

8. 变量要加修饰符，同类型的放在一起。
9. 类要小于 1000 行，避免编写代码量大的方法。

4 系统概述

4.1 工作流设计

4.1.1 业务说明

此版本的流程设计器是以 AngularJs + JsPlumb + Iscroll-Zoom 等技术是界面更加的清新，简单，易操作。流程在包含原有功能之外还加入了子流程，会签，节点授权，工作日处理，虚拟节点，链路条件.....新的功能点，更好的适合逻辑业务的展开。

4.1.2 逻辑设计

4.1.2.1 设计器初始化界面

1、界面左边呈现流程名称以及节点信息，并且可以针对单个节点进行“编辑”，“授权”，“删除”操作

2、界面右边以图形化的形式呈现流程节点以及链路信息，每一条链路上面可以进行针对该链路的条件设置，如果设置了条件，则必须满足之后才能通过此链路，到达指定节点。双击节点可以进行编辑，双击链路可以删除链路，单击“条件”可以编辑该链路条件。

3、按钮区域
（1）、打开流程 所有的流程需要从流程列表中进行打开，打开之后会加载节点信息和链路信息，并且以图形化的形式进行展现。否则界面上除流程按钮之外的其他按钮均不可用。
（2）、**增加节点** 打开节点增加界面，必须已经选择了流程之后可进行操作。
（3）、**保存流程** 进行流程保存，
（4）、**提交流程** 将打开的流程置为可用状态，其他同类型的流程全部置为不可用状态

4.1.2.2 流程

1、**流程列表** 展现所有流程信息，需要分页，增加筛选条件：流程名称，流程类型，流程状态。每一行数据可以编辑，删除，打开流程按钮点击之后将此流程展现在设计器初始化界面。需要增加流程按钮。

2、**增加/编辑流程** 流程名称必填，流程类型和启动方式必须选择。

4.1.2.3 节点

基本信息 节点类型，运行模式，执行者，超时处理为下拉选择取字典，节点时限，超时时限，序号，警告期限必须为数字必填。节点的 URL 弹出模态窗体进行选择，同时在模态窗体中可以对 URL 进行添加和修改。

子流程信息 流程类型取字典，并且流程名称根据流程类型联动，取当前在用流程。启动方式取字典。填写的子流程信息录入子流程表中（wf_node_subflow）。

多选框部分 根据实际情况选择

4.1.3 处理设计（或算法）

（一）、保存流程

```
/// <summary>
/// 保存流程信息
/// </summary>
/// <param name="model"></param>
/// <returns></returns>
1 个引用
public MessageEntity SaveFlow(FlowModel model)
```

1、获取当前在用流程 ID 与保存的流程 ID 进行比较，如果大于等于在用的流程 ID，则删除原来链路，再进行链路保存并且修改节点所在的 XY 轴。

2、当前流程 ID 小于在用的流程 ID 时说明是历史流程，则创建一个新的临时流程版本，状态为不可用。

（二）、提交流程

```
/// <summary>
/// 提交按钮
/// </summary>
/// <param name="wfId"></param>
/// <returns></returns>
1 个引用
public MessageEntity SubmitFlow(decimal wfId)
{
```

1、提交流程 ID 必须大于或者等于在用的流程 ID 时才可以提交。

a、等于在用流程，将原来的所有这种类型的流程全部置为不可用状态，创建新流程并且将原来的节点信息赋值给新流程，然后通过原始节点和链路保存新的链路信息。

b、大于时将全部流程置为不可用状态，将该临时版本流程置为可用。

（四）、JsPlumb 插件主要事件：

1、dblclick 双击事件

2、addEndpoint 添加端口（TopRight 右上方 BottomRight 右下方 LeftMiddle 左方）

3、connectPorts 连接线

4、deleConnect、deleteLink 删除连接线

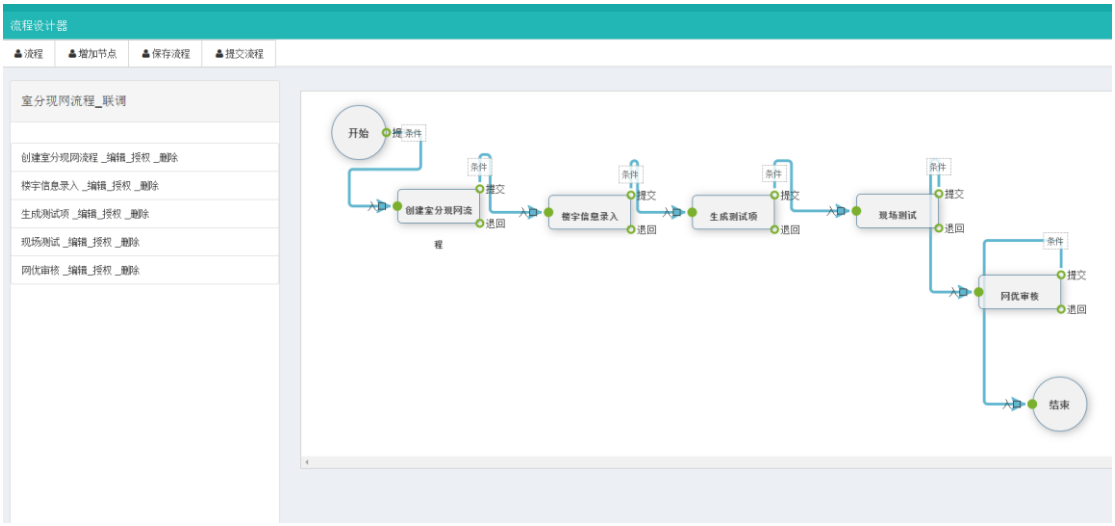
5、JsOverlay Overlay 样式

6、jsPlumbConnection 连接结束后事件 ， 连接线不能连接自己，同一状态出口不能重复连接，必须连接开始和结束节点

7、btnDeleteNode 删除节点

4.1.4 界面设计

流程设计界面 示例：

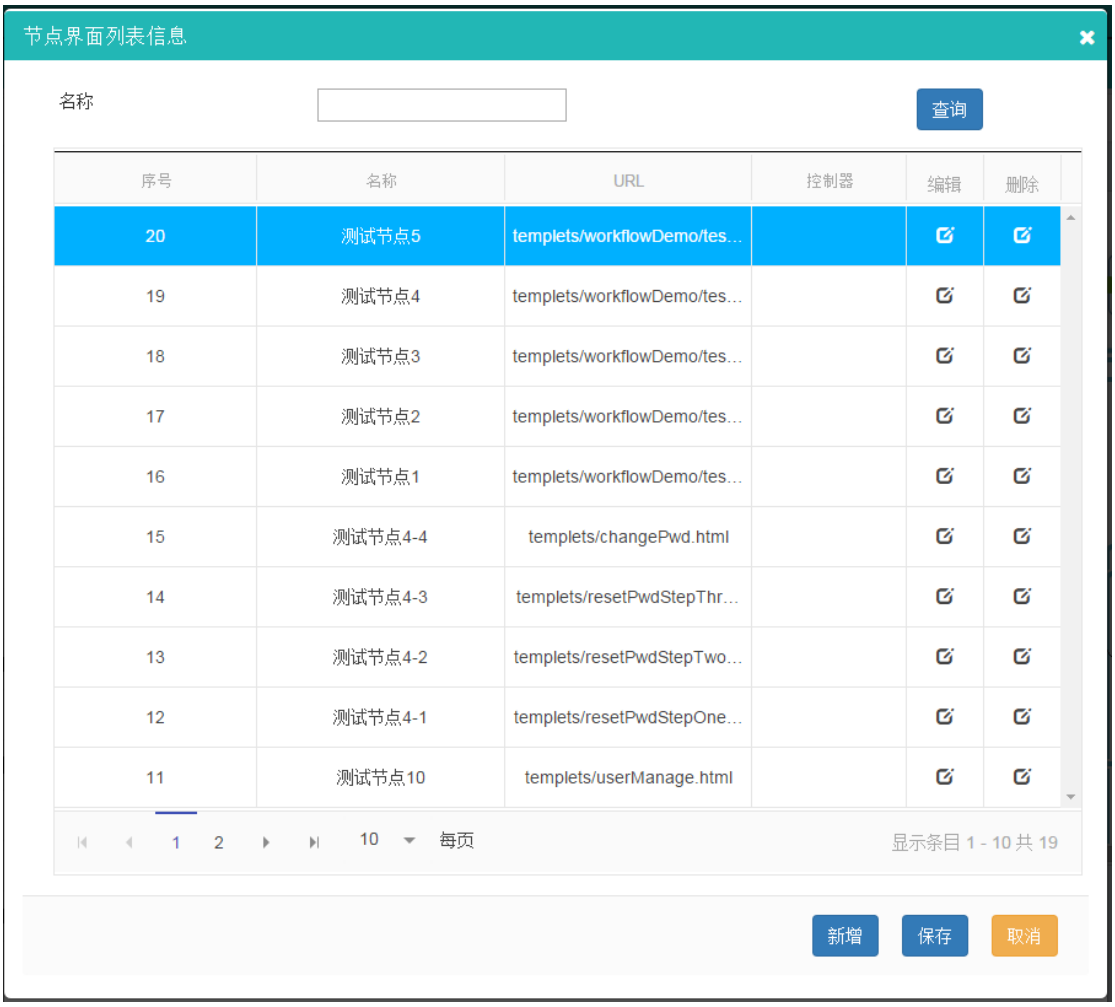


流程列表界面 示例：

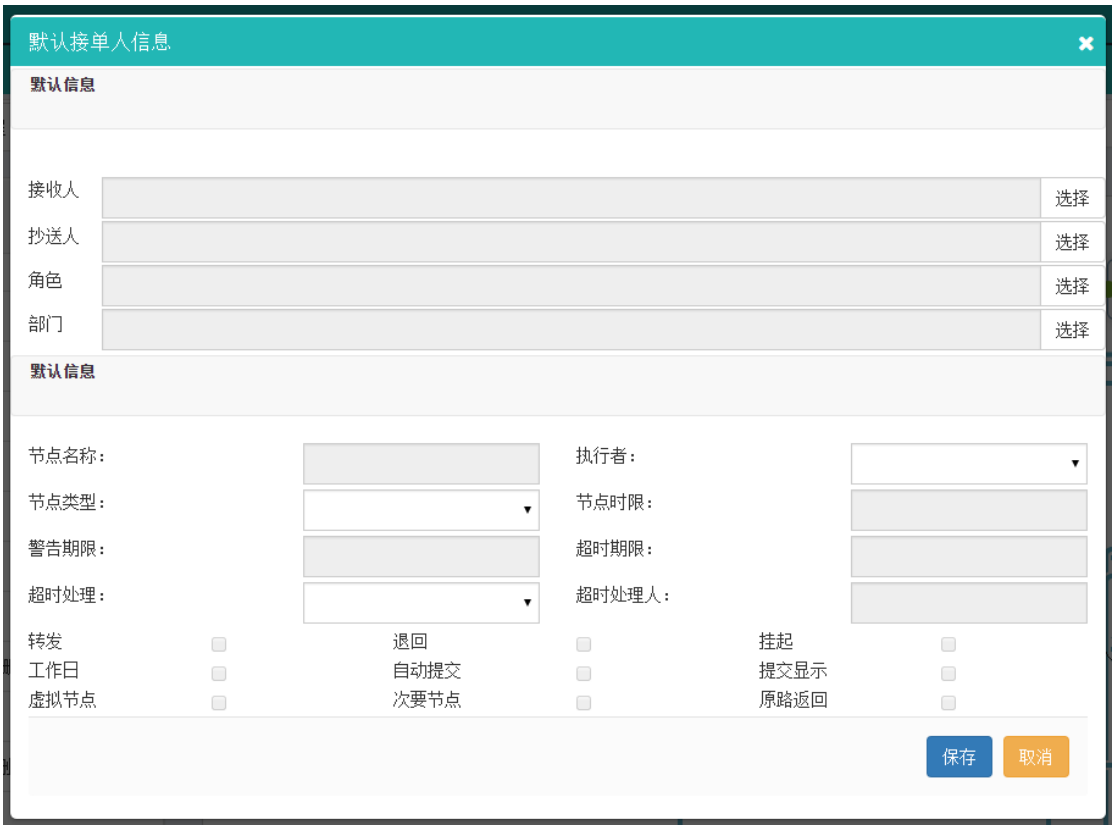
流程添加/编辑界面 示例：

节点编辑/添加界面 示例：

界面选择/新增/编辑界面 示例：


节点界面列表信息
名称
查询
序号
名称
URL
控制器
编辑
删除
20
测试节点5
templets/workflowDemo/tes...
19
测试节点4
templets/workflowDemo/tes...
18
测试节点3
templets/workflowDemo/tes...
17
测试节点2
templets/workflowDemo/tes...
16
测试节点1
templets/workflowDemo/tes...
15
测试节点4-4
templets/changePwd.html
14
测试节点4-3
templets/resetPwdStepThr...
13
测试节点4-2
templets/resetPwdStepTwo...
12
测试节点4-1
templets/resetPwdStepOne...
11
测试节点10
templets/userManage.html
1
2
10
每页
显示条目 1 - 10 共 19
新增
保存
取消

默认节点（授权）界面 示例：



链路条件编辑/添加界面 示例：

4.1.5 程序说明

1、目前程序采用的是 MySql 数据库。
2、设计器插件是用 Jsplumb 插件。
3、流程界面缩放插件是 iscroll-zoom。

设计器缩放代码

```
var myScroll = new IScroll('#wrapper', {
    zoom: true,
    scrollX: true,
    zoomMax: 5,
    zoomMin: 0.5,
    scrollY: true,
    mouseWheel: true,
    wheelAction: 'zoom'
});

document.addEventListener('touchmove', function (e) { e.preventDefault(); }, false);
```

设计器实例创建

```
var instance = jsPlumb.getInstance({
    // 默认拖动参数
    DragOptions: { cursor: 'pointer', zIndex: 2000 },
    // the overlays to decorate each connection with.  note that the label o
    // case it returns the 'labelText' member that we set on each connection
    ConnectionOverlays: [
        ["Arrow", { location: 1, foldback: 0.8 }]
    ],
    Container: "content"
});
```

```
// 节点支持拖拽属性
instance.draggable($('.node'));
```

```
// 连接线结束后事件
instance.bind("jsPlumbConnection", function (conn, originalEvent)...);
```

打开设计器流程初始化呈现

```
// 打开流程
$scope.OpenFlowBaseInfo = function (FlowItem) {

    // 打开流程前，存在面板上存在流程则清除
    if (_flowNodes != null || _linkList != null) {
        $scope.cleanFlow();
    }

    // 获取流程节点信息
    httpService.post('WorkFlow', 'DesignerServices', 'GetFlowNodes', { wfId: FlowItem.Wf_Id }).then(function (data) {

        // 开始和结束节点
        $scope.addPorts(instance, "", 2);

        // 节点
        _flowNodes = data;

        // HTML 模板
        $('#menHeader').html("<h4>" + FlowItem.Wf_Name + "</h4>");

        var htmlStr = "<ul class='list-group' id='ulMenu'>"; // <li class='list-group-item'>" + FlowItem.Wf_Name + "</li>

        for (var i = 0; i < data.length; i++) {

            // 开始和结束节点不放入
            htmlStr += "<li class='list-group-item' id='isShow" + data[i].Node_Id + "'><span  style='cursor:pointer;' ng-click='btnAddNodesSvg(\"" + data[i].Node_Name + "\","

            // 插入节点
            var node1 = $scope.addNode('flow-panel', data[i].Node_Id, data[i].Node_Name, { x: data[i].X + "px", y: data[i].Y + "px" });

            // 节点出口
            $scope.addPorts(instance, node1, 1);
        }

        // 节点支持拖拽属性
        instance.draggable($('.node'));

        htmlStr += "</ul>";
```

```
        // 节点支持拖拽属性
        instance.draggable($('.node'));

        htmlStr += "</ul>";

        // 将节点信息放入
        $compile($('#control-panel').html(htmlStr))($scope);

        postModel = {
            workflowId: FlowItem.Wf_Id
        };

        // 链路
        httpService.post('WorkFlow', 'WorkflowLinksDal', 'QueryLinksByWfId', { workflowId: FlowItem.Wf_Id }).then(function (dataLink) {

            // 加入开始和结束节点
            data.push({ Node_Id: 1, Node_Type: 1, Node_Name: "开始", Wf_Id: FlowItem.Wf_Id, X: "5", Y: "20" });
            data.push({ Node_Id: 2, Node_Type: 3, Node_Name: "结束", Wf_Id: FlowItem.Wf_Id, X: "820", Y: "380" });

            // 链路集合
            for (var j = 0; j < dataLink.length; j++) {


                var nodeStart = null;

                var nodeEnd = null;

                // 根据链路获取开始节点和结束节点
                for (var i = 0; i < data.length; i++) {

                    // 开始节点
                    if (data[i].Node_Id == dataLink[j].Start_Node_Id) {
                        // 创建节点
                        nodeStart = jsPlumb.getSelector('#' + data[i].Node_Id)[0];
                    }


                    // 结束节点
                    if (data[i].Node_Id == dataLink[j].End_Node_Id) {
                        // 创建节点
                        nodeEnd = jsPlumb.getSelector('#' + data[i].Node_Id)[0];
                        ,

                    }

                    // 节点不为空时进行连线 连接线
                    if (nodeStart != null && nodeEnd != null) {

                        $scope.connectPorts(instance, nodeStart, nodeStart.innerText, nodeEnd, nodeEnd.innerText, dataLink[j].Executive_Action);
                    }
                }

            },
            function (errorMsg) {

                Notification.error({ message: "获取链路信息失败!\n错误信息:" + errorMsg, delay: 6000 });

            });

        },
        function (errorMessage) {

            Notification.error({ message: "获取节点信息失败!\n错误信息:" + errorMessage, delay: 6000 });
        });
    }
```

删除节点以及点

```
    //   清空流程信息，从新打开
    $scope.cleanFlow = function () {

        // 删除端点

        for (var i = 0; i < _flowNodes.length; i++) {

            // 开始和结束节点不删除
            if (!(_flowNodes[i].Node_Id == 1 || _flowNodes[i].Node_Id == 2)) {
                // 删除端点
                var ac = ["-TopRight", "-BottomRight", "-LeftMiddle"];

                for (var j = 0; j < ac.length; j++) {

                    instance.deleteEndpoint(_flowNodes[i].Node_Id+ac[j]);
                }

                // 删除节点
                $("#" + _flowNodes[i].Node_Id).remove();
            }
        }
    }
```

删除连接线

```
// 删除连接线
$scope.deleConnect = function (c) {
    //清空连线
    instance.detach(c);
    //更新逻辑关系
    var sourceid = c.sourceId;
    var targerid = c.targetId;
    var parentid = $("#" + targerid).attr("parentId");
    if (parentid != undefined) {
        parentid = parentid.replace("|" + sourceid, "").replace(sourceid + "|", "");
        $("#" + targerid).attr("parentId", parentid);
    }

    // 节点支持拖拽属性
    instance.draggable($('.node'));


    // 删除集合中的链路
    for (var i = 0; i < _linkList.length; i++) {

        if (_linkList[i].Start_Node_Id == sourceid && _linkList[i].End_Node_Id == targerid) {
            _linkList.splice(i, 1);
            return;
        }
    }

}
```

添加端点

```
    },
// 添加线与线之间的点 1 普通节点 2开始结束
$scope.addPorts = function (instance, node, type) {

    // 开始和结束节点
    if (type == 2) {

        // 开始节点只有出口
        instance.addEndpoint(jsPlumb.getSelector('#1')[0], sourceEndpoint, {
            anchor: "RightMiddle", uuid: "1-TopRight"
        });

        // 结束节点只有入口
        instance.addEndpoint(jsPlumb.getSelector('#2')[0], targetEndpoint, {
            anchor: "LeftMiddle", uuid: "2-LeftMiddle"
        });

    } else {

        // 普通节点
        instance.addEndpoint(node, sourceEndpoint, {
            anchor: "TopRight", uuid: node.getAttribute("id") + "-TopRight"
        });

        instance.addEndpoint(node, escEndpoint, {
            anchor: "BottomRight", uuid: node.getAttribute("id") + "-BottomRight"
        });

        //// 入口
        var targetUUID = node.getAttribute("id") + "-" + "LeftMiddle";
        instance.addEndpoint(node, targetEndpoint, { anchor: "LeftMiddle", uuid: targetUUID });

    }

}
```

添加连接线

```
        // 连接线
        $scope.connectPorts = function (instance, node1, port1, node2, port2, action) {

            // 链路条件样式
            overlays = [
["Arrow", { location: 0.8 }, { location: 1, foldback: 0.8 }],
["Label", {
    location: 0.1,
    label: "条件",
    id: "label",
    cssClass: "aLabel",
    events: {

        // 链路条件事件
        click: function (labelOverlay, originalEvent) {
            // 节点ID
            $scope.openContion(labelOverlay.component.sourceId, labelOverlay.component.targetId, action);
        }
    }
}]
```

删除连接线

```
    // 删除连接线
    $scope.deleConnect = function (c) {
        //清空连线
        instance.detach(c);
        //更新逻辑关系
        var sourceid = c.sourceId;
        var targerid = c.targetId;
        var parentid = $("#" + targerid).attr("parentId");
        if (parentid != undefined) {
            parentid = parentid.replace("|" + sourceid, "").replace(sourceid + "|", "");
            $("#" + targerid).attr("parentId", parentid);
        }

        // 节点支持拖拽属性
        instance.draggable($('.node'));


        // 删除集合中的链路
        for (var i = 0; i < _linkList.length; i++) {

            if (_linkList[i].Start_Node_Id == sourceid && _linkList[i].End_Node_Id == targerid) {
                _linkList.splice(i, 1);
                return;
            }
        }

    }
```

打开链路条件

```
// 打开链路条件
$scope.openContion = function (startNode, endNode, actionType) {

    // 根据节点ID找到节点信息
    var startNodeName = "";
    var endNodeName = "";
    for (var i = 0; i < _flowNodes.length; i++) {
        if (_flowNodes[i].Node_Id == startNode) {
            startNodeName = _flowNodes[i].Node_Name;
        }
        if (_flowNodes[i].Node_Id == endNode) {
            endNodeName = _flowNodes[i].Node_Name;
        }
    }

    // 找到选择链路，并将条件保存
    var conditionList = null;
    var conditionCount = 0;
    // 获取界面中链路信息
    for (var i = 0; i < _linkList.length; i++) {
        if (_linkList[i].Start_Node_Id == startNode && _linkList[i].End_Node_Id == endNode && _linkList[i].Executive_Action == actionType) {
            conditionList = _linkList[i].ConditionList;
            conditionCount = i;
        }
    }

    if (conditionList == null || conditionList.length < 1) {
        // 获取数据库中链路条件信息
        httpService.post("WorkFlow", "DesignerServices", "GetConditionByLinkId", { start_node_id: startNode, end_node_id: endNode, executive_action: actionType }).then(

        function (data) {
            conditionList = data;
```

```
                // 打开条件界面
                var modalInstance = $modal.open({
                    templateUrl: '../../templets/Designer/LinkCondition.html',
                    controller: 'LinkConditionCtrl',
                    size: 'lg',
                    backdrop: 'static',
                    resolve: {
                        models: function () {
                            return {
                                conditionList: conditionList,
                                startNodeName: startNodeName,
                                endNodeName: endNodeName,
                                startNodeId: startNode,
                                endNodeId: endNode,
                                actionType: actionType,
                                wfId: _flowInfo.Wf_Id,
                            };
                        }
                    }
                });

                modalInstance.result.then(
                //保存事件
                function (models) {

                    // 找到选择链路，并将条件保存
                    _linkList[conditionCount].ConditionList = models;
                },
                //取消事件
                function () { });
            },
            function (msg) {

                Notification.error({ message: "获取链路条件信息失败\r\n" + msg, delay: 6000 });
            }

            );
        }

    }
```

连接线结束后事件

```
// 连接线结束后事件
instance.bind("jsPlumbConnection", function (conn, originalEvent) {

    var status = 1;
    // 开始节点
    var startNode = conn.connection.sourceId;
    // 结束节点
    var endNode = conn.connection.targetId;

    // 验证
    var _isOk = true;
    if (startNode == 1 && endNode == 2) {

        Notification.error({ message: "开始节点不能直接连接结束节点！", delay: 6000 });
        // 删除连接线
        instance.detach(conn);
        return;
    }

    if (startNode == endNode) {

        Notification.error({ message: "请不要连接自己！", delay: 6000 });
        // 删除连接线
        instance.detach(conn);
        return;
    }
    // 获取状态出口
    // BottomRight 退回，TopRight 提交
    if (conn.sourceEndpoint.anchor.type == "BottomRight") {
        // 退回
        status = 2;
    }
    else
        // 提交
        status = 1;
```

```
        // 判断是否存在链路条件，如果不存在则添加
        if (conn.connection.getOverlay("label") == null) {
            conn.connection.addOverlay($scope.JsOverlay("where", status), "");
        }


        if (_linkList == null)
            _linkList = new Array();

        for (var i = 0; i < _linkList.length; i++) {

            // 规划的线路在原来链路中存在
            if (_linkList[i].Start_Node_Id == startNode && _linkList[i].End_Node_Id == endNode) {

                if (_linkList[i].Executive_Action == status) {

                    Notification.error({ message: "不能重复连接！", delay: 6000 });
                    // 删除连接线
                    instance.detach(conn);
                    return;

                } else {
                    // 存在链路没有状态出口
                    _linkList[i].Executive_Action = status;
                    return;
                }
            }
        }
        // 链路信息
        var obj = new Object();
        obj.Wf_Id = _flowInfo.Wf_Id;
        obj.Start_Node_Id = startNode;
        obj.End_Node_Id = endNode;
        obj.Sort = _linkList.length + 1;
        obj.Executive_Action = status;
        if (obj.Executive_Action == "") {
            obj.Executive_Action = 1;
        }
        // 放入集合中，并打开状态出口进行选择
        _linkList.push(obj);

    });
```

清空流程事件

```
    //  清空流程信息，从新打开
    $scope.cleanFlow = function () {

        // 删除端点

        for (var i = 0; i < _flowNodes.length; i++) {

            // 开始和结束节点不删除
            if (!(_flowNodes[i].Node_Id == 1 || _flowNodes[i].Node_Id == 2)) {
                // 删除端点
                var ac = ["-TopRight", "-BottomRight", "-LeftMiddle"];

                for (var j = 0; j < ac.length; j++) {

                    instance.deleteEndpoint(_flowNodes[i].Node_Id+ac[j]);
                }

                // 删除节点
                $("#" + _flowNodes[i].Node_Id).remove();
            }
        }
    }
```

获取节点的 XY 轴

```
    // 获取节点的XY轴
    $scope.GetNodeXY = function () {
        for (var i = 0; i < _flowNodes.length; i++) {
            // 节点信息
            var node = jsPlumb.getSelector('#' + _flowNodes[i].Node_Id)[0];

            if (node != null || node != undefined) {
                _flowNodes[i].X = node.currentStyle.left.replace("px", "");
                _flowNodes[i].Y = node.currentStyle.top.replace("px", "");
            }
        }
    }
```

后台保存流程方法

```
1 个引用
public MessageEntity SaveFlow(FlowModel model)
{
    if (model == null || model.BaseFlow == null || model.LinksList == null || model.NodesList == null)
        return MessageEntity.GetMessage(MessageEntity.ErrorType.SqlError);


    // 获取当前在用流程信息
    var cuurFlow = _baseFlowDal.GetBaseInfoList(Convert.ToDecimal(model.BaseFlow.Wf_Type));

    // 当前流程ID大于在用状态的流程ID时，说明是临时版本流程，直接删除链路重新录入即可
    if (cuurFlow != null && model.BaseFlow.Wf_Id >= cuurFlow.Wf_Id)
    {
        // 先删除链路
        DBManager.Execute("LinksWfIdDelete", new { wf_id = model.BaseFlow.Wf_Id });

        // 保存链路
        this.CreaLinkFlow(model.LinksList);

        // 修改节点的XY轴
        SetNodeXY(model.NodesList);
    }
    else
    {
        // 插入新的版本流程
        // 状态默认为不可用 彭文超 2015年11月2日 11:33:08
        model.BaseFlow.Status = 2;
        var _baseInfoMsg = this.CreateFlow(model.BaseFlow);

        // 节点
        var newList = this.InserNodes(_baseInfoMsg.WfId, model.NodesList);

        // 链路 、 直接保存
        this.CreaLinkFlow(model.LinksList);
       // this.InsertLinks(_baseInfoMsg.WfId, model.NodesList, newList, model.LinksList);

    }

    return MessageEntity.GetMessage(MessageEntity.ErrorType.Success);
}
```

后台提交流程方法

```
1 个引用
public MessageEntity SubmitFlow(decimal wfId)
{

    // 获取需要提交的流程信息
    var subFlow = _baseFlowDal.GetBaseInfoWfIdModel(wfId);

    // 获取当前在用流程信息
    var cuurFlow = _baseFlowDal.GetBaseInfoList(Convert.ToDecimal(subFlow.Wf_Type));

    // 6、只有可以进行保存操作的流程才可以进行提交操作
    // 4、当流程ID>正在运行的流程的ID时，流程可以进行保存操作，否则，任何流程不能进行保存操作
    if (cuurFlow != null && wfId < cuurFlow.Wf_Id)
        return MessageEntity.GetMessage(MessageEntity.ErrorType.SqlError, "当前流程早于在用流程，流程不可提交！");

    //subFlow.Crt_Date.CompareTo(cuurFlow.Crt_Date) == -1
    if (cuurFlow != null && cuurFlow.Wf_Id == wfId)
    {
        // 需事物执行
        //DBManager.ExecuteTransaction("", (args) =>
        //{
        // 创建新的版本流程
        // 将原来的流程全部置为不可用
        _baseFlowDal.UpdateStatus(Convert.ToDecimal(subFlow.Wf_Type));
        // 获取最新的版本信息
        subFlow.Status = 1;

        var msg = this.CreateFlow(subFlow);
        // 新流程ID
        subFlow.Wf_Id = msg.WfId;

        // 获取原始节点信息
        var oldNodesList = _workflowNodesDal.GetNodesWfId(wfId);

        // 创建新的节点信息
        var newNodesList = this.InserNodes(subFlow.Wf_Id, oldNodesList);
```

```
            // 创建新的流程信息，并且置为可用，注意链路上的节点信息
            // 链路
            // 获取原始链路信息
            var oldLinkList = _workflowLinksDal.QueryLinksByWfId(wfId);
            // 链路
            this.InsertLinks(subFlow.Wf_Id, oldNodesList, newNodesList, oldLinkList);


            //    return true;
            //});
        }
        else
        {

            //if (cuurFlow.Wf_Id == 0)
            //{
            //    return MessageEntity.GetMessage(MessageEntity.ErrorType.SqlError, "当前流程没有在用流程，请先保存，然后再提交！");
            //}

            // 需事物执行
            //DBManager.ExecuteTransaction("", (args) =>
            //{
            // 将原来的流程全部置为不可用
            _baseFlowDal.UpdateStatus(Convert.ToDecimal(subFlow.Wf_Type));

            // 将当前流程版本置为可用状态
            _baseFlowDal.UpdateWfIdStatus(wfId);

            //return true;
            //});
        }

        return MessageEntity.GetMessage(MessageEntity.ErrorType.Success);
    }
```

4.1.6 数据库（或者模型）设计

4.1.7 性能设计

4.1.8 限制和约束

1、任何时刻同一流程类型只有一个正在运行的流程

2、任何流程都可以打开查看

3、提交创建新的流程，此类型其他流程全部置为不可用状态

4、当流程 ID>正在运行的流程的 ID 时，流程可以进行保存操作，否则，任何流程不能进行保存操作

5、历史流程的从新启用问题，历史流程既然是历史，肯定是不能更改的

6、只有可以进行保存操作的流程才可以进行提交操作

4.2 工作流引擎

5 设计概述

<填写设计的概要，包括对各种所使用的设计方法的简要描述>
<如：本设计采用面向对象的方法对系统进行分析,系统使用 C/S 结构,并且使用 MVC 模型对系统建模>

5.1 设计约束

<包括
（1）需求约束。从需求文档（如《用户需求说明书》和《软件需求规格说明书》）中提取需求约束，例如：
本系统应当遵循的标准或规范
软件、硬件环境（包括运行环境和开发环境）的约束
外部接口/协议的约束
用户界面的约束
软件质量的约束，如正确性、健壮性、可靠性、效率（性能）、易用性、清晰性、安全性、可扩展性、兼容性、可移植性等等。
（2）隐含约束。有一些假设或依赖并没有在需求文档中明确指出，但可能会对系统设计产生影响，应当尽可能地在此处说明。例如对用户教育程度、计算机技能的一些假设或依赖，对支撑本系统的软件硬件的假设或依赖等。>

5.2 设计策略

<根据产品的需求与发展战略，确定设计策略（Design Strategy）。例如：

- 扩展策略。说明为了方便本系统在将来扩展功能，现在有什么措施。
- 复用策略。说明本系统在当前以及将来的复用策略。
- 折衷策略。说明当两个目标难以同时优化时如何折衷，例如“时－空”效率折衷，复杂性与实用性折衷。>

5.3 技术实现

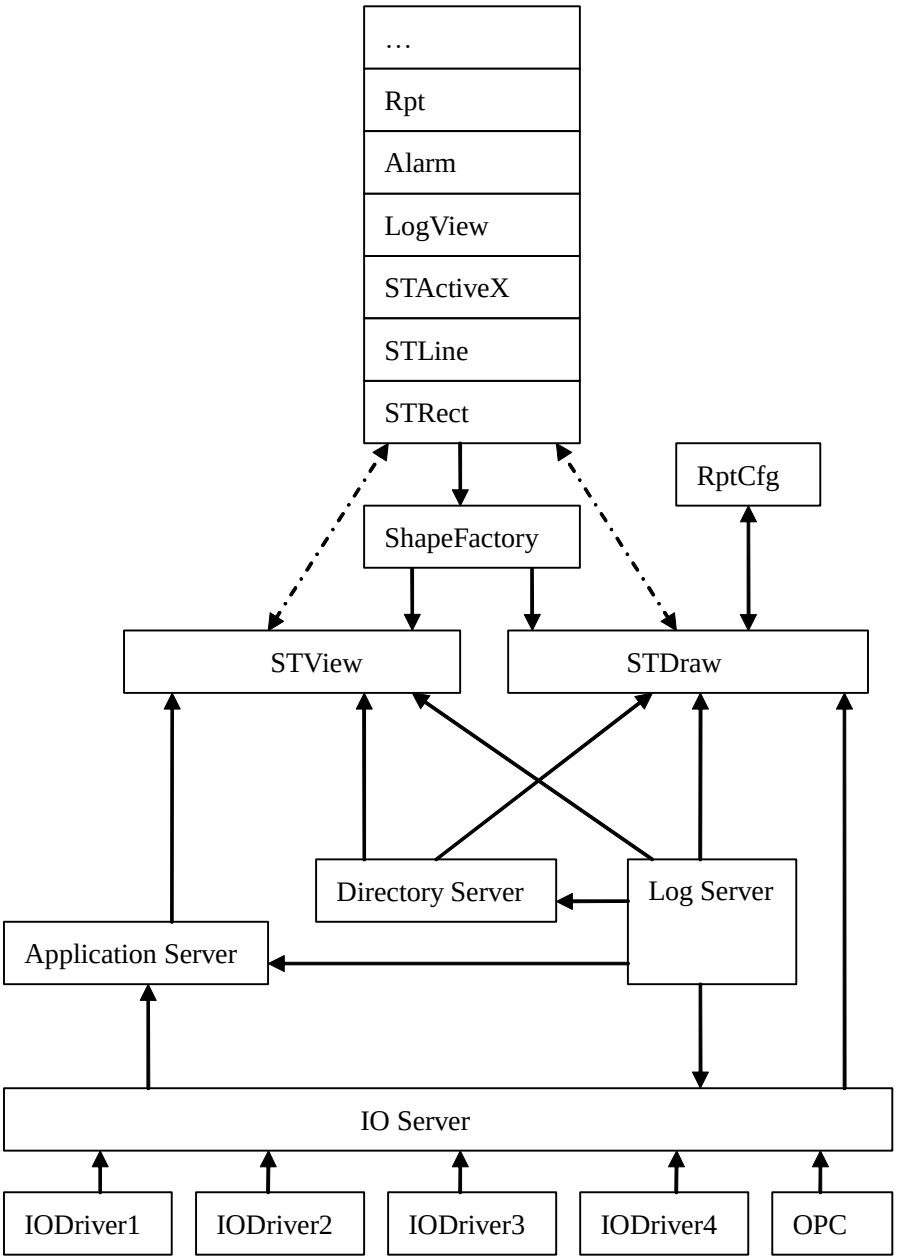
<本系统所采用的技术以及该技术的说明>
<如：系统使用 XML 技术来增强系统配置的灵活性,并且 xml 同时亦可以作为与外部进行数据交换的标准格式.在图形表现层上系统使用 WPF 技术,WPF 技术是 windows 系统在 XP 之后的系统标准的图形接口,,它利用 D3D10 技术提供了丰富的表现效果以及令人满意的速度.>

6 系统总体框架

6.1 网络结构图

6.2 组件结构图

〈这部分要求提供高层系统结构的描述，使用方框图来显示主要的组件的交互.〉
〈如



在这个图中，每个箭头都代表着一个接口的使用，从接口的提供者到接口的使用者。应该在这里描述系统中所有出现的接口调用，如：

提供者	调用者	接口名称
IODriver	IOServer	IIODriver
IODriver	IOServer	IIODevice
LogServer	ApplicationServer,STView,STDraw,…	ILog
…	…	…

>

6.3 外部接口

<这里描述系统对外提供或要求外部提供的接口>

<如:

Application Server 需要通过实现 OPC 接口向外暴露数据;

LogServer 需要对外提供 SQL 查询接口

对于一些大家熟知的接口,可以简单列举出接口名称即可,如果是自定义的接口,则应该在此处详细地写明接口的规格。

接口，是一组相关方法的组合。在接口的规格说明中，要说明接口中各个方法的顺序、名称、参数、返回值类型等；对于每个参数，也应该说明其类型、名称、含义以及是输入还是输出等。对于每个接口，都应该按如下格式描述：

如：

名称	功能	提供者
ISTIODriverServer	操作 IO 服务器	IOServer
ISTDirecytoryServer	操作目录服务器	DirectoryServer
ILogServer	操作日志服务器	LogServer
ISTApplicationServer	操作 Application 服务器	ApplicationServer
IXMLDOMDocument	操作 X M L 文档	外部（MSXML3.DLL）
ISTShape	操作图形对象	STxxx

6.4 组件汇总表

<以列表的方式写出系统的各个组成部分(子系统),该表的列可以根据具体的情况来设置,但一般需要包含名称、功能、实现文件、是否为重要组件四列,

如:

名称	功能	实现文件	是否为重要组件
STDraw	系统组态	STDraw.exe	是
STView	系统监控	STView.exe	是
Application Server	系统实时数据库	ApplicationSvr.exe	是
Log Server	日志服务器	LogSvr.exe	否
Directory Server	组态服务器	DirectorySvr.exe	是
IO Server	IO 管理服务器	IOServer.exe	否
Rpt	报表浏览/打印	ReportView.ocx	否
STxxx	各种图形对象	STxxx.dll	是

>

7 系统组件

<这一部分分别描述 4.3 表中的各个组件, 每个组件作为一个小节>

7.1 组件 1 <如: STDraw 组件>

7.1.1 功能描述

<这里概括地描述该组件的主要功能
如: STDraw 组件完成了系统的组态功能。它在一个集成的界面里给用户提供了画面组态、位号组态、设备组态、报表组态、用户安全设置等系统配置功能。

7.1.2 关键技术

<这里说明该组件中使用到的一些关键的技术。
如:
1. STDraw 使用了 XML 文件来进行系统配置;
1. 使用 VBS 宏脚本来对程序界面、组态逻辑功能、菜单、按钮等进行自定义;
2. 使用了 OGG 作为程序的绘制引擎;
3. 支持 ActiveX 对象插入;
>

7.1.3 提供的接口

<说明该组件向外提供的接口,这个接口一般是由外部该组件的使用者调用的。在这里,接口的描述可以是概括性的,但一般要求定义出接口的名称及大概的功能。从系统框架结构图来看,基本上从一个组件出发的一个箭头就代表着一次外部组件对该组件所提供的某个接口的使用。可以使用表格的方式来描述组件提供的接口, 如:

接口名称	大概功能
ISTDrawApplication	用来表示组态程序进程本身的对象，它可以提供关于组态程序进程的一些信息；提供一种组态时在全局层面上执行的方法。可能包含的属性有：Visible、ActiveDocument 等；可能包含的方法有：LoadDrawing、NewDrawing、GetDrawing、Close 等。
ISTDrawDocumentPic	用来表示在组态程序进程中正在组态的一幅流程图文档对象。它提供了访问和操作正在组态中的一幅流程图的方法。它可能包含的属性有：Shapes、ShapeCount 等；可能包含的方法有：

	AddShape、Clear、Remove、IndexOf 等。
ISTDrawDocumentTags	用来表示在组态程序进程中位号组态对象。它提供了访问和操作位号组态的方法。它可能包含的属性有：Tags、TagCount 等；可能包含的方法有：AddTag、Clear、Remove、IndexOf、Find 等。

>

7.1.4 需要的接口

<描述该组件需要使用到的外部的接口，该接口由其它组件（或系统外部）提供。如果使用的接口是众所周知的，可以仅仅列出接口的名称；如果该接口是由系统内部其它组件提供的，应该指出提供该接口的组件名称。如果可能，应该使用超链接的方式把接口链接到相应的组件说明。从系统框架结构图上看，指向一个组件的箭头往往意味着该组件对外部接口的使用，所以通过分析系统框架结构图应该容易地总结出组件使用外部接口的情况。组件使用外部接口的情况也可以用表格来表示，由于接口的功能在接口提供者组件的部分有比较详细的说明，这里的“功能”列可以写的非常简单。

如：

名称	功能	提供者
ISTIODriverServer	操作 IO 服务器	IOServer
ISTDirecytoryServer	操作目录服务器	DirectoryServer
ILogServer	操作日志服务器	LogServer
ISTApplicationServer	操作 Application 服务器	ApplicationServer
IXMLDOMDocument	操作 X M L 文档	外部（MSXML3.DLL）
ISTShape	操作图形对象	STxxx

>

7.2 组件 N

<同 5 .1>

7.2.1 功能描述

<同 5 .1.1>

7.2.2 关键技术

<同 5 .1.2>

7.2.3 提供的接口

<同 5 .1.3>

7.2.4 需要的接口

<同 5 .1.4>

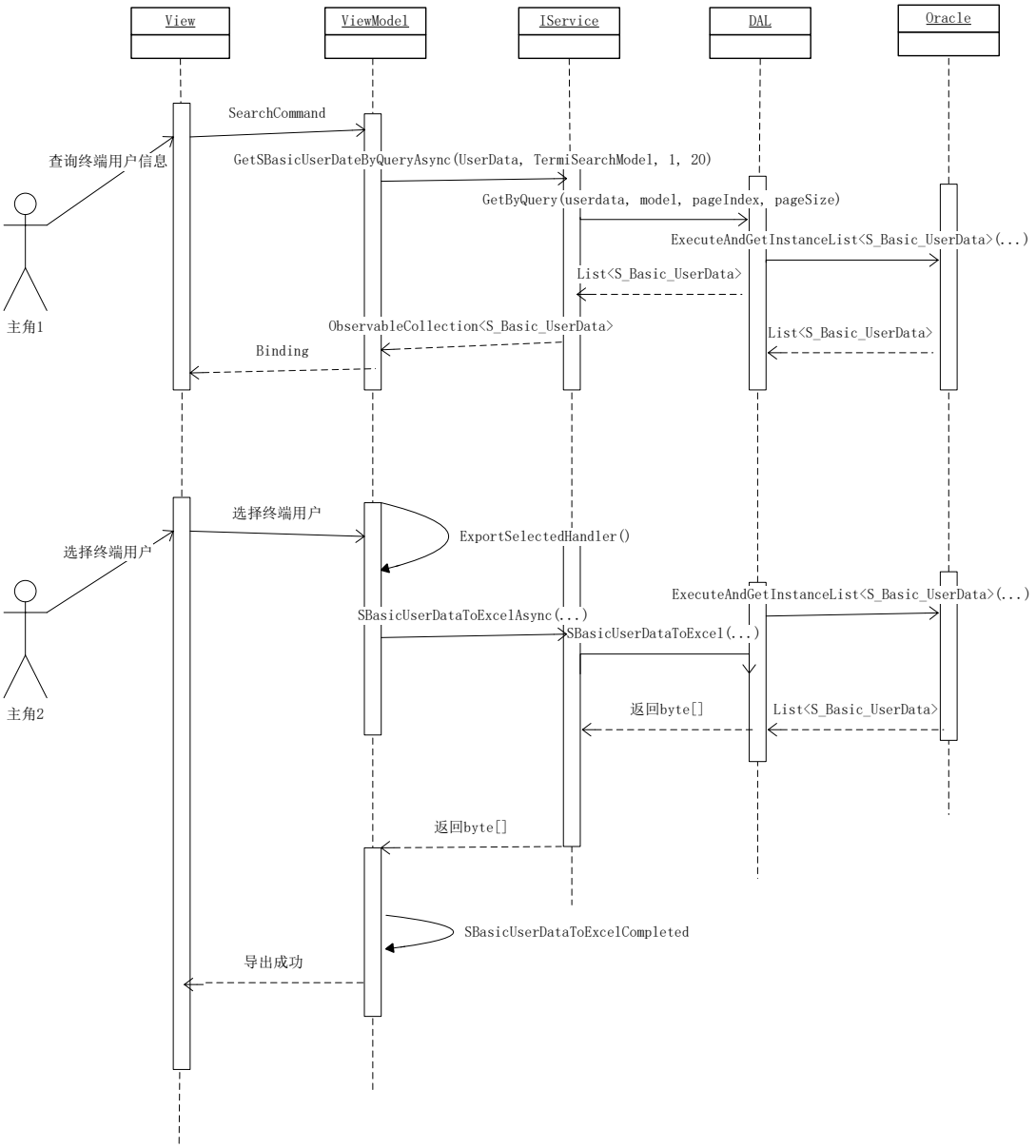
8 公众版数据分析设计

8.1 终端用户查询

8.1.1 业务说明

该模块主要描述：由手机终端软件采集终端数据，经过解析，入库，在界面上展示终端用户数据。并可以根据用户账号，手机号，IMEI 号查询，可以根据用户选择导出终端数据。

8.1.2 逻辑设计



终端库查询时序图

8.1.3 处理设计(或算法)

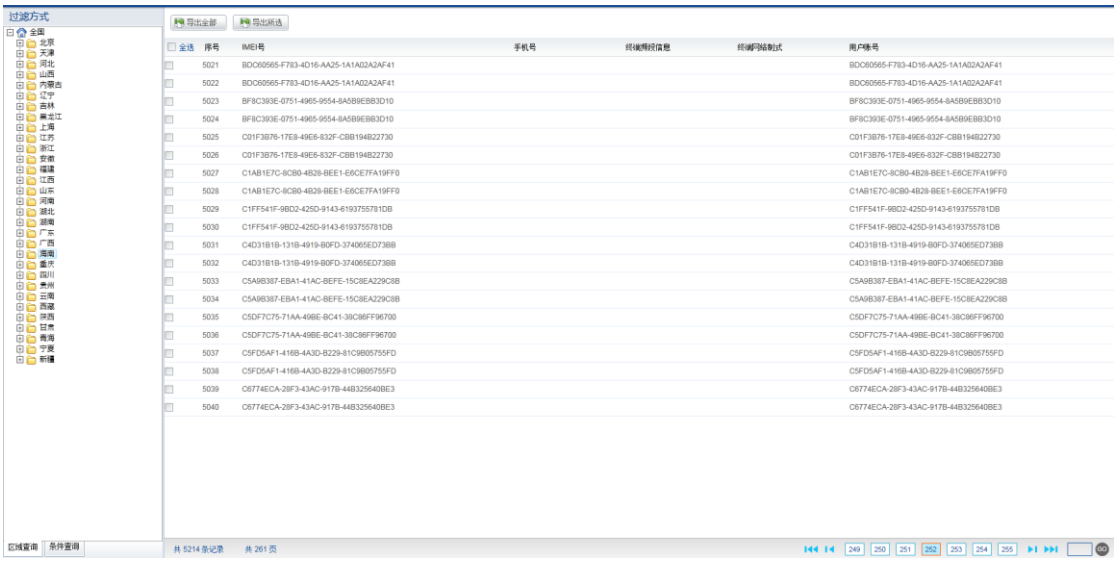
错误处理：写入操作日志[Logger("模块名称", "操作模块", "操作行为", "调用方法")]
[Logger("公众版数据管理", "终端库管理", "终端库数据查询", "GetSBasicUserDateByQuery")]
[Logger("公众版数据管理", "终端库管理", "终端库数据查询", "GetSBasicUserDateCount")]
[Logger("公众版数据管理", "终端库管理", "终端库数据导出", "GetSBasicUserDateByQuery")]

8.1.4 界面设计

1. 菜单设计：

2. 对话框设计：

3. 终端库查询主页设计：



8.1.5 程序说明

公用方法在时序图中已说明。在服务端，配置脚本如下：

```
<alias alias="IS_Basic_UserDal" type="CoolTest.Online.ExpNomal.IDAL.IS_Basic_UserDal,CoolTest.Online.ExpNomal.IDAL" />
<alias alias="S_Basic_UserDal"  type="CoolTest.Online.ExpNomal.DAL.S_Basic_UserDal,CoolTest.Online.ExpNomal.DAL" />
<alias alias="IS_Basic_UserBLL" type="CoolTest.Online.ExpNomal.IBusiness.IS_Basic_UserBLL,CoolTest.Online.ExpNomal.IBusiness" />
<alias alias="S_Basic_UserBLL" type="CoolTest.Online.ExpNomal.Business.S_Basic_UserBLL,CoolTest.Online.ExpNomal.Business" />
```

```
<register type="IS_Basic_UserBLL" mapTo="S_Basic_UserBLL" />
<register type="IS_Basic_UserDal" mapTo="S_Basic_UserDal" />
```

该配置描述服务定义及服务接口映射

8.1.6 数据库(或者模型)设计

表名称 S_BASIC_USER　　用途 用于终端库数据保存

字段名称	标识	类型	长度	格式	值域	能否为空	是否为主键	是否为外键	备注
用户唯一标识	USER_AUID	VARCHAR2(32)				N			
用户类型	USER_TYPE	INTEGER				Y			
用户编号	USER_ID	VARCHAR2(50)				Y			
用户密码	USER_PASSWD	VARCHAR2(50)				Y			
省编号	PROV_ID	INTEGER				Y			
市编号	CITY_ID	INTEGER				Y			
区域编号	AREA_ID	INTEGER				Y			
是否在线	IS_OL	INTEGER				Y			
最后登录时间	LAST_LOGIN_DATE	INTEGER				Y			
积分	INTEGRAL	INTEGER				Y			
真名	REAL_NAME	VARCHAR2(30)				Y			
手机号	MOBILE_NO	VARCHAR2(16)				Y			
电子邮件	MAIL	VARCHAR2(50)				Y			

是否有效	IS_EFF	INTEGER				Y			
创建时间	CRT_DATE	INTEGER				Y			
修改时间	MODIFY_DATE	VARCHAR2(50)				Y			
修改用户唯一标识	MODIIFY_USER_AUID	VARCHAR2(50)				Y			
修改用户编号	MODIFY_USER_ID	VARCHAR2(50)				Y			
备注	REMARK	VARCHAR2(100)				Y			
内外用户	IN_OUT_USER	INTEGER				Y			
地址	ADDR	VARCHAR2(100)				Y			
IMEI	IMEI	VARCHAR2(50)				Y			
等级	LVL	INTEGER				Y			
激活码	ACTIV_CD	INTEGER				Y			
终端型号	MOB_MODEL	VARCHAR2(32)				Y			
日志数	LOG_CNT	INTEGER				Y			
异常日志数	EXP_LOG_CNT	INTEGER				Y			
行为日志数	ACTION_LOG_CNT	INTEGER				Y			
性别	SEX	INTEGER				Y			
年龄	AGE	INTEGER				Y			
生日	BIRTHDAY	INTEGER				Y			
版本	VSN	VARCHAR2(20)				Y			
MAC 地址	MAC	VARCHAR2(50)				Y			
剩余积分	LEFT_INTEGRAL	INTEGER				Y			
更新时间	UPD_DATE	DATE				Y			
供应商	VENDER	VARCHAR2(30)				Y			
纬度	LATI	NUMBER				Y			
经度	LONGI	NUMBER				Y			
上次等级	LAST_LVL	INTEGER				Y			
上次等级更新时间	LAST_LVL_DATE	DATE				Y			
最后上传 LOG 时间	LAST_LOG_DATE	DATE				Y			
主动测速 LOG 数	INIT_LOG_CNT	INTEGER				Y			
运营商编号	CARRIER_ID	INTEGER				Y			

表名称 I_BASIC_USER_EXPAND 用途 用户扩展表

字段名称	标识	类型	长度	格式	值域	能否为空	是否为主键	是否为外键	备注
用户编号	USER_ID	VARCHAR2(120)				N			
终端制式名称	MOB_NET_TYPE_NAME	VARCHAR2(60)				N			
终端最高制式	MOB_MAX_NET_TYPE	INTEGER				N			
终端最高制式名称	MOB_MAX_NET_TYPE_NAME	VARCHAR2(30)				N			
是否普通用户	IS_BASIC_USER	INTEGER				N			
是否 VIP 用户	IS_VIP_USER	INTEGER				N			
是否集团用户	IS_BLOC_USER	INTEGER				N			

是否高产出用户	IS_HIGH_PROC_USER	INTEGER				N			
是否高异常用户	IS_HIGH_EXP_USER	INTEGER				N			
更新时间	UPD_DATE DATE	SYSDATE				N			

8.1.7 性能设计

终端库查询，考虑到数据量大的问题，按分页查询。

8.1.8 限制和约束

用户账号，手机号码，用户 IMEI 号模糊查询

9 测试要点

测试要求：

1. 同一程序内的代码书写是否为同一风格
2. 程序中函数、子程序块分界是否明显
3. 注释是否符合既定格式
4. 注释是否正确反映代码的功能
5. 变量定义是否正确（长度、类型、存储类型）
6. 变量的命名是否相似
7. 是否存在声明过，但从未引用或者只引用过一次的变量
8. 全局变量定义和用法在各个模块中是否一致
9. 常量是否被做为形式参数进行传递
10. 测试数据是否具有一定的代表性
11. 测试数据是否包含测试所用的各个等价类（边界条件、次边界条件、空白、无效）
12. 每一组测试数据的测试结果是否与预期结果一致
13. 产生输入/输出错误时，系统是否进行检测并处理
14. 输出信息中是否存在文字书写错误和语法错误
15. 控件布局是否合理、美观
16. 异常信息表述是否正确
17. 软件是否按预期方式处理错误
18. 画面文字（全、半角、格式、拼写）是否正确
19. 产生的文件和数据表的格式是否正确
20. 错误日志的表述是否正确
21. 是否严格按照需求原型实现界面。
22. 是否满足需求，实现功能。

附录 1 模块版本描述

模块名称	标识	可运行版本	依赖模块及版本
终端库查询	TerminalMangement	V4.0.2.39	无
重点用户感知评估	KeyUserAssessmentView	V4.0.2.39	无
异常事件分析	ExceptionEventAnalyzePage	V4.0.2.39	无
用户异常事件查询	AbnormalEventsManagement	V4.0.2.39	无
覆盖评估	CoverageEvaluatePage	V4.0.2.39	无
业务指标体系配置	BusinessInSysManagement	V4.0.2.39	无
网络与终端户匹配	NetworkMatchTerminalPage	V4.0.2.39	无
区域网络质量分析	AreaQualityAnalysisPage	V4.0.2.39	无
小区异常事件查询	CellEventsManagement	V4.0.2.39	无
应用行为分布及质量分析	AppActionAnalysisView	V4.0.2.39	无