前端框架思路文档

[一、 名词解释 1](#_Toc99876772)

[（一） 用户（user） 1](#_Toc99876773)

[（二） 角色、节点、来源、信息类型 1](#_Toc99876774)

[（三） 页面（page）、表单（form）、列表（list） 2](#_Toc99876775)

[二、 生命周期 2](#_Toc99876776)

[（一） 配置数据说明 2](#_Toc99876777)

[（二） 当前页面生命周期 8](#_Toc99876778)

[（三） 当前页面跳转 14](#_Toc99876779)

[三、 前端权限管理 14](#_Toc99876780)

[（一） 权限分类： 14](#_Toc99876781)

[（二） 前提： 15](#_Toc99876782)

[（三） 页面权限 15](#_Toc99876783)

[（四） form权限 15](#_Toc99876784)

[（五） list权限 15](#_Toc99876785)

[（六） 总结： 16](#_Toc99876786)

[四、 页面逻辑函数 16](#_Toc99876787)

[（一） dialog函数调用 16](#_Toc99876788)

[（二） form可调用函数 17](#_Toc99876789)

[（三） list列表可调用函数 19](#_Toc99876790)

[五、 不同页面解决方案示例 21](#_Toc99876791)

[（一） 登录 21](#_Toc99876792)

[（二） 登录成功 21](#_Toc99876793)

[（三） 注销 21](#_Toc99876794)

[（四） 顶部菜单渲染 21](#_Toc99876795)

[（五） 左侧菜单渲染 21](#_Toc99876796)

[（六） 修改密码 21](#_Toc99876797)

[（七） 列表页 21](#_Toc99876798)

[（八） 待办 21](#_Toc99876799)

[（九） 项目管理页面 22](#_Toc99876800)

[（十） 文件上传 22](#_Toc99876801)

[（十一） 文件下载 22](#_Toc99876802)

[（十二） 导入 22](#_Toc99876803)

[（十三） 导出 22](#_Toc99876804)

[（十四） 搜索 22](#_Toc99876805)

[（十五） 机构\部门人员数据添加 22](#_Toc99876806)

[（十六） 添加部门人员信息 22](#_Toc99876807)

[（十七） 机构\部门角色数据添加 22](#_Toc99876808)

[（十八） 机构\部门人员数据添加--带角色 22](#_Toc99876809)

[（十九） 菜单渲染 23](#_Toc99876810)

[（二十） 库存入库解决方案 23](#_Toc99876811)

[（二十一） 树形结构、层级结构渲染--点击渲染子数据 23](#_Toc99876812)

[（二十二） 树形结构、层级结构渲染--全量渲染 24](#_Toc99876813)

[（二十三） 数据迁移 25](#_Toc99876814)

[（二十四） 角色权限配置 26](#_Toc99876815)

[（二十五） 角色公共弹窗 27](#_Toc99876816)

[（二十六） 人员管理 28](#_Toc99876817)

[（二十七） 商城首页商品列表渲染解决方案 28](#_Toc99876818)

[（二十八） 购买逻辑 28](#_Toc99876819)

[（二十九） 用户画像 29](#_Toc99876820)

[（三十） 列表动态设置 29](#_Toc99876821)

[（三十一） 列表动态排序 29](#_Toc99876822)

[（三十二） 富文本编辑器插件 29](#_Toc99876823)

[（三十三） 已读、未读信息列表 29](#_Toc99876824)

[（三十四） 中英文切换 29](#_Toc99876825)

[（三十五） 工作流启动 29](#_Toc99876826)

[（三十六） 完成任务 29](#_Toc99876827)

[（三十七） 历史审批任务 29](#_Toc99876828)

[（三十八） 审批列表跳转详情页 29](#_Toc99876829)

[（三十九） 判断信息是否在审批中 30](#_Toc99876830)

[（四十） 表单打印预览+水印 30](#_Toc99876831)

[（四十一） 流程图 30](#_Toc99876832)

[（四十二） 任务列表信息处理函数逻辑 31](#_Toc99876833)

[（四十三） 搜索框历史记录--类似百度 31](#_Toc99876834)

[（四十四） 发送邮件 31](#_Toc99876835)

[（四十五） 文件预览 31](#_Toc99876836)

[（四十六） 文件在线编辑 31](#_Toc99876837)

[（四十七） 表格标题行排序 31](#_Toc99876838)

[六、 附件：草稿箱解决方案 31](#_Toc99876839)

[（一） 草稿箱表中字段 31](#_Toc99876840)

[（二） 总结 31](#_Toc99876841)

# 名词解释

每一个页面有一个配置文件，配置文件的命名格式：页面名称\_DC.js。里面具体的示例见dataCenter.js，下面我们结合dataCenter.js示例，对不同的概念分解，不同的示例剖析。

## dataCenter（数据中心）

页面看到、用到、需要处理的所有数据都会在dataCenter中体现。可以理解为前端框架的window对象。

## 用户（user）

当前登录用户的相关信息，这部分数据不用配置，登录接口会返回用户相关信息，登录方法会保存用户信息。

## 页面进入方式（pageEntry）

回退、恢复、跳转进入。

## 信息类型（state）：

主要用于区分信息类型(state)：草稿=draft，纯信息=info，修改记录=modifyRecord，归档=archive，启动流程=startProcess，完成任务=task

备注：这里的state与接口参数中info.type是对应的

## 页面权限

### 节点（pageNode）:

当前页面所处节点名称。

### 来源（fromNode）:

当前页面的状态是从哪一个状态跳转过来，fromNode记录了来源状态。

### 角色（role）：

登录时用户的所有角色信息，会保存在dataCenter.user和浏览器的sessionStorage中。由于用户角色和管理端角色的定义有区别。

管理端角色分为：

* 系统角色（系统角色一般为后端管理员角色，前端无法修改删除，默认只获取当前登录机构下的系统角色）
* 数据角色（orgId匹配机构部门id，默认只获取当前登录机构下的机构角色）

用户角色：

* 系统角色（提前预设好的，不能变动的角色）
* 信息角色（角色的orgId匹配用户的机构部门id）

备注：登录时后端会将用户的的所有角色都返回。角色可能是部门角色，角色的机构id和部门id是否都记录、返回。用户的机构部门id后端是否返回。信息可能属于部门，信息的orgid记录什么？信息所属机构和部门是否后端返回了。

* 项目角色 (当前数据的id匹配角色的orgId)
* 任务角色 (当前数据的id或者pid匹配角色的orgId，或者pids包含角色的orgId)

备注：前端登录时会返回当前用户当前机构下所有的角色，这些角色都会被记录保存。进入到具体信息详情的时候，才需要区分角色类型，匹配对应角色类型。

* 数据角色 (人员id在信息的数据角色集合字段中)

备注：数据角色信息中dataField字段存：信息的数据角色集合字段

### 权限分类：

用户端的权限和管理员端的权限有所不同，本文中要是没有特指，权限指的是用户端权限。管理员端权限在在管理员端定义，这里不累述；用户端的权限主要指当权限不同时，页面属性（显示、隐藏、只读、必填、选填等）不同。这个权限可以在页面dateCenter中配置。实际上页面属性由‘pageNode’、‘fromNode’、用户角色三者共同决定。

在管理端时，信息只有在栏目下才能是某个概念。同一个信息可以存在于不同栏目，在哪个栏目获取/展示，信息就是这个栏目对应的概念。而在用户端时，一个页面展示同一个信息在不同栏目的内容，需要确定一个主栏目。例如：项目是一条项目信息，存在于项目栏目中。同时项目也是项目管理部下的一个项目部门存在于部门栏目。在项目栏目中对项目信息可设置角色权限，在部门栏目中对项目部门可设置角色权限。后端将这些角色权限都记录为用户角色，前端在项目页获取用户角色后，将用户角色进行了细分。公司项目管理员可以修改所有的项目信息，所以将公司项目管理员这个用户角色定义为信息角色。当前项目的项目经理可以修改当前项目信息，所以项目经理这个角色定义为项目角色。

#### 功能权限：

角色不同，页面显示不同。

#### 信息权限：

数据不同，页面显示不同。

### 总结

根据pageNode，fromNode这两个参数对页面进行细分，形成页面状态。pageNode将页面分成多个节点， fromNode记录当前页面来源。根据fromNode参数，实际可以将页面细分成更小的状态。由于状态更细，就可以根据状态的不同，更方便对页面进行管理，比如管理权限。

同一页面，不同的角色（role）、节点（pageNode）、来源（fromNode）不一样时，显示效果是不同的。

角色数据会从用户（user）中获取，节点（pageNode）、来源（fromNode）会从页面中获取。注意：来源（fromNode）与节点（pageNode）是影响整个页面。我们也可以这样理解，当页面显示不一样时，一定是角色（role）、节点（pageNode）、来源（fromNode）有区别。

## 页面（page）、表单（form）、列表（list）

一个页面(page)中包含表单（form）和列表（list），一个页面可以包含多个form、多个list，form与list可以相互嵌套。

原则：一个页面中只有一个mainForm或一个mainList，不能出现多个mainForm，也不能出现多个mainList，mainForm与mainList也不能同时存在。

当一个页面出现mainForm时，系统会认为这个页面是form页面，会调用form的初始化方法。

当一个页面出现mainList时，系统会认为这个页面是list页面，会调用list的初始化方法。

## 当前页面

浏览器正在运行的页面就是“当前页面”，前面讨论的页面（page）就是“当前页面”。当前页面由前端html、页面配置信息、页面数据组成。这些信息相互组合，最后形成客户看到的页面功能。

## 上一页面和下一页面

实际上前端框架没有“上一页面”与“下一页面”这两个概念，只有“当前页面”这个概念。如果由“当前页面”回退到“上一页面”，跳转完成后“上一页面”就会成为“当前页面”。如果由“当前页面”跳入“下一页面”，跳转完成后“下一页面”就会成为“当前页面”。

# 配置数据说明

## dataCenter：数据中心

### page

page:{

pageUrl: '',//页面完整url

pageName: '',//页面名称（页面html文件名称，不含“.html”后缀）

pageNode: '1',//当前页面所处节点名称，0、1、2、3等

fromNode: 'normal',//当前页面的状态是从哪一个状态跳转过来，fromNode记录了来源

~~classId : '',//页面主栏目id~~

~~infoId: '',//页面主信息id~~

~~zj\_pagePermission\_update:true,//主信息编辑权限~~

~~zj\_pagePermission\_add:true,//主信息新增权限~~

~~zj\_pagePermission\_delete:true//主信息删除权限~~

},

menu: {}, //页面菜单数据

org: {

orgId: '',//组织id

orgName: '',//组织名称

},//组织信息

roles: {},//角色信息

list: {},//list相关配置

form: {},//form相关配置

user: {

roles: [ //进页面分配，出页面删除

{

roleId: '', //角色id

roleName: '',//角色名称,

orgId: '',//组织id

orgName: '',//组织名称

roleType: '',

//角色类型 角色类型:1=系统角色，2=用户角色，3=项目角色，4=任务角色

},

],

}

showBlock:[ //页面全局显示设置

allRole: { //角色

'1': {//pageNode

'normal': [//formNode

], //显示模块的id

}

}

] ,

hiddenBlock:[ //页面全局隐藏设置

allRole: { //角色

'1': {//pageNode

'normal': [//formNode

], //隐藏模块的id

}

}

] ,

};

### list

dataCenter["list"][listId] = {

//列表classId

classId: "",

//是否需要模版

isNeedTemp:false,

//需要获取的字段名，英文逗号分隔

fields:"",

//列表数据类型

state:"",

//是否有子集 默认不传 1=需要

isNeedSubset:0,

//是否查询缓存 1=不查缓存

noCache:0,

//查询未删除、已删除

isdelete:0,//0正常查、1只查删除的、2都查、默认0

//是否查询ES

isEs:0,//默认为0，1=查询ES

//ES查询的索引，多个英文逗号隔开

index:"",

//ES高亮查询-fields-高亮字段 pretags-高亮css形式

\_ES\_highlight:{"fields":["title"],"pretags":"<font color= 'red'>","posttags":"</font>"},

//查询已读、未读

isOrNotRead : "",//默认为空，all=全部，read=已读列表，unread=未读列表 返回值中zj\_readstatus = 0未读，1=已读

//分页组件，每页条数

eachPageNum:[10,20,50],

//分页形式

pageingType:1, //分页类型 1=正常分页，2=本地全量数据分页。3=本地id集合分页（本地有全量数据但是字段不全，缺少列表要用的字段） 4=ES分页

page: { //分页设置

pageSize: 10, //每页条数

currPage: 1 //当前页

},

searchInfo: { //查询参数

orderType: "",

searchValue: [""],

searchField: [""],

searchCondition: [""]

},

groupBy:{ //聚合查询，默认注释，不存在

"groupFiled": "",

"infoclass":"",

"showFileds":[{

"fname":"",//显示字段

"fun":"",//聚合函数名

"asName":""//字段别名

}],

},

//自定义查询条件

getParamFun: "",

//信息列表 页面dataCenter可以不写

listInfos: [],

//一般情况下 table---false row----true

isRow: false,

isPersist: true, //页面json---false 持久化 -----true

deleteIds: [],

//list searchRole有关，通过name找到input，再通过listSearchParent规则找到父级隐藏（默认在sysSet中设置，list设置>sysSet设置）。

listSearchParent:".parent().find('span label')",

//分页插件样式 目前只有2

paginationType: 2,

//存储表头模板

temphead: '',

//存储表每行模板

temprow: '',

//分页记住上一页选择结果

reCheck:true, //true=记录

event:{

"beforeRender":{

'all':{//全局配置

'all':"",//全局配置

},

'1':{//pageNode

'add':"",//fromNode

},

},

"afterRender":{

'all':{//全局配置

'all':"",//全局配置

},

'1':{//pageNode

'add':"",//fromNode

},

},

"beforeDelete":{

'all':{//全局配置

'all':"",//全局配置

},

'1':{//pageNode

'add':"",//fromNode

},

},

"afterDelete":{

'all':{//全局配置

'all':"",//全局配置

},

'1':{//pageNode

'add':"",//fromNode

},

},

},

/\*控制列表的列显示 \*/

show: {

/\*

有的就隐藏

通过input的name控制input隐藏 再通过dataCenter[sysSet]的listSearchhidden规则找到父级隐藏

\*/

searchRole: {

allRole: {

//pageNode

'1': {

//fromNode

'manager': ['number', 'name', 'note', 'title'], //input的name值

'2': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

'2': {

'1': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

'4': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

},

role1: {

'1': {

'manager': ['number', 'name', 'note', 'title'],

'2': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

'2': {

'1': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

'4': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

},

role2: {

'1': {

'manager': ['number', 'name', 'note', 'title'],

'2': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

'2': {

'1': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

'4': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

},

},

listRole: {

allRole: {

'1': {

'index': {

thead: ['name', 'note'],//删除列。有的就删掉

row:[0,1,4,6], //删除行，有的就删掉

},//有的就删掉,

'2': {

thead: ['name', 'note'],//有的就删掉

row:[0,1,4,6],

}

},

'2': {

'index': {

thead: ['name', 'note'],//有的就删掉

row:[0,1,4,6],

},//有的就删掉,

'2': {

thead: ['name', 'note'],//有的就删掉

row:[0,1,4,6],

},

},

},

role2: {},

'allRole+role2':{}

},

//showBlock不是控制input，是根据id控制哪些部分需要显示

showBlock: {/\*控制块的显示 \*/

allRole: {

'1': {

'manager': ['caiwuDHreportmaterialname', 'caiwuDHreportmaterial'],

'2': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

},

},

hiddenBlock: {/\*控制块的隐藏 \*/

allRole: {

'1': {

'manager': ['caiwuDHreportmaterialname', 'caiwuDHreportmaterial'],

'2': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

'2': {

'1': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

'4': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

},//哪些块被显示

role1: {

'1': {

'manager': ['caiwuDHreportmaterialname', 'caiwuDHreportmaterial'],

'2': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

},

'2': {

'1': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

'4': ['caiwuDHmeetingminutesname', 'caiwuDHmeetingminutes'],

}

},//哪些块被显示

},

field\_showBlock: {

allRole: {

'1': {

'normal': {

'status': {

'有效': ['zjMark\_classificationManagementList\_status'],

}

}

},

},

},

field\_hiddenBlock: {

allRole: {

'1': {

'normal': {

'status': {

'有效': ['zjMark\_classificationManagementList\_status'],

}

}

},

},

},

},

};

### form

dataCenter["form"][formId]={

classid:"", //栏目id

infoId: '', //信息id

draftInfoId: '',//草稿箱id，进入页面先判断draftInfoId是否传过来，是就去查草稿，没有就判断infoId

state: 'info',//信息提交类型：纯信息=info，修改记录=modifyRecord，归档=archive，启动流程=startProcess，完成任务=task

isNeedTemp:false,//需要模版

fileds:"",//需要获取的字段名，英文逗号分隔

projectorg: "0",//是否创建项目部门 0=不创建，1=创建

    isAwaitPostPosition:0,// 提交接口后端是否是同步处理逻辑,默认为0，1=后端新增修改接口同步

isEs:0,//默认为0，1=接口同步更新ES

down: {

/\*从服务器上下载的数据，有两组数据，一般前端在修改数据时只修改formNewInfo，

formInfo可以保留，不修改，用来判断前端页面是否对数据做了修改

\*/

info:{},//原始数据

formInfo: {},

formNewInfo: {},

},

decode: [],//需要解码的字段

resetFields:[],//页面需要初始化清空内容的字段

notDeleteCache: [],//页面不需要初始化清空内容的字段

starShow:".parent().find('span').eq(0)||addClass||zj-MandatoryRedStar-css",//给必填项添加红星，定位元素||操作类型||红星css对应class（默认在SysSet中设置，form设置>SysSet设置）。

deleteStarShow:".parent().find('.zj-MandatoryRedStar-css')",//删除必填星号（默认在SysSet中设置，form设置>SysSet设置）。

    draftSearchJson :[], //草稿箱搜索字段内容，

classPermission:{

        allRole: {

            //只读 通过input的name控制input的只读状态

            view: {

                '1'/\*pageNode \*/: {//当前页面号

                    'index'/\*fromNode \*/: true, //只读

                }

            },

            //必填项 //通过input的name控制input 必填

            mustInput: {

                '1': {

                    'index': true,

                }

            },

            //选填项 //通过input的name控制input 选填

            maybeInput: {

                '1': {

                    'index':true,

                }

            },

            //通过class让元素隐藏  class命名规则：xxx\_hidden\_xxx

            hidden: {

                '1': {

                    'index': true,

                }

            },

            //showBlock不是控制input，是根据id控制哪些部分需要显示  FIXME:id命名规则：xxx\_showBlock\_xxx

            showBlock: {

                '1': {

                    'index':true,

                }

            },

        },

    },

//相关操作，前、后置函数配置

event:{

        "beforeSubmit":{

            'all':{//全局配置

                'all':function(){},//全局配置

            },

            '1':{//pageNode

                'add':function(){},//fromNode

            },

        },

        "afterSubmit":{

            'all':{//全局配置

                'all':function(){},//全局配置

            },

            '1':{//pageNode

                'add':function(){},//fromNode

            },

        },

        "beforeInit":{

            'all':{//全局配置

                'all':function(){},//全局配置

            },

            '1':{//pageNode

                'add':function(){},//fromNode

            },

        },

        "afterInit":{

            'all':{//全局配置

                'all':function(){},//全局配置

            },

            '1':{//pageNode

                'add':function(){},//fromNode

            },

        },

        "afterSubmitDraft":{

            'all':{//全局配置

                'all':function(){},//全局配置

            },

            '1':{//pageNode

                'add':function(){},//fromNode

            },

        },

    },

roleShow: {

allRole: { //角色

view: { //只读配置

'1': { //pageNode

'normal':{ //formNode

}, //输入元素的name值

},

},

mustInput: { //必填项

'1': {

'normal': [],//输入元素的name值

},

},

maybeInput: {//选填项

'1': {

'normal': [],//输入元素的name值

},

},

hidden: { //隐藏设置

'1': {

'normal': [],//元素的class

},

},

showBlock: { //显示设置

'1': {

'normal': [],//元素的class

},

},

field\_showBlock: {//根据字段内容设置显示

allRole: {

'1': {

'normal':{

"status":{ //字段名

'失效':["zjEvent\_user\_update"],//字段值：需要显示的元素class

'作废':["zjEvent\_use\_update"],//字段值：需要显示的元素class

}

},

},

},

},

field\_hiddenBlock: {//根据字段内容设置隐藏

allRole: {

'1': {

'normal':{

"status":{

'失效':["zjEvent\_user\_update"],//字段值：需要显示的元素class

'作废':["zjEvent\_user\_update"],//字段值：需要显示的元素class

}

},

},

},

},

},

},

//任务权限

taskPermission: {

//通过class让元素隐藏 class命名规则：xxx\_hidden\_xxx

hidden: {

'1': {

'index': {

"sourceoperation": {//字段名

//字段值:隐藏class集合

"更多意见": ['caiwuDHreportmaterialname', 'caiwuDHreportmaterial'],

}

}

},

},

//showBlock不是控制input，是根据id控制哪些部分需要显示 FIXME:id命名规则：xxx\_showBlock\_xxx

showBlock: {

'1': {

'index': {

"sourceoperation": {//字段名

//字段值

"更多意见": ['caiwuDHreportmaterialname', 'caiwuDHreportmaterial'],

}

}

},

},

},

workFlow:{ //流程图相关设置

processkey: 'TCFAP', //流程id

taskField:[], //将哪些信息中的字段存到任务表的description中

createType:"2",// 启动流程：创建任务模式 1=不创建,2=正常创建；任务撤回、驳回：1=只创建初始节点任务，2=正常创建任务 FIXME:追加、指定

msgtype: { //msgtype流转信息

1:{

'add': {

'submit': '\_1TO2',

'submit1': '\_1TO3',

'cancel': 'cancel1',

}

},

},

},

}

## sysSet：系统设置

dataUrl: " http://localhost:8080/ ", //项目后端地址

isLoadingLayer: false,//是否显示loading框

backCount:5,//页面session保存次数

mainFormId:'mainForm',//系统默认mainFormId对应的id

mainListId:"mainList",//系统默认mainListId对应的id

starShow:".parent().find('span').eq(0)||addClass||zj-MandatoryRedStar-css",//给必填项添加红星，定位元素||操作类型||添加红星css对应class（form中可再次设置，form中权重大于sysSet）。

deleteStarShow:".parent().find('.zj-MandatoryRedStar-css')||removeClass||zj-MandatoryRedStar-css'",//删除必填星号, 定位元素||操作类型||删除红星css对应class（form中可再次设置，form中权重大于sysSet）。

listSearchParent :".parent().find('span label')",//list searchRole有关，通过name找到input，再通过listSearchParent规则找到父级隐藏（list中可再次设置，list中权重大于sysSet）。

showPageNum:5,//显示页码数（list中可再次设置，list中权重大于sysSet）

pageindex:1,//第几页（list中可再次设置，list中权重大于sysSet）

color:{

mustInput: ' zj-MandatoryRedBorder-css ' //必填项红框class名称

},

classId: { //栏目classId

knowledge: 'adf41773-6fee-41eb-b023-688db3edba77', //知识栏目

},

dataDictionary: {//项目数据词典

sysLabelClassificationSort: "074ca173-3f73-41ff-87fc-38c787503298",//系统标签排序规则

},

roleTemp: {//固定角色id

contact: '07957934-231b-4b7d-8f7d-724f60439a55', //接口人

},

dowmloadTemplate: { //下载模版Id

companyList: {

id: "ec91fa94-6ad7-44b9-8470-0c0f5bba9b4d",//模版文件id

name: "企业清单" //下载文件名称

},

},

fileFolder: {//文件夹路径

operationmanual:"20211210-c4d9-4e0e-20211210doc",//操作手册

},

errorMsg: {},//错误信息

decode: [], //全局压缩项

signal: {}//菜单配置

loginFailure: function (data) {//未登录时执行的方法

        alert("请先登录系统!");

    }

## 常用变量名

zj-MandatoryRedStar-css : 必填项红星class，加上此class，会在元素前面加上\*

{

display: inline-block;

margin-right: 4px;

color: #ff4d4f;

font-size: 12px;

font-family: SimSun, sans-serif;

line-height: 1;

content: "\*";

}

zj-MandatoryRedBorder-css : 必填项红色边框class

{

Border:1px solid #95003d

}

zj-none-css//隐藏

{

display: none !important;

}

# 当前页面

## 当前页面生命周期

生命周期的顺序，跳转进入页面时，页面工作分六个步骤，其中前两个步骤进入页面就执行，后面个步骤都要等到内容加载完成后（也就是$.ready()）中执行。

有一点要说明的，dataCenter.state只会影响页面渲染，不会影响数据提交。

to的内容：

{

pageNode: '1',//当前页面所处节点名称，0、1、2、3等

fromNode: 'normal',//当前页面的状态是从哪一个状态跳转过来，fromNode记录了来源

state:’’,//信息类型

infoId:’’//信息id

classId:’’//栏目id

}

判断用户登录状态：

登录状态由后端判断，页面调用接口，如果登录失效，后端会返回固定特征码，底层ajax函数根据特征码判断是否失效，失效执行未登录callback函数。未登录callback函数内容可根据实际项目需求自定义。

### 页面准备工作

不管是哪种方式进入页面，最终页面相关信息都在浏览器的sessionStorage中，后面都是对sessionStorage相关内容进行处理。

sessionStorage内容：

oldSession：[ //多个历史页面的session信息集合

{

page:{

pageUrl: '',//页面完整url

pageName: '',//页面名称（页面html文件名称，不含“.html”后缀）

pageNode: '1',//当前页面所处节点名称，0、1、2、3等

fromNode: 'normal',//当前页面的状态是从哪一个状态跳转过来，fromNode记录了来源

},

menu: {}, //页面菜单数据

org: {

orgId: '',//组织id

orgName: '',//组织名称

},//组织信息

list: {},//list相关配置

form: {},

to:{},

other:{}

},//页面信息pageInfo

{...},//另一个页面session信息

{…}//另一个页面session信息

]

有两种方式进入页面时，sessionStorage信息不完整，先要准备好这些信息，方便后面处理。

#### 浏览器前进、后退

* 获取sessionStorage中oldSession（历史页面的session集合）
* 判断oldSession的length是否大于1。正常情况下至少为2（登陆成功后跳转到首页length=2）

小于等于1的情况：

* 1、浏览器已打开其他页面，输入系统页面url，页面加载好后点击浏览器回退，回到之前的其他页面，再点击浏览器前进按钮，回到系统页面。这是情况同直接输入url，不用额外处理。
* 2、浏览器已在系统某个页面，输入其他页面url（例如baidu.com），浏览器跳转到百度页。在百度页，点击回退回到系统页面，浏览器会自动还原系统页面session，不用额外处理。
* 使用pop()方法获取oldSession数组的最后一个元素backSession，并将其移除出oldSession
* 从backSession中获取pageName（页面名称）
* 将删除最后一个元素的oldSession重新存到sessionStorage中
* 使用window.history.pushState将当前页面直接添加到浏览器历史记录中，修改浏览器历史记录，禁用前进按钮（原因：点击前进，对应页面的session无法获取恢复，session已被删除，又没有用to传值，前进的页面相关参数无法获取）。

#### 浏览器地址栏输入url跳转

url示例:http://127.0.0.1:8081/web/LLR/LLR.html?zjxx=zjxx&pageNode=1&fromNode=normal&infoId= 6d9f70d6-79c0-11ec-9b9b-0050569d2396&state=draft

url中?等内容使用encodeURIComponent压缩，解析时使decodeURIComponent解码

* 根据url中的相关参数准备session
* 判断有没有关键词（zjxx=zjxx），没有不处理
* 有：将整理url中的参数整理成对象requestObj
* 判断sessionStorage中的oldSession是否存在，不存在就用url的参数
* 判断sessionStorage中的oldSession的相关属性和url的参数是否一致，不一致用url的（相关属性：pageUrl、pageNode、formNode、infoid）
* 将oldSession中length-1的seesion对象的to属性 = requestObj
* 将修改后的oldSession存到浏览器seesionStorage中

### 页面生成，执行pageReady();

### 初始化dataCenter中相关属性

调用\_init()函数，\_init()解决问题过程如下：

* 从localStorage中获取user信息，初始化dataCenter.user
* 获取浏览器url，初始化pageUrl、pageName
* 获取sessionStorage中oldSession，初始化dataCenter. oldSession
* oldSession.pop().to获取oldSession数组的最后一个元素的to属性的值toSession
* 根据dataCenter.sessionFields['base']中设置的属性名，将toSession中对应的属性初始化到dataCenter中。
* 判断页面类型（list还是form）

##### list：

根据dataCenter.sessionFields[list]、dataCenter.sessionFields[listpage]中设置的属性名，将toSession中对应的属性初始化到dataCenter['list'][sysSet.mainListId]中。

##### form：

根据dataCenter.sessionFields[form]中设置的属性名，将toSession中对应的属性初始化到dataCenter[‘from’][sysSet.mainFormId]中。

### 必填项恢复原始显示样式

绑定必填项输入框红框恢复默认样式（删除红框class）事件：当必填项输入框不为空失去焦点后，将红框class删除。

### 绑定列表中自动计算事件

table中子孙标签中含有class名computingOfExpression的元素失去焦点后，获取当前DOM元素的computingOfListId属性值（值为listId），获取list的计算规则，调用recalculate函数计算list中相关内容。

### 调用initPage()函数初始化页面

#### 初始化select、日期、富文本编辑器等插件

#### 初始化page权限：当前处理详情页权限--classId和infoId不为空

* 根据page中的classId和infoId调用信息权限函数getInfoPermission
* 获取页面主信息的修改、添加、删除权限，并将权限信息写入page中

#### 根据dataCenter中showBlock、hiddenBlock设置调用initPageShowBlock()、initPageHiddenBlock()函数，初始化页面DOM显示、隐藏。

#### 加载页面模版

* 获取页面上所有含有“[template]”属性的DIV。
* 根据“[template]”属性的值，从templateObj中获取对应内容，渲染到对应DIV中。

#### 语言切换switchLanguage()

#### 初始化dataCenter.form中所有的form（非list中的form）

* 备份form的roleShow -backupFormRoleShow(formId):form的roleShow备份到form的roleShowBack中。后面会重新生成roleShow的内容，所以这里先保存原始数据。
* 重置input内容- restFormInput(formId)：根据form的resetFields设置，将form中对应的input（通过name匹配）设置为空值。页面刷新或者回退，浏览器会保留之前input中输入的值，浏览器历史记录机制。
* 动态添加roleShow的数据-concordanceForm(formId)。
* 删除form所有输入元素的必填项红星class-initCancelInputState(formId)：先将必填项红星class删除，后面再根据form中的配置动态添加。
* 初始化只读项：根据form中view 设置调用initViewInput(formId)函数将对应输入元素（通过name匹配）设置为只读;
* 初始化隐藏项；根据form中hidden设置调用initHiddenClass(formId)函数将对应元素（通过class匹配）设置为隐藏;
* 初始化显示板块：根据form中showBlock设置调用initFormBlock(formId)函数将对应元素（通过class匹配）设置为显示;
* 初始化必填项页面标识（红星标识）：根据form中mustInput设置调用initMustInput(formId)函数将对应元素（通过name匹配）加上必填项红星class。
* 将form中相关元素（通过class匹配）绑定submit（class为zj\_submitButton） 、submitDraf（class为zj\_submitDraftButton）方法。
* 页面新建按钮：
* 按钮固定class=”zjDC\_addButton”、zj-infoId="{{id}}"(固定写法) 、zj-pageUrl=""(编辑详情页地址)、zj-self=false  是否新开页签  true=新开  false=当前页跳转
* 点击获取信息id和编辑页地址，固定pageNode=1,formNode=”update”跳转。

#### 获取页面类型（form、list），根据页面类型调用对应函数渲染数据。

##### form页面-数据渲染

###### form的infoId为空：则不需要获取数据

* 将form的down.info、down.formInfo、down.formNewInfo设置为空对象{}
* 执行beforeInit方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行beforeInit-pageNode-formNode属性内容，再执行beforeInit属性内容。
* 获取id=formId的标签中beforeInit属性，先执行beforeInit-pageNode-formNode属性内容，再执行beforeInit属性内容
* 获取当前form配置中的event属性，先执行beforeInit-pageNode-formNode下方法，再执行beforeInit-all-all下方法
* 执行js文件中beforeInit的方法
* 获取form中所有有strictType属性的元素，根据strictType的值（固定值对应固定规则）调用设置输入限制-inputStrict
* 执行afterInit方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterInit-pageNode-formNode属性内容，再执行afterInit属性内容。
* 获取id=formId的标签中beforeInit属性，先执行afterInit-pageNode-formNode属性内容，再执行afterInit属性内容
* 获取当前form配置中的event属性，先执行afterInit-pageNode-formNode下方法，再执行afterInit-all-all下方法
* 执行js文件中afterInit的方法

###### form的infoId不为空：获取数据

* 整理接口参数
* 根据dataCenter[form][formId]中state的值，调用对应的方法获取数据，并将数据整理成统一的数据格式给回调函数。根据form的state值配置class内容：
* 信息：

infoClass:{

classId:”info”,

uuid:栏目id

}

返回结果的infos[0]

* 草稿：

infoClass:{

classId:” draft”,

}

返回结果的infos[0].infoJson

* 修改记录：

infoClass:{

classId:” modifyRecord”,

uuid:栏目id

}

返回结果的infos[0]

* 归档：

infoClass:{

classId:” archive”,

uuid:栏目id

}

返回结果的infos[0]

* 执行downInfo方法存储、转码数据到down.info（原始数据）、down.formInfo、down.formNewInfo中（方法详细说明见页面逻辑）
* 将原始数据存到down.info
* 根据sysSet中decode配置将转码数据转回正常数据- escapeObjToHtmlObj
* 根据form中decode配置将转码数据转回正常数据
* 将已转码的数据存储到down.formInfo和down.formNewInfo中
* 判断是否获取栏目模版，获取则将form表单替换成栏目模版—待与后端确认完善
* 执行beforeInit方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行beforeInit-pageNode-formNode属性内容，再执行beforeInit属性内容。
* 获取id=formId的标签中beforeInit属性，先执行beforeInit-pageNode-formNode属性内容，再执行beforeInit属性内容
* 获取当前form配置中的event属性，先执行beforeInit-pageNode-formNode下方法，再执行beforeInit-all-all下方法
* 执行renderForm渲染表单（方法详细说明见页面逻辑）
* 根据down.formInfo数据的key值匹配form中输入元素的name赋值value
* 获取form中所有有strictType属性的元素，根据strictType的值（固定值对应固定规则）调用设置输入限制-inputStrict
* 执行afterInit方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterInit-pageNode-formNode属性内容，再执行beforeInit属性内容。
* 获取id=formId的标签中afterInit属性，先执行afterInit-pageNode-formNode属性内容，再执行afterInit属性内容
* 获取当前form配置中的event属性，先执行afterInit-pageNode-formNode下方法，再执行afterInit-all-all下方法

##### form页面-数据提交：

判断form是否是信息提交、草稿提交、流程相关提交

###### 信息：点击class为zj\_submitButton按钮或者直接执行submit(formId)方法

* 判断form是否设置提交防抖、节流
* 执行beforeSubmit方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行beforeSubmit-pageNode-formNode属性内容，再执行beforeSubmit属性内容。
* 获取id=formId的标签中beforeSubmit属性，先执行beforeSubmit-pageNode-formNode属性内容，再执行beforeSubmit属性内容
* 获取当前form配置中的event属性，先执行beforeSubmit-pageNode-formNode下方法，再执行beforeSubmit-all-all下方法
* 执行js文件中beforeSubmit的方法
* 执行serializeForm获取页面上form表单信息（方法详细说明见页面逻辑）
* 获取所有的输入元素的name与value的键值对
* 将获取的数据存到down.formNewInfo中
* 根据sysSet中decode配置将已获取的数据转义- html2Escape
* 根据form中decode配置将已获取的数据转义-html2Escape
* 将已转义的数据存储到up.formInfo和中
* 执行checkDialogForm，检查必填项和选填项，根据情况提示。
* 判断form的state的值，信息提交、启动流程、完成任务
* 启动流程：state=启动流程
* 设置参数模版id：parmas.info.moudleid = dataCenter.form[formId].workFlow.processkey;
* 完成任务（修改信息）：state= 完成任务
* 调用getTaskParmas函数收集任务参数
* 调用新增、更新信息方法
* 执行afterSubmit方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterSubmit-pageNode-formNode属性内容，再执行afterSubmit属性内容。
* 获取id=formId的标签中afterSubmit属性，先执行afterSubmit-pageNode-formNode属性内容，再执行afterSubmit属性内容
* 获取当前form配置中的event属性，先执行afterSubmit-pageNode-formNode下方法，再执行afterSubmit-all-all下方法
* 执行js文件中beforeSubmit的方法

###### 草稿：点击class为zj\_submitDraftButton按钮或者直接执行submitDraft(formId)方法

* 执行serializeForm获取页面上form表单信息，serializeForm函数的功能在前面已经有了描述。
* 执行getDraftSubmitParameter将form的up.formInfo、dataCenter.user整理成接口需要的参数，其中根据form的draftSearchField配制，将up.formInfo相关字段和内容整合成draft中searchJson的值。
* 调用新增、更新草稿方法
* 执行afterSubmitDraft方法
* 获取form中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterSubmitDraft-pageNode-formNode属性内容，再执行afterSubmitDraft属性内容。
* 获取id=formId的标签中afterSubmitDraft属性，先执行afterSubmitDraft-pageNode-formNode属性内容，再执行afterSubmitDraft属性内容
* 获取当前form配置中的event属性，先执行afterSubmitDraft-pageNode-formNode下方法，再执行afterSubmitDraft-all-all下方法
* 执行js文件中afterSubmitDraft的方法

###### 任务其他操作(同意、驳回、撤回、加签、转办、终止、冻结、解冻)：点击class为zjDC\_operationTask按钮(taskType="操作类型")

* 任务操作(同意、驳回、撤回、加签、转办)：
* 调用getTaskParmas函数收集任务参数
* 项目操作(终止、冻结、解冻)：
* 调用getFormProjectParmas函数收集项目参数
* 调用run接口方法方法

##### list页面

###### 判断list中是否配置搜索参数方法，

* 没有：使用默认方法getSeachParameter
* 有：执行配置的方法

###### 根据state判断列表数据类型，调用对应方法获取数据

###### 执行downInfo方法存储、转码数据

* 将原始数据存到listDownInfos
* 根据sysSet中decode配置将转码数据转回正常数据- escapeObjToHtmlObj
* 根据list中decode配置将转码数据转回正常数据
* 将已转码的数据存储到list的listInfos中

###### 调用backupListState(listId);

备份权限配置

###### 调用concordanceList(listId);

整合权限

###### 根据list中是否有class名为：zjDC\_listSearch模块判断是否存在搜索模块，存在渲染renderListSearch

* 整合用户list搜索权限配置getListSearchRole
* 初始化搜索模块显示、隐藏
* 绑定搜索中查询、重置、取消事件：（通过class匹配）绑定，查询（class为zj\_searchButton）、重置（class为zj\_resetButton）、取消（class为zj\_cancelButton）。

###### 将listInfos中每一条信息的infoNodeSys、infoStateSys设置为页面的pageNode、fromNode值。作为list中form的属性。

###### 判断是否获取栏目模版，获取则将list的渲染模版tempRow替换成栏目模版

###### 执行beforeRender方法

* 获取list中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行beforeRender-pageNode-formNode属性内容，再执行beforeRender属性内容。
* 获取id=listId的标签中beforeRender属性，先执行beforeRender-pageNode-formNode属性内容，再执行beforeRender属性内容
* 获取当前list配置中的event属性，先执行beforeRender-pageNode-formNode下方法，再执行beforeRender-all-all下方法
* 执行js文件中beforeRender的方法

###### 执行renderListItem渲染表单（方法详细说明见页面逻辑）

* 记录list中form的红框
* 判断listInfos无数据且设置getlistInfosData，执行getlistInfosData方法整理数据
* 获取字段排序设置，对信息进行排序
* 获取删除行设置，删除对应的数据
* 判断前端分页函数getPaginationConditions是否设置，设置了就执行
* 根据list类型table、row调用不同的方法渲染行

1. renderListItemByRow（页面非table样式）

* 获取渲染模版
* 获取删除列设置，删除模版中对应列
* for循环渲染每一行数据数据
* 渲染每一行中的form

1. renderListItemByData（页面table样式）

* 获取渲染模版
* 获取删除列设置，删除模版中对应列
* for循环渲染每一行数据数据
* 渲染每一行中的form
* 设置输入限制 inputStrict
* 恢复list中红框

###### 根据list分页设置渲染分页组件

###### 初始化list中模块显示设置：根据list中showBlock设置调用initListShowBlock（方法详细说明见页面逻辑）

* 整合用户角色，以下设置都是以角色为基础
* 根据模块id，初始化模块的显示
* 根据数据中字段名称和值、模块class，初始化模块的显示
* getListListListRole

###### 初始化list中模块隐藏设置：根据list中hiddenBlock设置调用initListHiddenBlock（方法详细说明见页面逻辑）

* 整合用户角色，以下设置都是以角色为基础
* 根据模块id，初始化模块的隐藏
* 根据数据中字段名称和值、模块class，初始化模块的隐藏
* getListListListRole

###### 执行afterRender方法

* 获取list中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterRender-pageNode-formNode属性内容，再执行afterRender属性内容。
* 获取id=listId的标签中afterRender属性，先执行afterRender-pageNode-formNode属性内容，再执行afterRender属性内容
* 获取当前list配置中的event属性，先执行afterRender-pageNode-formNode下方法，再执行afterRender-all-all下方法
* 执行js文件中afterRender的方法

###### 根据class名（zjDC\_ deleteButton）绑定每一行删除事件

* 按钮固定class=”zjDC\_ deleteButton”、zj-infoId="{{id}}"(固定写法) 。
* 点击获取信息id
* 二次弹框确认
* 执行beforeDelete
* 获取list中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行beforeDelete-pageNode-formNode属性内容，再执行beforeDelete属性内容。
* 获取id=listId的标签中afterRender属性，先执行beforeDelete-pageNode-formNode属性内容，再执行beforeDelete属性内容
* 获取当前list配置中的event属性，先执行beforeDelete-pageNode-formNode下方法，再执行beforeDelete-all-all下方法
* 执行js文件中beforeDelete的方法
* 判断list的isPersist是否需要持久化数据
* 持久化：
* 调用删除信息接口删除数据，
* 删除成功回调中调用getListBlockData(listId)渲染列表
* 执行afterDelete

获取list中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterDelete-pageNode-formNode属性内容，再执行afterDelete属性内容。

获取id=listId的标签中afterDelete属性，先执行afterDelete-pageNode-formNode属性内容，再执行afterDelete属性内容

获取当前list配置中的event属性，先执行afterDelete-pageNode-formNode下方法，再执行afterDelete-all-all下方法

* 非持久化：
* 当前列表数据listData中根据已获取的当前数据id，删除当前数据
* 调用renderListByData方法渲染整合好的新数据
* 执行afterDelete

获取list中有class=’ zjDCFrameEvent’的子孙标签，按照actionorder排序先执行afterDelete-pageNode-formNode属性内容，再执行afterDelete属性内容。

获取id=listId的标签中afterDelete属性，先执行afterDelete-pageNode-formNode属性内容，再执行afterDelete属性内容

获取当前list配置中的event属性，先执行afterDelete-pageNode-formNode下方法，再执行afterDelete-all-all下方法

###### 每行编辑按钮：

* 按钮固定class=”zjDC\_updateButton”、zj-infoId="{{id}}"(固定写法) 、zj-pageUrl=""(编辑详情页地址)。
* 点击获取信息id和编辑页地址，固定pageNode=1,formNode=”update”跳转。

###### 每行复选框按钮：

* 按钮固定class=” zjDC\_listCheckBox”、zjDC-infoId="{{id}}"(固定写法) 、zj-pageUrl=""(编辑详情页地址)。

###### 顶部全选按钮：

* 按钮固定class=” zjDC\_listAllCheckBox”。

###### 批量删除按钮：

* 按钮固定class=” zjDC\_batchDeleteButton”。

###### 自动计算list中需要合计的内容

## 当前页面跳转

浏览器回退：往上一个页面跳转，当前页面的准备工作—无

浏览器前进：往下一个页面跳转，当前页面的准备工作—无

Js代码往下一个页面跳转，当前页面的准备工作

### 在dataCenter.to中，配置相关信息

dataCenter.to = {

pageNode: "",//目标页面的pageNode，不传使用默认值

fromNode: "",//目标页面的fromNode，不传使用默认值

infoId: "",//如果目标页面是form页面，会自动根据信息id获取信息渲染

state: "",//信息类型

other:"",//其他需要传递到目标页面的内容

}

### 调用toUrl函数跳转页面

* 获取当前页面sessionStorage中的~~user、~~oldSession， 为后面新开页签恢复当前页面sessionStorage使用。
* 调用函数saveSession，获取最新的~~user、~~oldSession信息，存到sessionStorage中。
* 跳转页面或者打开新页签

# 页面权限配置：

## 整页权限

框架先执行dataCenter.showBlock配置，再执行dataCenter.hiddenBlock。所以hiddenBlock的优先级大于showBlock

## form权限

### form区域控制：

显示（showBlock）是根据id控制哪些部分需要显示

### form的内控制：

控制显示（view-通过name操作）、隐藏（hidden-通过class操作）、必填（mustInput-通过class操作）、选填（maybeInput-通过class操作）、流程控制（msgType）

## list权限

### list区域控制：

显示（showBlock）是根据id控制哪些部分需要显示

### list列控制：

thead控制列（删除列）、row控制列（删除行）

### list搜索控制：

searchRole，有的就显示，没有的就不显示。现在做是通过input的name进行控制。可以将input对应的父节点整体删除。

## 总结

* form和list可以相互嵌套，各自权限互不影响。
* 不能由配置解决的权限问题，在业务逻辑中由硬代码解决。

页面权限与dom元素对应关系

**页面：**

showBlock\_blank（显示）：dom元素id—配置有问题，功能没有实现

hiddenBlock\_blank（隐藏）：dom元素id—没有这个配置

list：

**搜索区域**

showBlock（显示）：dom元素name

hiddenBlock（隐藏）：dom元素name—没有这个配置

**整个list区域**

showBlock（显示）：dom元素id

hiddenBlock（隐藏）：dom元素id

**列**

删除：dom元素（标题头）name

**行**

删除：需要删除行的行数，从0开始。

showBlock（显示）：dom元素class—没有这个配置

hiddenBlock（隐藏）：dom元素class—没有这个配置

**根据字段值影响行**

field\_showBlock（显示）：dom元素class

field\_hiddenBlock（隐藏）：dom元素calss

form：

view（只读）：dom元素name

mustInput（必填）：dom元素name

mayInput（选填）：dom元素name

hidden（隐藏）：dom元素class

showBlock（显示）：dom元素id

# 页面逻辑函数

## dialog函数调用

### showFormDialog (formId, temName, output, callback)：

* 根据formId和temName调用showDialog添加dialog的DOM
* 调用initFormPage(formId)初始化Form
* 调用renderForm(formId) 渲染Form
* 禁止底层的body滚动
* 当点击确定按钮时
* 调用form校验方法checkDialogForm
* 执行dataCenter['form'][formId].setOutInput或者
* 执行callback“回调函数”
* 点击取消按钮
* 移除dialog中的Dom
* 调用recoveryInputState(formId)恢复备份
* 启用底层的body滚动

### showFormDialogToList (formId, temName, inputObj, callback)

* 根据formId和temName调用showDialog添加dialog的DOM
* 调用initFormPage(formId);初始化Form
* 调用renderForm(formId) 渲染Form
* 禁止底层的body滚动
* 当点击确定按钮时
* 调用form校验方法checkDialogForm
* 根据inputObj获取原始数组数据
* 获取dialog中form的数据
* 将form的数据更新到原始数组数据中
* 将更新后的原始数组数据存到inputObj中
* 执行callback“回调函数”
* 点击取消按钮
* 移除dialog中的Dom
* 调用recoveryInputState(formId)恢复备份
* 启用底层的body滚动

### initAjaxFormDialog (formId, temName, inputObj, callback)

* 根据formId和temName调用showDialog添加dialog的DOM
* 调用initFormPage(formId);初始化Form
* 调用getAllFormBlockData (formId) 请求后端渲染Form
* 禁止底层的body滚动
* 当点击确定按钮时
* 调用form校验方法checkDialogForm
* 根据inputObj获取原始数组数据
* 获取dialog中form的数据
* 将form的数据更新到原始数组数据中
* 将更新后的原始数组数据存到inputObj中
* 执行callback“回调函数”
* 点击取消按钮
* 移除dialog中的Dom
* 调用recoveryInputState(formId)恢复备份
* 启用底层的body滚动

### alertWarnDialog (title,content,okCallback)：

弹出框，有标题、内容、确定回调函数

### confirmDialog(title, content, okCallback, cancelCallback)：

二次确认弹出框，有标题、内容、确定回调函数、取消回调函数

### alertDialogForHide：

提示框，n秒后自动隐藏

### 用户可以在这几个dialog的基础上封装自己的其他的dialog

## form可调用函数

### getAllFormBlockData(formId)：获取form信息、渲染

* 整理参数，根据form的state值配置class内容：
* 信息：

class:{

classId:”info”,

uuid:栏目id

}

* 草稿：

class:{

classId:” draft”,

}

* 威客：

class:{

classId:” witkeyInfo”,

uuid:栏目id

}

* 归档：

class:{

classId:” archive”,

uuid:栏目id

}

* 调用信息详情接口获取数据
* 执行downInfo方法存储、转码数据
* 将返回结果的infos[0]存到down.info
* 判断form的state值是否是草稿，是：以下转码原始数据就是infos[0].infoJson，不是：infos[0]
* 根据sysSet中decode配置将转码数据转回正常数据- escapeObjToHtmlObj
* 根据form中decode配置将转码数据转回正常数据
* 将已转码的数据存储到down.formInfo和down.formNewInfo中
* 判断是否获取栏目模版，获取则将form表单替换成栏目模版
* 执行beforeInit方法
* 执行renderForm渲染表单
* 执行afterInit方法

### initFormPage(formId)：初始化页面效果

* 调用backupFormRoleShow备份form的roleShow
* form的roleShow备份到form的roleShowBack中

后面会重新生成roleShow的内容，所以这里先保存原始数据。

* 调用concordanceForm动态添加roleShow的数据。
* 调用restFormInput重置input内容

根据form的resetFields设置，将form中对应的input（通过name匹配）设置为空值。页面刷新或者回退，浏览器会保留之前input中输入的值，浏览器历史记录机制。

* 调用initCancelInputState删除form所有输入元素的必填项红星class

先将必填项红星class删除，后面再根据form中的配置动态添加。

* 调用initViewInput初始化只读项

根据form中view 设置将对应输入元素（通过name匹配）设置为只读;

* 调用initHiddenClass初始化隐藏项

根据form中hidden设置将对应元素（通过class匹配）设置为隐藏;

* 调用initFormBlock初始化显示板块

根据form中showBlock设置将对应元素（通过class匹配）设置为显示;

* 调用initMustInput初始化必填项页面标识（红星标识）

根据form中mustInput设置将对应元素（通过name匹配）加上必填项红星class。

* 将form中相关元素（通过class匹配）绑定submit（class为zj\_submitButton） 、submitDraf（class为zj\_submitDraftButton）方法。

### renderForm(formId)：渲染，通过dom查找input，根据name属性，分别修改value

* 获取form的down.formInfo的数据
* 获取form中所有的输入元素input、checkbox、radio、select、textarea和span，根据down.formInfo数据的key值匹配form中输入元素的name赋值value
* 获取form中所有有strictType属性的元素，根据strictType的值（固定值对应固定规则）调用inputStrict方法设置输入限制

### renderFormByData (formId, data)：根据数据渲染form

* 将data.id赋值给form的infoId
* 将data赋值给down.formInfo和down.formNewInfo
* 执行beforeInit方法
* 执行renderForm渲染表单
* 执行afterInit方法

### downInfo(formId,data)：存储、转码数据

* 将data的infos[0]存到down.info
* 判断form的state值是否是草稿，是：以下转码原始数据就是infos[0].infoJson，不是：infos[0]
* 根据sysSet中decode配置将转码数据转回正常数据- escapeObjToHtmlObj
* 根据form中decode配置将转码数据转回正常数据
* 将已转码的数据存储到down.formInfo和down.formNewInfo中

### submit(formId)：form提交

* 执行beforeSubmit
* 执行serializeForm方法收集form数据存到form的down.formNewInfo
* 执行checkFormMustInput检查必填项，有必填项未填，弹框提示
* 执行getFormParmas整理form提交参数
* 调用新增、更新方法提交数据
* 执行afterSubmit

### serializeForm(formId)：收集form数据

* 创建空的表单数据对象{}
* 判断form的infoId属性是否有值，有值则将infoId配置的值赋给表单对象的id属性
* 遍历form中所有的输入元素input、checkbox、radio、select、textarea，收集输入元素的name和value
* 根据sysSet中decode配置将数据压缩- htmlObjToEscapeObj
* 根据form中decode配置将数据压缩

## list列表可调用函数

### getListBlockData(listId, fomId)：调用后台方法,参数为:块listId(与dataCenter.list里的字段对应)

* 判断list中是否配置收集参数方法，
* 没有：使用默认方法getSeachParameter收集参数
* 有：执行配置的方法收集参数
* 调用获取信息列表接口获取数据
* 执行downInfo方法存储、转码数据
* 将原始数据存到listDownInfos
* 根据sysSet中decode配置将转码数据转回正常数据- escapeObjToHtmlObj
* 根据list中decode配置将转码数据转回正常数据
* 将已转码的数据存储到list的listInfos中
* 根据接口返回结果设置list的分页参数
* 调用renderList渲染列表

### renderList(listId,fomId)：显示表格

listTableId中的table由三个部分组成，三个部分共用一个父级元素id

* 搜索项class：

listSearch，里面有class：item的子项，在子项中，class=title显示标题。在listSearch有几个特殊的input：。

* orderType：排序类型。
* listSearch通过searchRole结合当前用户的角色，来控制子项是否保留；
* 若searchRole中定义的某一项，在页面上不存在，则新增一个class叫做item的input子项,
* 并将此项的值赋给input的name(初步默认只新增input)。
* 搜索部分有几个按钮，class分别是，搜索按钮：searchButton、重置按钮：resetButton、取消按钮：cancelButton
* 表格

表格现在有两种，一种含有table，一种没有table，每一行的有class为row

* class：listTable。table中的item，及tr项。table中的th和tr分开，   tr中的th增加name属性，具体这个属性对应的th是否要显示由权限决定，可在dataCenter中配置该列的显示权限。权限中含在thead中的不显示。tr的模板直接从table中去取，在tr中的每一个td中的中用{{title}}记录需要的字段的值
* 没有table：每一行含有row的class
* 分页项 class：

pageCount共多少页，currentPage当前页，first第一页，last最后一页 当前选择页selected、preclick 上一页按钮，nextclick 下一页按钮、class=listPagination，里面有class=item（selected）的子项，对自相中的a标签修改。定义分页项的模板，就直接在这里定义，以后页面直接使用这个模板就行了

### submitListForm(listId, callback): 保存数据列表数据

* 调用checkListForm方法，校验list中所有form的必填项，有必填项就提示
* 调用getJsonFromListForm获取list中所有form的数据listData
* 获取list的deleteIds，需要删除的数据
* 判断listData和deleteIds是否为空
* 都不为空：调用新增、修改信息接口，回调中调用删除信息接口
* listData不为空：调用新增、修改信息接口
* deleteIds不为空：调用删除信息接口

### renderListByData(listId, data, formId) : 根据数据渲染list

* 将数据存储到list的listInfos中
* 调用renderList渲染列表

### returnListTempByData(listId, listInfos): 根据模板和数据生成DOM

* 获取list的内容模版tempRow
* 循环数据listInfos，将数据与模版tempRow结合
* 返回生成DOM

# 页面渲染

## list渲染

* 循环listInfos，获取每一条信息
* 执行replace将模版中占位符标签替换成信息中对应字段数据
* 占位符标签示例：

{{title}} //具体某个字段

{{[orgid ][title]}} //字段内容json，获取json的某个属性

{{orgid.title}} //左联字段

{{[orgid.title][title]}} //左联字段内容json，获取json的某个属性

* 根据list的thead设置删除对应列
* 将已渲染好数据的模版append到list中并显示
* list业务逻辑属性：
* beforeRender：添加在id=”listid”的标签中，渲染前执行
* afterRender：添加在id=”listid”的标签中，渲染后执行
* beforeDelete：添加在id=”listid”的标签中，列表删除行确定前执行
* afterDelete：添加在id=”listid”的标签中，列表删除行确定后执行

## form渲染

* form渲染标签：input、select、textarea、span
* 获取标签的name属性
* 判断name是否是formInfo的属性，是就赋值对应数据
* form业务逻辑属性：
* beforeInit：添加在id=”formid”的标签中，渲染前执行
* afterInit：添加在id=” formid”的标签中，渲染后执行
* beforeSubmit：添加在id=” formid”的标签中，提交前执行
* afterSubmit：添加在id=” formid”的标签中，提交后执行
* afterSubmitDraft：添加在id=” formid”的标签中，草稿提交后执行

## list、form中标签业务逻辑属性

以下业务逻辑属性添加在对应标签中。

* beforeRender：添加当前标签中，标签渲染前执行
* afterRender：添加当前标签中，标签渲染后执行
* beforeSubmit：添加在当前标签中，form提交前执行
* afterSubmit：添加在的标签中，form提交后执行
* actionOrder:执行顺序，默认的优先级最低。多个标签都有相关属性时，可以设置标签之间属性对应代码的执行顺序。

## 非list、form中标签业务逻辑属性

* beforeRenderDOC：添加当前标签中，document ready前执行
* afterRenderDOC：添加当前标签中，document ready渲染后执行
* actionOrder:执行顺序，默认的优先级最低。多个标签都有相关属性时，可以设置标签之间属性对应代码的执行顺序。

# 前端框架相关功能

## 人员

### 设置人员信息

dataCenter.user.setUer(人员对象);

### 设置人员角色

添加角色

dataCenter.user.addRole(人员角色对象);

删除角色

dataCenter.user.deleteRole(人员角色id);

### 是否拥有某个角色

dataCenter.user.isHasRole(人员角色id);

## 页面跳转

toUrl (url-跳转地址, self-是否当前页打开, clearRole –是否清除角色默认= true, obj-需要传递的参数to)

### 草稿箱列表转到信息详情页

draftToDetail

### 待办列表跳转审批详情页

## dialog

### 显示dialog(list)

### 显示dialog(form)

### 类似alert弹出框dialog

alertWarnDialog (title-标题,content-内容,okCallback-确定按钮回调)：

### 二次确认弹出框dialog

confirmDialog(title-标题,content-内容,okCallback-确定按钮回调, cancelCallback-取消按钮回调)

### 提示框，n秒后自动隐藏

alertDialogHide (title-标题,content-内容 element-触发对象, time-保持时间, callback-隐藏后回调) {

## form

### 获取信息详情、渲染页面form

getAllFormBlockData(formId)

### 初始化form页面效果

initFormPage(formId)

### 渲染form

renderForm(formId)

### 根据数据渲染form

renderFormByData (formId, data-信息对象)

### 获取数据后，渲染form前，前置函数

beforeInit

### 渲染form后，后置置函数

afterInit

### form提交信息

submit(formId)：

### 收集form数据

serializeForm(formId)

### 收集form数据，form提交信息前，前置函数

beforeSubmit

### form提交信息后，后置函数

afterSubmit

### form检查必填项，并提示

checkDialogForm (formId)

### 重置form

resetForm(formId)

### 提交草稿

submitDraft

### 提交草稿后，后置函数

afterSubmitDraft

### 启动流程

### 完成任务

## list

### 获取信息列表、渲染页面list

getListBlockData(listId, fomId)

### 查询

### 渲染list

renderList(listId,fomId)

### 树形结构渲染--全量渲染

### 提交列表数据

submitListForm(listId, callback)

### 根据数据渲染list

renderListByData(listId, data, formId)

### 根据模板和数据生成DOM

returnListTempByData(listId, listInfos):

### 获取数据后，渲染list前，前置函数

beforeRender

### 渲染list后，后置函数

afterRender

### 获取List数据集合json

getJsonFromListForm(listId,formId)

### 检查list必填项

checkListFormMustInput(listId,formId)

### 获取列表勾选数据

### 分页记住上一页选择结果

### 已勾选数据默认选中

### 删除列表信息

### 删除前置函数

beforeDelete

### 删除后置函数

afterDelete

## 文件

### 导入

### 导出

### 上传文件

### 批量上传文件

### 下载文件

### 批量下载文件

### 多个文件打包下载

## 接口

### 新增修改信息接口

zjxxAjaxAddOrUpdate(params-参数, callback-成功回调, err-错误回调)

### 获取信息详情接口

zjxxAjaxDetail(params-参数, callback-成功回调, err-错误回调)

### 删除接口

zjxxAjaxDelete(params-参数, callback-成功回调, err-错误回调)

### 批量删除同一个栏目下信息接口

batchDelete (classId-栏目id, id-信息id集合,callback-成功回调, err-错误回调)

### 批量删除不同栏目下信息接口

batchDeleteForDiffColumn (infoList-参数集合, callback-成功回调, err-错误回调)

### 获取信息列表

zjxxAjaxList(params-参数, callback-成功回调, err-错误回调)

### 导入信息接口

zjxxAjaxExport(params-参数, callback-成功回调, err-错误回调)

### 导出信息接口

zjxxAjaxImport(params-参数, callback-成功回调, err-错误回调)

### 批量处理接口

zjxxAjaxBatch(params-参数, callback-成功回调, err-错误回调)

### Run处理接口

zjxxAjaxRun(params-参数, callback-成功回调, err-错误回调)

# 项目公共功能解决方案示例

## 登录

## 注销

## 修改密码

## 顶部菜单

## 左侧菜单

## 列表动态设置

## 列表动态排序

## 树形结构、层级结构渲染--点击渲染子数据

### 点击展开子数据

### 点击收起子级

## 富文本编辑器插件

## 已读、未读信息列表

## 中英文切换

## 表单打印预览+水印

## 搜索框历史记录--类似百度

## 发送邮件

## 文件预览

## 文件在线编辑

# 附件：草稿箱解决方案

## 草稿箱表中字段

id：草稿主键

title:标题

ftitle:副标题

pageName：保存页面名称

pageNode：节点（pageNode）当前页面所处节点名称

fromNode：来源，当前节点是从哪一个节点跳转过来的

draftSession：保存页面session，便于以后恢复到这个页面

creatorId：创建者id

classId：信息栏目id

infoId：信息id

comments：来源

infoJson：信息的所有字段信息整理成一个json字段保存在草稿箱中

searchJson：这个字段主要是给草稿箱列表使用，因为在草稿箱列表中，本质上没有必要使用infoJson字段，而将列表重要使用的字段信息直接保存在searchJson中一遍列表使用。这个字段的值可以在form中配置。

## 总结

* 草稿表信息id和userid联合主键，
* 审核过程中保存的草稿是任务草稿，草稿id是任务id
* 本质上不需要记录工作流信息，记录的是流程对应信息主体（概念实体）。

主任务如果有多条线，只会执行其中一条，不可能同时多条成立

任务集需指定内部子任务执行方式：并行、竞争