

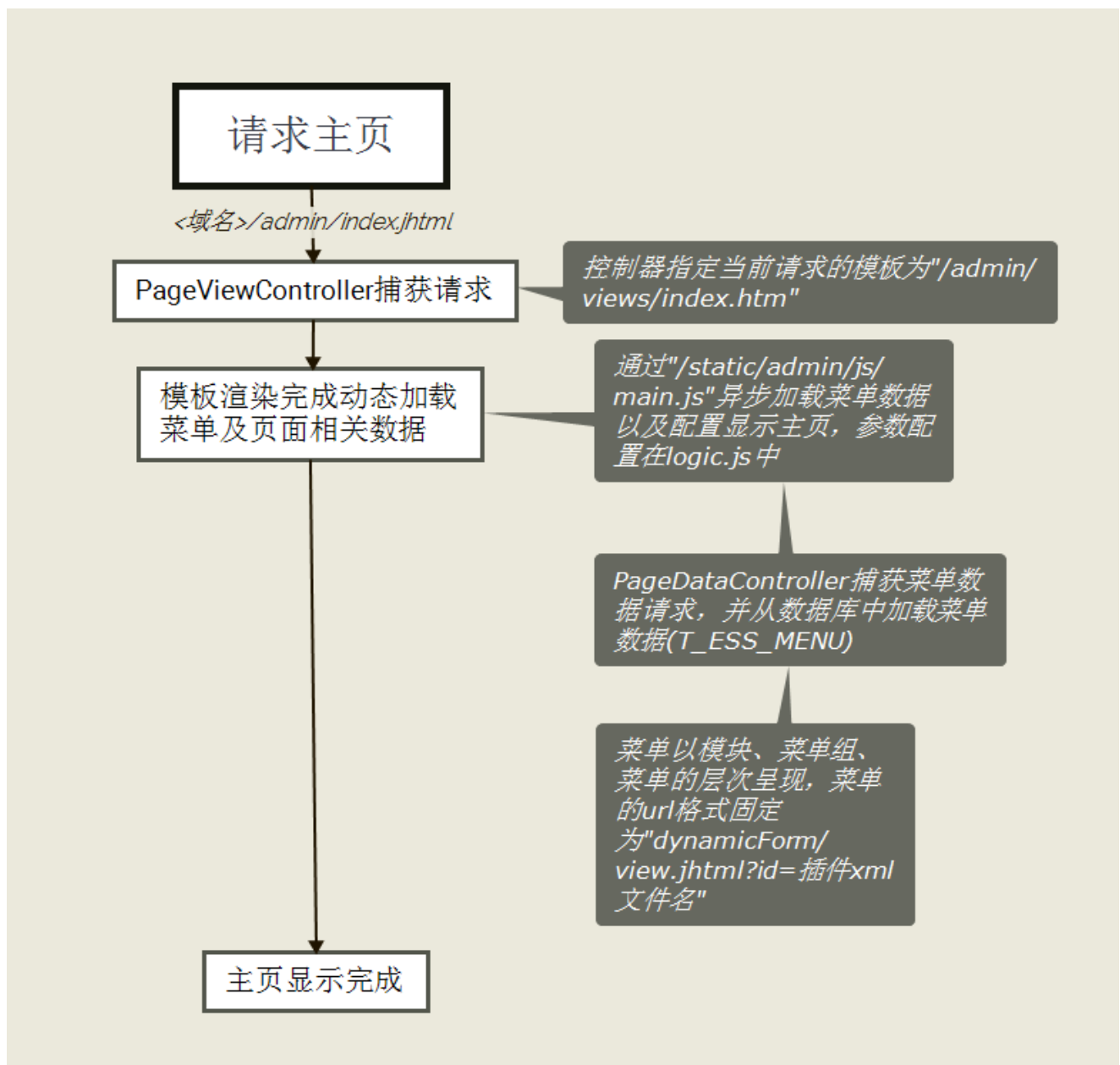
## 目录

- [后台主页请求流程详解](#)
- [dao层](#)
- [组件](#)
  - [tree树形组件](#)
- [页面插件开发](#)
  - [插件开发xml约束/智能提示文件配置](#)
  - [listView列表](#)
    - [列表开发](#)
    - [页面组件说明](#)
    - [datagrid.js简介](#)
  - [billView单据](#)
    - [单据开发](#)
    - [页面流程](#)
  - [dynamicForm.js介绍](#)
    - [dyform.callEventService\(event.type\)执行流程](#)

**JFS**框架项目文档记录

## 后台主页请求流程详解

---



## dao层

- DaoSupport中的save方法会数据新增返回的主键绑定到入参的dynamicObject中，引用传递，即之后可直接通过dynamicObject.getId()获取之前新增数据的主键。方法如下所示：

```
public DynamicObject save(WebContext webCtx, DSql sql,
    DynamicObject dynamicObject) {
    KeyHolder keyHolder = new GeneratedKeyHolder();
    getDbCtx().update(sql.toSQLString(webCtx),
        new MapSqlParameterSource(dynamicObject), keyHolder);

    dynamicObject.setId(Long.valueOf(keyHolder.getKey().longValue()));
    return dynamicObject;
}
```

# 组件

## tree树形组件

使用 `<tree>` 标签，父标签为 `datagrid`，可以单独配置子标签 `node`。

但若配置了 `url` 参数属性则无需再配置 `node` 子标签，会通过 `url` 后台异步获取json格式数据。

`url` 属性统一配置为 `/dynamicForm/treeList.jhtml`。

后台会获取参数 `treeType`，值为xml文件中 `tree` 标签的 `id` 属性值

## 页面插件开发

本系统的页面开发使用xml配置文件的方式，将html元素组件化为xml配置文件中的标签。

页面请求使根据id参数，找到Admin.WebApp工程下的etc/admin/views/system目录下文件名对应的xml文件。

程序在解析xml文件时，会根据每一个标签找到对应的组件转换器，转换器搭配对应的freemarker组件模板文件，渲染成html字符串。

解析过程会根据xml文件dom树的深度来进行递归操作。

## 插件开发xml约束/智能提示文件配置

所有xml文件头增加如下配置

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pageview SYSTEM "http://xd.k3cloud.kingdee.com/dtd/JFS.dtd" >
```

配置成功后便会有系统标签提示，类似于ibatis/mybatis的配置文件开发时的智能提示，比较方便。

## listView列表

### 列表开发

- 标准格式

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pageview SYSTEM "http://xd.k3cloud.kingdee.com/dtd/JFS.dtd" >
<pageview type="listView" name="名称" plugin="指定ListViewPlugin的子类"
  extend="IListView"(可选) editview="当前列表的数据编辑页面的xml文件名">
  <datasource>
    <querylist>配置当前列表的数据查询sql，框架自动完成对数据的检索/分页
  等</querylist>
  </datasource>
  <view>
    <toolbar id="toolbar" target="datagrid">
      工具栏按钮/按钮组<menubutton>/<menubuttongroup>
    </toolbar>
    <listdatagrid id="listentry">
      <filter id="filter">搜索框
        <searcher id="defaultsearch"
          placeholder="输入名称、标号" name="查询" width = "100"/>
        </filter>
        <datagrid id="datagrid" showTree="true">
          <tree id="userType" title="用户分类" rootName="全部">
            <node id="b2b" value="1" name="B2B用户"/>左侧导航树
            <node id="b2c" value="2" name="B2C用户"/>
          </tree>
          <fields>表格列
            <text id="number" name="编号"/>
            <text id="userName" name="用户名"/>
            <text id="realName" name="真实姓名"/>
            <text id="userType" name="用户类型"/>
            <checkbox id="status" name="禁用状态" defaultvalue="1"/>
          </fields>
        </datagrid>
      </listdatagrid>
    </view>
  </pageview>

```

## \* 标签说明

### 根节点

#### 各参数说明

必选: **type**-> `listView` 插件类型, **name**->顶部显示名, **plugin**-> `ListViewPlugin`的子类

可选: **extend**-> `IListView` 指定此属性将会继承父xml的全部但是可覆盖, **editview**->指定双击表格行/点击新增按钮时跳转到的单据 页面

### datasource

指定默认列表显示数据的查询sql, 配置了正确的查询sql之后会调用 `CoreService` 中的

`findDefault` 查询数据并显示

且实现了分页/过滤等, 过滤需要手动重写 `ListViewPlugin` 的

`buildQueryFilter(webCtx, view, param)` 方法

## toolbar

放置列表表格对应的工具路栏，以`target`属性指定表格id

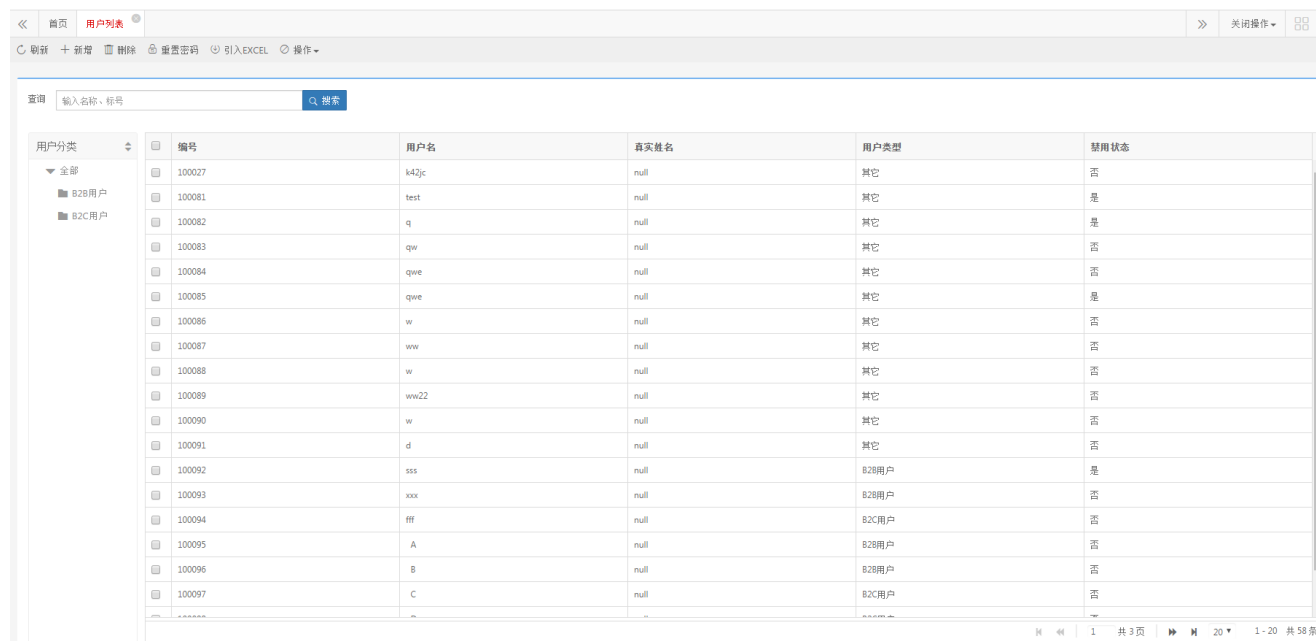
子标签可有 `menubuttongroup` 或 `menubutton`

`menubuttongroup`以下拉的形式显示，内部可放置`menubutton`

`menubutton`与`menubuttongroup`可指定`icon`属性来显示图标，`icon`属性对应于 `iconFactory` 中绑定的图标内容

`menubutton`通过指定 `iservice` 来绑定调用后台 `toolbarClick` 方法内部指定的事件

页面显示如下所示



用户分类	编号	用户名	真实姓名	用户类型	禁用状态
全部	100027	k4jpc	null	其它	否
828用户	100081	test	null	其它	是
82C用户	100082	q	null	其它	是
	100083	qw	null	其它	否
	100084	qwe	null	其它	否
	100085	qwe	null	其它	是
	100086	w	null	其它	否
	100087	ww	null	其它	否
	100088	w	null	其它	否
	100089	ww22	null	其它	否
	100090	w	null	其它	否
	100091	d	null	其它	否
	100092	sss	null	828用户	是
	100093	xxx	null	828用户	否
	100094	fff	null	82C用户	否
	100095	A	null	828用户	否
	100096	B	null	828用户	否
	100097	C	null	82C用户	否

## 流程简介

- 所有的列表插件类均继承自 `ListViewPlugin`，列表与单据插件的共同顶级父类为 `DynamicFormPlugin`
- 列表加表格完成后，会刷新表格，见 `datagrid.htm` 中 `.refresh()` 方法
- `datagrid.js` 中配置的 `refresh` 方法，将 `loaddata` 命令传回后台
- 后端统一入口为 `/dynamicForm/callService.jhtml`，用于获取后台命令以及回传数据
- 调用的处理方法为

`((DynamicFormPlugin) plugin).callEvent(this.webCtx, view, new ClientEvent(event));`，实质回调当前插件配置的具体实现类中的对应方法，因具体插件的父类 `ListViewPlugin` 以及顶级父类 `DynamicFormPlugin` 都做了一些相应的操作，所以在操作具体的插件类时，可以节省一些工作，注意类及方法间的调用关系

- `DynamicFormPlugin` 中 `loaddata` 事件调用的对应方法为 `bindData`，在 `ListViewPlugin` 中实现
- `ListViewPlugin` 中 `bindData` 方法会判断xml配置文件是否配置了 `dataSource` 标签且有查询sql语句，若存在则执行数据查询，并拼接 `buildQueryFilter` 方法进行数据获取
- 如果xml没有配置查询sql，则会同时调用 `buildQueryFilter` 与 `onDataLoad` 方法进行数据处理
- 在返回数据之前调用 `beforeBindData` 方法

所以一般情况下，具体列表插件实现类只需要重写**buildQueryFilter**方法与**onToolbarClick**模板代码示例

```
@Override
public QueryFilter buildQueryFilter(WebContext webCtx, ListView view,
    IHashMap queryParams) {
    QueryFilter queryFilter = new QueryFilter();
    String queryStr = queryParams.getString("defaultsearch");
    if(queryStr != null && !"".equals(queryStr)){
        queryFilter.appendFilter("搜索条件");
    }
    String treeFilter = queryParams.getString("treeFilter");
    if(null != treeFilter && !"".equals(treeFilter)){
        queryFilter.appendFilter("树形导航过滤条件");
    }
    return queryFilter;
}

@Override
public void onToolbarClick(WebContext webCtx, ListView view,
    ToolbarClientEvent event) {
    List<SelectedRow> selectedRows = event.getSelections();
    switch (event.getId()){
        case "delete"://删除操作
            view.showMessage("数据删除成功!");
            break;
        default :
            break;
    }
    view.refresh();
}
```

**buildQueryFilter**方法简介:

1. **buildQueryFilter**用于拼接sql查询时的一些过滤条件
2. 所有过滤条件拼接到变量**filter**
2. 可以对数据搜索以及左侧导航树的条件进行过滤来获取目标结果
3. 过滤条件可以通过**queryParams**拿到
4. 搜索条件为**searcher**标签指定的id，即**queryParams.getString(id)**
5. 树形导航过滤条件通过**queryParams.getString("treeFilter")**获取,值为Node的value属性

**onToolbarClick**方法简介:

1. 用于操作工具栏按钮点击事件的回调
2. **event.getSelections()**获取当前表格选中的行
3. 对**view**的各种操作，会执行**OperationProxy.execCommand**保存相应的命令
4. **view**对象的顶级父类**ViewProxy**中的变量**operationProxy**会保存相应的操作

## 页面组件说明

- 每个组件如果需要兼容列表都需要在组件中添加如下：

```
<#if (control.inlinedatagrid!='false')== 'false'>
//组件html文本
<#else>
var editControl${control.id}_${pageId} = function(value, options){
    return customEditElement("${pageId}", "${control.type}", value, options);
}
</#if>
```

- 其中调用 `customEditElement` 方法存在于 `datagrid.js`

## datagrid.js简介

- `customEditElement`用于获取填充在datagrid中的html渲染代码，会根据组件的type获取组件的html代码，在单元格聚焦时显示编辑状态，即可编辑表格。如果需要扩展，添加switch case
- `customValue`用于绑定组件最新的值到表格cell,如input更改后的值，失焦后绑定到cell，如果需要扩展，添加switch case
- `formatter`用于初始化页面时绑定对应组件的值到对应的单元格，如果需要扩展，添加switch case

## billView单据

### 单据开发

- 

### 页面流程

- 页面初始化时，调用dynamicform.js中的billView下的 `initEvent`，若是新增页面，则在保存时会将页面组件的数据组装好用于传回后台，编辑页面则将后台既定格式数据绑定到各个组件上显示
- 从列表双击跳转到编辑页面，调用 `billView.bindData(pageId,dataObj)` 绑定数据

### 点击保存按钮流程

- dyform.init中已绑定的 `$('#content').on('click', ".toolbar a,.op-btn a",function(event){});` 事件
- 事件触发后调用 `dyform.callEventService(o,"onToolBarClick");`

详情查看：[callEventService方法执行细节](#)

3. 在callEventService方法中调用billView.initEvent取到当前页面中组件的值
4. 取值操作在\$.customerVal()中进行，组件type分别调用对应的组件赋值/取值方法，如果增加组件，也是在这个方法中进行扩展取值/赋值方法
5. 如果有分录，则调用 `billview.billViewGrid` 生成从表表格数据

## dynamicForm.js介绍

### dyform.callEventService(event,type)执行流程

1. 调用 `dyform.initEvent(event)`；处理 `event` 参数
2. 调用 `$.delJson(json,paramKey)` ,删除json中的paramKey对应的数据
3. 调用 `$.splitId()` 分割包含\_的字符串，获取 `controlId` 、 `pageId`
4. 将必须参数组装到event，包括optarget、pageId、id(controlId)、type(当前事件类型)、pageSessionId
5. 调用 `$.getPageType(pageId)` 获取当前页面的类型，属于listView或billView
6. 根据当前页面类型，分别调用listView.initEvent(event,pageId,type)与billView.initEvent(event,pageId,type)

#### billView.initEvent(event,pageId,type)

参数示例

event -> 封装基本参数信息

```
id:"save"
optarget:""
pageId:"bj8GFxEF8H"
pageSessionId:"R29vZHNTS1VFZGI0Vmllldw=="
type:"onToolBarClick"
```

pageId -> 页面唯一标识

```
pageId="bj8GFxEF8H"
```

type -> 当前事件类型

```
type="onToolBarClick"
```

执行细节

1. 调用eval("Form\_"+pageId)获取当前单据表单内的组件的结构
2. 遍历当前结构，调用\$.customerVal(组件id,组件type,pageId),在方法内部会根据组件type调用对应组件的取值/赋值方法 `$.*Val(id,pageId,value)`
3. 调用billView.getPkValue(pageId)获取当前单据的主键值
4. 将数据转变成json字符串并绑定到event的dataobject属性上



