|  |  |
| --- | --- |
| 机密 | |
| **密级** | **一** |

**金证股份 • KJDP3.5平台**

**（Kingdom Java Develop Platform）**

**应用及开发指引**

|  |  |  |  |
| --- | --- | --- | --- |
| **研制单位** | 深圳市金证科技股份有限公司 | | |
| **文档名称** | 金证kdjp3.5平台应用及开发指引 | | |
| **文档编号** |  | **文档版本** | 1.0 |
| **文档状态** | 🞏草稿 🗹 完成 | | |
| **审 批** |  | **审批日期** |  |

**修改记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文档版本** | **文档状态** | **修订日期** | **修订作者** | **修订内容** |
| 1.0 | C | 2014-03-04 | 林毅 | 新编制 |
| 1.1 | M | 2014-03-06 | 方晓丹 | 补充内容 |
| 1.2 | M | 2014-03-06 | 林毅 | 补充 |
| 1.3 | M | 2014-03-10 | 方晓丹 | 补充 |
| 1.4 | M | 2014-03-20 | 方晓丹 | 添加阅读指南。  修改2.3开发方法、3.后台业务开发、4前台页面开发等。  主要包括：3.2.2.2VIEW结合KUI使用说明章节；3.3.2.2service语法说明及例子；3.4.2.2bex语法说明及例子 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

文档状态标识：C – Created A – Added M – Modified D – Deleted

**阅读指南**

|  |  |
| --- | --- |
| **人员／部门** | **建议阅读内容** |
| 平台开发人员 | 开发方法重点关注2.3开发方法、3后台业务开发、4前台页面开发章节所描述的开发详解 |
| 业务系统  开发人员 | 原子业务开发重点关注3.4原子业务章节。  流程引擎开发重点关注5业务流程设计章节。  门户引擎开发重点关注6门户设计章节 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

目录

[1. 引言 7](#_Toc383419755)

[1.1 目的和范围 7](#_Toc383419756)

[1.2 术语和定义 7](#_Toc383419757)

[2. 概述 8](#_Toc383419758)

[2.1 技术架构 8](#_Toc383419759)

[2.2 系统原型 9](#_Toc383419760)

[2.3 开发方法 10](#_Toc383419761)

[2.3.1 程序体系及开发步骤 10](#_Toc383419762)

[2.3.2 开发组织 11](#_Toc383419763)

[2.3.3 开发流程及过程管理 13](#_Toc383419764)

[2.4 业务原子 13](#_Toc383419765)

[2.5 原型展望 14](#_Toc383419766)

[2.6 开发环境 14](#_Toc383419767)

[3. 后台业务开发 16](#_Toc383419768)

[3.1 后台业务开发模型 16](#_Toc383419769)

[3.1.1 业务原子化 16](#_Toc383419770)

[3.1.2 业务编排 16](#_Toc383419771)

[3.1.3 辅助工具 16](#_Toc383419772)

[3.2 业务视图(VIEW) 17](#_Toc383419773)

[3.2.1 业务视图概述 17](#_Toc383419774)

[3.2.2 VIEW使用说明 17](#_Toc383419775)

[3.3 服务层处理(Service) 21](#_Toc383419776)

[3.3.1 服务层概述 21](#_Toc383419777)

[3.3.2 Service使用说明 21](#_Toc383419778)

[3.3.3 服务请求对象 22](#_Toc383419779)

[3.3.4 服务结果集 23](#_Toc383419780)

[3.4 业务引擎对象(BEX) 23](#_Toc383419781)

[3.4.1 Bex概述 23](#_Toc383419782)

[3.4.2 Bex配置说明 24](#_Toc383419783)

[3.5 原子业务(ATOM) 26](#_Toc383419784)

[3.5.1 原子业务概述 26](#_Toc383419785)

[3.5.2 原子业务开发说明 28](#_Toc383419786)

[4. 前台页面开发 33](#_Toc383419787)

[4.1 前台开发概述 33](#_Toc383419788)

[4.2 页面开发 33](#_Toc383419789)

[4.2.1 View配置 33](#_Toc383419790)

[4.2.2 Html开发 33](#_Toc383419791)

[4.3 Javascript开发 34](#_Toc383419792)

[4.4 CSS开发 36](#_Toc383419793)

[4.5 KUI组件使用 36](#_Toc383419794)

[4.5.1 使用方法 36](#_Toc383419795)

[4.5.2 基本组件 38](#_Toc383419796)

[4.5.3 布局管理器组件 52](#_Toc383419797)

[4.5.4 菜单与按钮组件 71](#_Toc383419798)

[4.5.5 表单组件 80](#_Toc383419799)

[4.5.6 窗口组件 92](#_Toc383419800)

[4.5.7 数据表格和树形菜单组件 102](#_Toc383419801)

[5. 业务流程设计 124](#_Toc383419802)

[5.1 Web流程设计器 124](#_Toc383419803)

[5.1.1 设计器入口 124](#_Toc383419804)

[5.1.2 设计器使用 125](#_Toc383419805)

[5.1.3 流程节点属性 126](#_Toc383419806)

[6. 门户设计 128](#_Toc383419807)

[6.1 门户对象说明 128](#_Toc383419808)

[6.1.1 组件 128](#_Toc383419809)

[6.1.2 容器 128](#_Toc383419810)

[6.1.3 布局 128](#_Toc383419811)

[6.2 门户界面设计 129](#_Toc383419812)

[6.2.1 门户界面说明 129](#_Toc383419813)

[6.2.2 门户设计使用 129](#_Toc383419814)

[7. 业务系统通用模块 132](#_Toc383419815)

[7.1 参数管理 132](#_Toc383419816)

[7.1.1 系统参数管理 132](#_Toc383419817)

[7.1.2 数据字典管理 133](#_Toc383419818)

[7.1.3 系统菜单管理 134](#_Toc383419819)

[7.1.4 系统序号设置 135](#_Toc383419820)

[7.1.5 国际化管理 135](#_Toc383419821)

[7.1.6 系统服务流水管理 136](#_Toc383419822)

[7.1.7 系统服务管理 137](#_Toc383419823)

[7.1.8 系统图标速查表 138](#_Toc383419824)

[7.1.9 系统错误速查表 138](#_Toc383419825)

[7.2 操作员管理 139](#_Toc383419826)

[7.2.1 操作员信息管理 139](#_Toc383419827)

[7.2.2 岗位管理 140](#_Toc383419828)

[7.2.3 组织机构管理 141](#_Toc383419829)

[7.2.4 用户授权管理 141](#_Toc383419830)

[7.3 权限管理 142](#_Toc383419831)

[7.3.1 菜单授权 142](#_Toc383419832)

[7.3.2 临时授权 142](#_Toc383419833)

[7.3.3 岗位分配 142](#_Toc383419834)

[7.3.4 权限信息管理 144](#_Toc383419835)

[7.4 流程管理 144](#_Toc383419836)

[7.4.1 流程发布管理 144](#_Toc383419837)

[7.4.2 发起流程管理 148](#_Toc383419838)

[7.4.3 待办任务管理 150](#_Toc383419839)

[7.4.4 已办任务管理 153](#_Toc383419840)

[7.4.5 代理任务管理 154](#_Toc383419841)

[7.4.6 流程实例管理 155](#_Toc383419842)

[7.4.7 流程模型管理 156](#_Toc383419843)

[7.5 门户设计 162](#_Toc383419844)

[7.5.1 布局管理 162](#_Toc383419845)

[7.5.2 容器管理 163](#_Toc383419846)

[7.5.3 组件管理 164](#_Toc383419847)

[7.5.4 运行平台管理 165](#_Toc383419848)

[8. 开发环境 166](#_Toc383419849)

[9. KJDP开发工程配置 167](#_Toc383419850)

[9.1 安装JDK及配置 167](#_Toc383419851)

[9.2 安装Tomcat及配置 167](#_Toc383419852)

[9.3 升级浏览器版本至IE8以上版本 168](#_Toc383419853)

[9.4 安装开发工具 168](#_Toc383419854)

[9.5 安装数据库 168](#_Toc383419855)

[9.6 安装FireFox和FireBug及配置 168](#_Toc383419856)

[9.7 前端功能调试 169](#_Toc383419857)

[10. 编程规范 175](#_Toc383419858)

[10.1 版面书写规则（JAVA） 175](#_Toc383419859)

[10.2 命名规则（java） 176](#_Toc383419860)

[10.3 Mybaties sql书写规则 178](#_Toc383419861)

[10.4 注释描述规则 179](#_Toc383419862)

[10.5 程序开发约定 180](#_Toc383419863)

[11. 附注 184](#_Toc383419864)

[11.1 KJDP平台发展过程 184](#_Toc383419865)

# 引言

## 目的和范围

编写此指引旨在阐述**金证KJDP3.5平台**的技术架构、系统原型、开发管理、开发方法以及开发规范等方面内容，为使用此平台的开发项目组和技术人员提供指引，并规范开发及维护的过程。

此指引适用使用**金证KJDP3.5平台**开发的业务系统以及衍生业务系统的开发过程及维护过程。

## 术语和定义

**KJDP**：金证J2EE开发平台(Kingdom J2EE Development Platform)

**Controller**：平台控制器入口层，接收http/https请求转化为内部请求调用内部服务完成处理

**Service**：平台服务层，结合服务配置工厂，完成服务的识别，提供内部、外部接口

**Atom**：原子业务，最小独立业务单元，完成独立、完整的业务操作

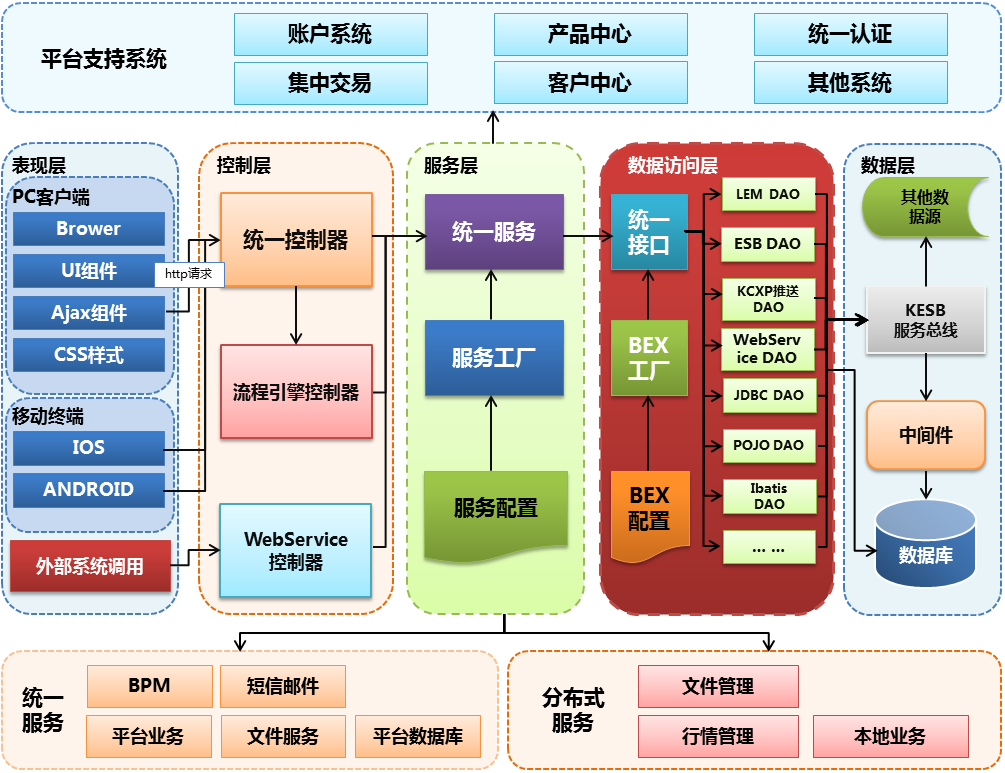
**Bex**：业务数据引擎，封装多数据源访问，通过统计接口提供到原子业务层、服务层调用

# 概述

KJDP3.5平台是定位构建流程化工作门户，将业务通过流程串联完成业务操作，从技术提供由菜单驱动方式往流程驱动的系统平台。

该平台延续了KJDP3.0的架构理念，具有低耦合、易部署的特点；采用内存数据库、分布式部署、业务原子化等核心技术，系统开发层次清晰，配置定义灵活，降低业务开发门槛，引入流程引擎将系统业务流程化管理，实现由菜单驱动转到流程驱动的新工作模式。

## 技术架构



* 表现层：表现层的主要功能是将用户接入到系统，通过PC终端或移动终端程序展示系统界面及系统功能。PC终端表现层采用MFC程序嵌套HTML、CSS、JAVASCRIPT进行构建；移动终端采用APP和安卓平台原生程序嵌套HTML、CSS、JAVASCRIPT进行构建部分移动端效果通过原生程序完成。表现层的数据请求通过AJAX方式提交到控制层。
* 控制层：控制层收到客户端请求完成统一字符集、会话验证等转换工作，如果请求属于流程处理类请求会提交到流程引擎控制器，流程引擎将调用服务层的服务进行业务处理，控制器将请求的SERVICE 和其他参数传递到服务层，进行处理。
* 服务层：服务层收到请求后，服务层通过java代码进行原子服务封装，支持原子服务继承、组合形成新的原子服务，根据配置决定对客户端开放部分服务，本身原子服务设计采用完全松耦合，服务间交互通过内存交互区实现。提供原子服务组装器支持服务器端javascript语言脚本组装服务并进行逻辑控制。
* 数据访问层：数据访问层提供LBM、ESB、KCXP、WEBSERVICE、JDBC、POJO、IBATIS 等数据访问通道，将数据请求在数据库、中间件、其他数据源中进行处理。
* 数据层是主要的数据源提供者，包含数据库DB 、企业服务总线(KESB)、金证中间件(KCBP/KCXP)等。

## 系统原型

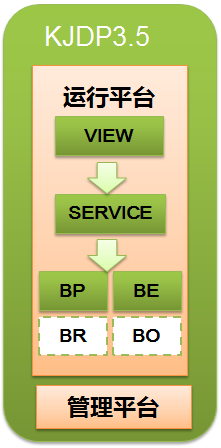
KJDP3.5分为两部分：管理平台和运行平台。

**管理平台**主要实现了公司业务系统的通用的基础模块及业务系统开发基础开发框架组件，包括：参数管理、员工管理、权限管理、流程管理、门户管理等等。这些业务公共模块在设计和实现上力求尽量与业务无关和尽量通用，各业务公共模块相互独立，可以根据具体业务系统的需求进行裁减。通过业务公共模块的支持，尽量使得业务系统的研发工作重点放在业务设计和业务实现上，减少和避免基础性和重复性的开发。

**运行平台**主要是提供给业务人员使用。运行平台会根据管理员在管理平台的设置动态加载相应的功能模块。运行平台的模块整合的是业务人员常用的一些功能模块。该平台的特点是简单，易用，实用。

## 开发方法

### 程序体系及开发步骤



开发步骤一般采用如上图所示的顺序：

1. 开发VIEW业务视图(即前端页面)。业务视图有2种开发方法：

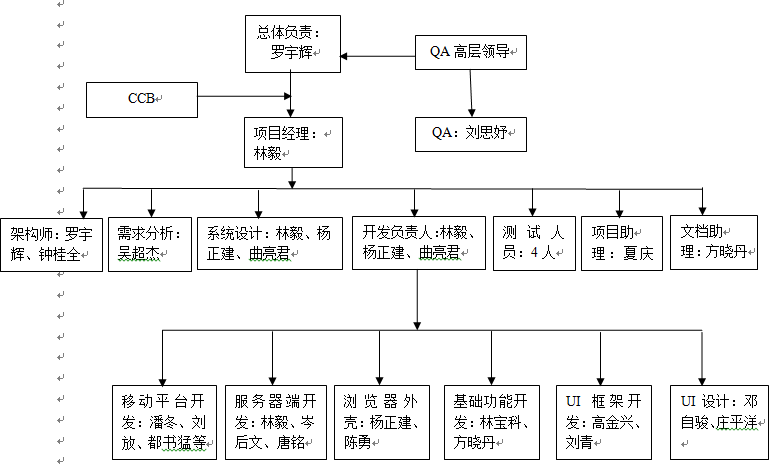
配置VIEW文件

编写HTML文件

1. 开发服务层SERVICE。服务层一般是配置service文件
2. 开发业务引擎对象BEX(对应上图的BP、BE、BO、BR、BA)。业务引擎对象一般是配置bex文件
3. 开发数据访问层。数据访问层一般是配置mybatis文件，写sql语句。

以上步骤具体开发方法见下文后台及前端的开发描述。这些步骤是最常用的开发方法，还有原子业务开发、流程设计器配置的开发也详见下文。

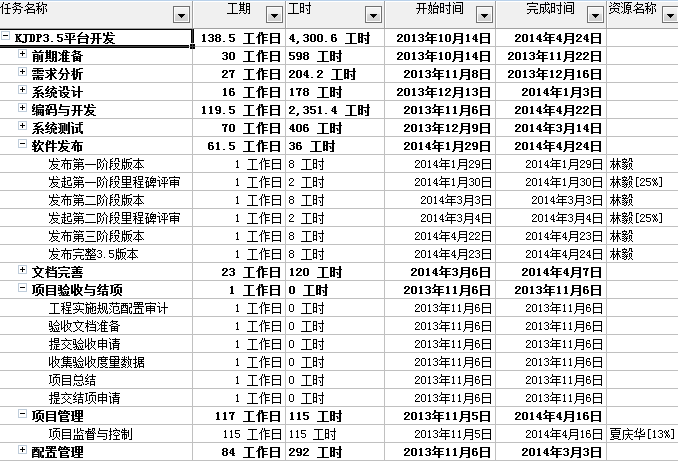
### 开发组织



|  |  |  |
| --- | --- | --- |
| 角色 | 人员 | 工作职责 |
| 总架构师 | 罗宇辉、钟桂全 | * 架构技术支持及关键问题决策； * 参与项目里程碑评审及其他关键评审工作。 |
| 项目经理 | 林毅 | * 负责本项目的管理工作，组织制定和完善项目计划，核查、监督项目计划的执行情况，进行项目的质量和风险管理； * 负责分配各小组的工作任务，督导各项目小组的工作进展； |
| 需求分析人员 | 吴超杰 | * 收集市场需求 * 研发部门需求 * 客户需求 * 完成需求分析文档 |
| 系统设计人员 | 曲亮君、杨正建、林毅 | * 平台需求分析说明书 * 平台概要设计 * 平台详细设计 * 技术框架选型 |
| 服务器端开发 | 林毅  岑后文  唐铭 | * 服务器端核心代码开发 * 原子业务引擎开发 * 工作流引擎开发 * 缓存管理开发 |
| UI框架开发 | 高金兴  刘青 | * 应用框架开发 * 门户界面开发 * 流程配置及流程展示界面开发 |
| 基础功能开发 | 林宝科、  方晓丹 | * 基础功能开发 * 系统管理功能 * 用户管理功能 |
| 浏览器外壳 | 杨正建  陈勇 | * 外壳基础、扩展架构开发 * 拍照、打印、读卡器、图片编辑、上传、消息、更新、数据缓存 |
| 移动架构师 | 潘冬 | * 移动端架构设计和开发 |
| 移动框架开发 | 刘放  都书猛 | * 移动框架开发 |
| 移动业务开发 | 钟大霆  杨彤 | * 移动功能业务开发 |
| UI设计 | 邓自骏  庄平洋 | * 客户端界面设计 * 移动UI设计 * 用户界面体验设计 |
| 测试人员 | 于章容  王艺萍  朱琳  朱兰兰 | * 负责制定测试计划； * 负责编写测试用例； * 负责测试环境的搭建和维护； * 负责验证系统开发组提交的集成版本的整体测试； * 进行测试总结并提交测试报告。 |
| 项目助理 | 夏庆华 | * 协助项目经理对项目实施过程进行监督与控制，包括项目范围、进度、质量、风险和问题等； * 协助项目立项、计划、执行、监控、收尾等过程执行； * 负责项目文档规范管理； * 编制项目周报、日报，组织项目会议等 * 协助项目经理协调项目中各项事务处理。 |
| 文档助理 | 方晓丹 | * 负责整理需求分析说明书、概要设计说明书、详细设计说明书、开发指引等开发各阶段文档整理 |

### 开发流程及过程管理

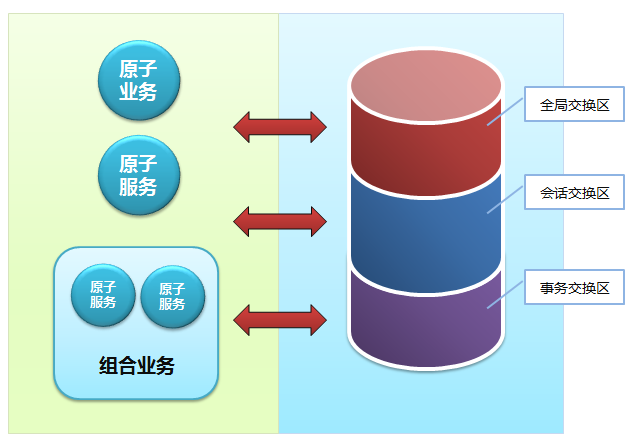
以下是开发团队各小组工作职能，及小组各阶段的工作重点示意图。项目开发将按照次计划进行，如有需要，会适当进行调整。



## 业务原子

本平台的主要特色之一是提供业务原子化的功能。业务原子化，是指将业务流程进行拆分，拆分成一个个最小的业务功能，即业务原子。在业务开发中，再将这些原子拼装成整体业务，实现完整的业务功能。

原子业务与数据交换关系如下：



## 原型展望

KJDP3.5希望应用到更多同类业务程序的研发项目组中去，不断的完善和积累（包括：程序部分和流程部分）。

## 开发环境

|  |  |
| --- | --- |
| 系统开发环境 | |
| 项目 | 描述 |
| 硬件平台 | 1G内存+，32位OS系统+ |
| 操作系统 | Windows、Linux、Aix、…… |
| 数据库系统 | 各操作系统平台的数据库服务及客户端软件，最低数据库版本： ORACLE 10g、PLSQL Developer8.0+ |
| 后台开发工具 | PLSQL Developer8.0+、Myeclipse8.5+ |
| 数据库设计工具 | PowerDesigner 12.5+ |
| 前端程序开发工具 | SpringSource Tool Suite2.7.1+、idea10.0+、editplus2.0+ |
| 代码管理工具 | SVN |

使用KJDP3.5通常建议建立多套开发和测试环境，建议开发组在Windows环境下开发，方便程序调试，测试组在Linux和windows server环境下测试，从保持对跨平台跨数据库的支持。

# 后台业务开发

## 后台业务开发模型

### 业务原子化

在KJDP3.5平台中，指的是不可分割的业务功能。将业务功能细分为最小的模块，即原子业务。在业务开发过程中，通过拼装成整体业务，实现需要的业务功能。以此提高系统的高可用性和减少业务开发人员的工作量。

### 原子业务框架

原子业务在形式上是由多个互不相关的JAVA类组成，继承了统一的原子业务父类，通过统一的业务工厂类动态注册到服务器中供服务接口调用。

业务工厂类是是原子业务的框架，负责原子的动态装载及服务接口的调用，是基础框架层与原子业务类的纽带，串联原子业务为服务层调用提供支持。

数据缓冲区是原子业务框架的重要组成部分，为原子业务之间提供数据传递、数据共享的空间，解除了原子业务互相访问的耦合问题。数据缓冲区分为三个级别：全局数据缓冲区、服务数据缓冲区、业务数据缓冲区；三个数据缓冲区的生命周期各有不同。在多个服务中共享数据采用全局缓冲区；在服务内部的多个原子业务间共享数据采用服务数据缓冲区；在单个原子业务方法中采用业务缓冲区。

### 服务逻辑编排

服务中会包含一个或多个(Atom原子业务/Bex业务访问)配置，多个业务配置默认采用串行执行。在特殊场景下也可以支持业务逻辑的编排，业务逻辑编排使用groovy语言的语法，可进行分支if,循环for等逻辑判断完成实际业务场景。

<service service\_code="atomTest" service\_name="开户"

service\_type="1" service\_cate="1" arrange\_type="1" soap\_enable="1"

attr1="">

**<script><![CDATA[**

**def ret;**

**if ("$MODEL\_KEY" == "lytest")**

**{**

**ret = doCall("TestAtom2");**

**}**

**return ret;**

**]]></script>**

<businesses>

<service\_business business\_code="TestAtom1"

business\_name="TestAtom" business\_method="doUserQuery" serial\_no="1"

business\_type="3" business\_cate="1" attr1="" />

<service\_business business\_code="TestAtom2"

business\_name="TestAtom2" business\_method="doQuery2" serial\_no="1"

business\_type="3" business\_cate="1" attr1="" />

</businesses>

</service>

以上模拟代码中通过脚本编排，完成对入参的访问，判断服务入参MODEL\_KEY如果等于lytest那么服务调用会跳过第一个原子业务，而直接支持第二个原子业务。

### 逻辑结构

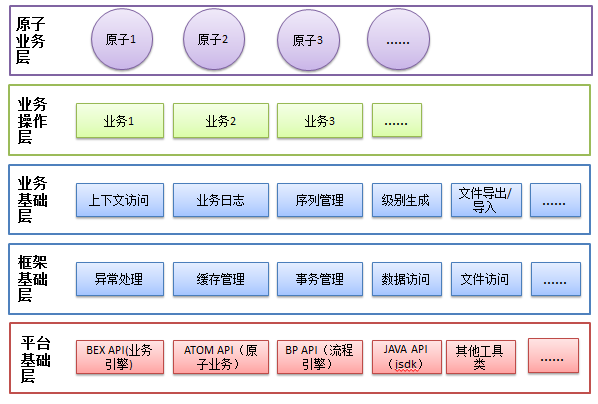


图 2-1 程序逻辑图

服务端程序按逻辑分五层：原子业务层、业务操作层、业务基础层、框架基础层、平台基础层.

以上结构是指在开发过程中也需要把代码按照该逻辑分开？

### 层次结构

* 平台基础层

由平台提供的API、开源工具类库、内部封装的类库等，以封装为类库的方式被其他模块进行调用，封装在AtomAbstract抽象类文件中。工具类库包括调用bex API、原子业务API、流程引擎API、JAVA API、第三方工具包等。

* 框架基础层

通过平台基础类派生的框架基础类BaseAtom类，继承平台基础层功能外，提供异常处理、缓存对象管理、业务间事务管理、数据访问、文件访问、异步处理等基础功能。

* 业务基础层

通过框架基础类派生的业务基础类BizAtom类，继承上层功能外，提供权限控制、业务日志、序列管理、级别生成、文件导入/导出等功能支持。

### 调用规范

服务端编程的各层对象设计时需遵循以下调用规范：

1. 原子业务层不能直接进行数据访问，需要通过业务操作层提供的方法封装进行访问。
2. 业务操作层尽量安装业务模型进行划分，如客户联系资料模块就会有一个联系资料的模型类，一个联系资料的业务操作类，通过原子业务包装提供到服务层调用。
3. 原子业务方法间调用需要通过框架基础层提供的统一访问接口进行访问，不可直接互相调用。
4. 原子业务层、业务操作层不可直接调用平台基础层接口，需要通过框架基础层/业务基础层封装的接口间接访问。

### 核心类图



图2-2 核心类图

如上图所示，AtomAbstract(平台基础层上)抽象类是所有原子业务类的基类，BaseAtom(框架基础类)继承AtomAbstract实现框架基础层支持,BizAtom(业务基础类)继承BaseAtom实现业务框架层支持。

## 业务视图(VIEW)

### 业务视图概述

业务视图是展示界面配置的方法，通过XML配置和框架公用页面函数，完成标准表格、表单、树等界面组件的构建，同时支持多个表格关联展示。

### VIEW使用说明

#### 配置规范

文件目录：所有VIEW配置文件必须存放在webapp\WEB-INF\conf\metaData\viewConf目录

命名规范：模块名-view.xml

例如，新建一个用户管理的view文件，则命名为：uum-view.xml。

#### 页面使用

1、公共参数界面

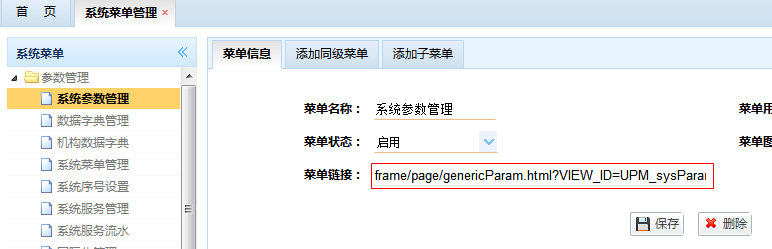
需要使用公共参数页面对VIEW配置进行解析和展示，公共参数界面路径如下webapp/frame/page/genericParam.html

2、菜单配置：

在系统菜单管理中，填写菜单信息后，加上配置文件路径及view的ID，路径为frame/page/genericParam.html?VIEW\_ID=UUM\_operInfoManage。

UUM\_operInfoManage为对应的VIEW的id，**并且要和view文件的前缀一致.**

设置路径如下：



3、其他说明：

VIEW配置命名必须使用模块名\_功能名，如果没有“\_”公共参数界面将不会解析。

#### VIEW配置详解

**View XML 标签是配合KUI组件共同使用。**

**使用说明：**

* <view></view>标签是一个大容器，首先是设置这个容器使用了哪一个KUI组件。目前这个大容器可以使用的KUI组件有：datagrid、treegrid、form、tree。
* 配置好view容器，再在该容器里面使用KUI组件。可以使用但不限于这些的组件有：input、combobox、combotree、datebox 、numberbox、textarea、validatebox、combo、timespinner等等

**View中使用KUI组件方法如下：**

1.**使用某个组件**：例如combotree，首先要把type设置为combotree

2.**使用该KUI组件中的属性：**要在每一个属性前面加上edt\_。例如设置下拉 panel 宽度，原来的属性是panelWidth，加上edt\_即edt\_ panelWidth ="230"。其他属性以此类推。

**以下是代码例子。代码例子后会针对每一个标签属性进行说明。**

<view id="UUM\_operInfoManage" render="datagrid" title="员工资料管理" script="../../business/js/manage-platform-func.js">

<conf service="userInfoQuery" colNumbers="2" pageSize="10" pagination="true" fit="true"/>

<qry\_item field="USER\_CODE" title="员工号" type="input" edt\_validType="zint[18]"/>

<qry\_item field="USER\_TYPE" title="员工类型" type="combobox" edt\_dict="USER\_TYPE" edt\_extItems="[{'dict\_des':'全部','dict\_val':''}]"/>

<btn\_item text="新增" service="userInfoInsert" title="新增员工信息" handler="userInfoAdd" iconCls="icon-add"/>

<col\_item field="USER\_CODE" title="员工号" type="input" width="60" edt\_width="140" edit\_flag="2"/>

<col\_item field="USER\_NAME" title="员工名称" type="input" width="100" edt\_width="140" edt\_required="true" edt\_validType="val[1,32]" />

<col\_item field="USER\_TYPE" title="员工类型" type="combobox" width="60" edt\_width="144" edt\_dict="USER\_TYPE" edt\_required="true" />

<col\_item field="ORG\_CODE" title="机构名称" edt\_required="true" edt\_width="140" width="100"

type="combotree"

edt\_req="[{service:'orgQuery'}]"

edt\_panelWidth="250"

edt\_panelHeight="200"

edt\_nodeId="ORG\_CODE"

edt\_nodeName="ORG\_NAME"

edt\_parNode="PAR\_ORG"

edt\_treeType="1"

/>

<col\_item field="POST\_ID" title="员工岗位" edt\_required="true" edt\_width="140" width="100"

type="combobox"

edt\_req="[{service:'postQueryAll'}]"

edt\_panelHeight="200"

edt\_valueField="POST\_ID"

edt\_textField="POST\_NAME"

/>

<col\_item field="OPEN\_DATE" title="开户日期" type="input" width="80" edt\_width="140" edit\_flag="3"/>

<col\_item field="OFF\_TEL" title="办公电话" type="input" width="120" edt\_width="140" edt\_validType="tel"/>

<col\_item field="MOBILE" title="移动电话" type="input" width="120" edt\_width="140" edt\_validType="mobile"/>

<col\_item field="EMAIL" title="电子邮箱" type="input" width="150" edt\_width="140" edt\_validType="email"/>

</view>

**注意：以上代码中列举了较为常用的组件和属性，注意粗体部分和下划线部分。**

* **<view>标签属性说明**

|  |  |  |
| --- | --- | --- |
| **属性(是否必填)** | **属性说明** | **缺省值** |
| view(Y) | view配置标签 | 无 |
| id(Y) | 唯一表示一个view | 无 |
| render(Y) | 引用KUI组件，包括datagrid、treegrid、form、tree | 无 |
| view中的title(Y) | 组件标题 | 无 |
| script(N) | 引用用到的JavaScript文件，配置相对路径 |  |
| conf(Y) | 属性配置标签 | 无 |
| gridMenu（N） | 表格工具菜单 可用逗号分隔  恢复默认：restoreDefault  冻结列：freezeColumns,  隐藏列：hideColumns,  隐藏查询条件：hideCols,  导出文件：fileExport,  文件打印：printReport | 无 |
| service(Y) | service请求名称 | 无 |
| colNumbers(N) | 查询条件每一行的列数 | 2 |
| pageSize(N) | datagrid的属性，当设置分页属性时，初始化每页记录数。 | 10 |
| pagination(N) | datagrid的属性，设置true将在数据表格底部显示分页工具栏。 | false |
| fit(N) | 设置为true时表格高度自适应 | false |
| qry\_item(Y) | 查询条件配置 | 无 |
| qry\_item中的field(Y) | 对应查询字段在数据库中的列名 | 无 |
| qry\_item中的  title(Y) | 查询名称 | 无 |
| qry\_item中的type(Y) | 查询类型 | 无 |
| qry\_item中的edt\_validType(N) | 查询条件可以输入的类型 | 无 |
| edt\_dict(N) | 数据字典对应名称(一般用于配置combobox) | 无 |
| edt\_extItems(N) | 手动添加默认值(一般用于配置combobox) | 无 |
| btn\_item(N) | 配置按钮标签(看需要而定) | 无 |
| btn\_item 中的text(Y) | 按钮名称(有配置按钮标签时为必填) | 无 |
| btn\_item 中的service(Y) | 点击按钮后发的请求 | 无 |
| btn\_item 中的title(N) | 点击按钮如果是弹出框，即为弹出框的标题 | 无 |
| btn\_item 中的handler(Y) | 点击按钮触发的方法 | 无 |
| btn\_item 中的iconCls(N) | 点击按钮如果是弹出框，即为弹出框的标题前面的图标 | 无 |
| col\_item(Y) | 列标签 | 无 |
| field(Y) | 列对应数据库字段名称 | 无 |
| col\_item 中的title(Y) | 列标题 | 无 |
| col\_item 中的width(N) | 列宽度 | 自适应 |
| col\_item 中的edt\_width(N) | 这是使用了KUI组件的属性，上面代码使用了type为input或combotree，即该属性编辑宽度 | 自适应 |
| edit\_flag(N) | edit\_flag 为1：在弹出框中显示，但是可insert，不可update；  2：在弹出框中显示不可insert，不可update；  3：该列在弹出框中不显示 | 无 |
| edt\_validType(N) | 有这几种类型：  val[1,32]：表示只能输入字符串，并且长度为1到32；  length [1,32]：表示长度为1到32；  zint[8]：表示只能输入8位的正整数，不包括0；  int[8]：表示只能输入8位的整数，包括0；  email：只能输入邮箱类型；  mobile：只能输入手机类型；  tel：只能输入固定电话类型； |  |

## 服务层处理(Service)

### 服务层概述

服务层为请求调入入口，根据请求的SERVICE 参数，在SERVICE工厂中查找对应的配置，再根据配置中的SERVICE\_BUSINESS子标签进行业务请求，目前支持的业务类型BEX（业务接口）、BAX（原子业务）。

### Service使用说明

#### 文件路径

在WEB-INF\conf\metaData\serviceConf路径下新建一个.xml文件。

命名均要以-service结尾。例如，新建一个用户管理的service文件，则命名为：uum-service.xml。

#### service语法说明及例子

service可以调用原子业务和bex，调用原子业务部分参见3.5.2.2解析服务层配置。

service请求bex有两种情况：

1. 请求一个bex
2. 请求两个到多个bex

请求多个bex只需要配置多个service\_business即可，例子如下：

<service service\_code="addExcluJob" service\_name="添加互斥岗位" service\_type="1" service\_cate="1" arrange\_type="1" soap\_enable="1">

<businesses>

<service\_business business\_code="addExcluJob" business\_method="" serial\_no="1" business\_name="添加互斥岗位" business\_type="1" business\_cate="1"/>

<service\_business business\_code="addCascadeExcluJob" business\_method="" serial\_no="1" business\_name="添加互斥岗位" business\_type="1" business\_cate="1"/>

</businesses>

</service>

**下面针对以上例子对service进行说明。**

* <service>标签属性说明

|  |  |  |
| --- | --- | --- |
| **属性(是否必填)** | **属性说明** | **缺省值** |
| service (Y) | service接口定义标签 | 无 |
| service\_code (Y) | 对应前端请求的service | 无 |
| service\_name (Y) | service名称 | 无 |
| service\_type(Y) | 类型。1：默认；2其他 | 1 |
| service\_cate(Y) | 分类。1：默认；2：不做查询只取view；20：调用工作流 | 1 |
| arrange\_type(Y) | 编排类型。1：至上而下顺序执行；2是根据脚本执行（找谭长伟） | 1 |
| soap\_enable(Y) | Webservice是否能调用。1：可以；2：不可以调用 | 1 |

* <service\_business>标签属性说明

|  |  |  |
| --- | --- | --- |
| **属性(是否必填)** | **属性说明** | **缺省值** |
| service\_business(Y) | 该标签为SERVICE原子业务定义 | 无 |
| business\_code(Y) | 对应bex的bex\_code | 无 |
| business\_method(N) | 对应原子业务的方法名称 | 无 |
| serial\_no(Y) | 现已不用 | 无 |
| business\_name(Y) | bex名称 | 无 |
| business\_type(Y) | 类型。根据此类型不同的值，分别调用： 1BEX 2BO 3ATOM，前者业务代码即为BEX接口代码，业务方法为空。中间为业务代码为BO对象代码，业务方法为对象某个方法。后面为原子业务接口 | 1 |
| business\_cate(Y) | 分类。1：默认 2其他 | 无 |

### 服务请求包

Web服务器传输请求解密后使用JSON格式数据

* 输入数据格式：

{

"REQUESTS": [

{

"REQ\_MSG\_HDR": {

"OP\_CODE": "1",

"OP\_ROLE": "181",

"OP\_BRANCH": "01",

"OP\_SITE": "127.0.0.1",

"OP\_WAY": "1",

"OP\_LANGUAGE": "1",

"OP\_PROGRAM": "",

"SERVER\_ID": "P0001041",

"MSG\_ID": 3251184

},

"REQ\_COMM\_DATA": {

"service": "P0001041",

"POST\_STA": "1"

}

}

]

}

### 服务应答包

* 返回输出数据格式：

<?xml version=*"1.0"* encoding=*"UTF-8"*?><requests><![CDATA[

{

ans:[

{"service":{"flag":"1","serialNo":""},"message":[{"flag":"1","prompt":"成功","rows":"2","times":"100ms"},{"flag":"1","prompt":"成功","rows":"2","times":"100ms"}],"config":{},"data":[[{"khxm":"heyz","khbh":"123"},{"khxm":"heyz","khbh":"123"}],[{"khxm":"heyz","khbh":"123"},{"khxm":"heyz","khbh":"123"}]]},

{"service":{"flag":"1","serialNo":""},"message":[{"flag":"1","prompt":"成功","rows":"2","times":"100ms"},{"flag":"1","prompt":"成功","rows":"2","times":"100ms"}],"config":{},"data":[[{"khxm":"heyz","khbh":"123"},{"khxm":"heyz","khbh":"123"}],[{"khxm":"heyz","khbh":"123"},{"khxm":"heyz","khbh":"123"}]]}]

}

]]></requests>

## 业务引擎对象(BEX)

### Bex概述

BEX收到SERVICE层的请求后，根据请求的参数，找到与BEX\_CODE属性相对应的BEX服务工厂进行处理，BEX主要提供LBM、ESB、KCXP、WEBSERVICE、JDBC、POJO、IBATIS 等数据访问通道，将数据请求在数据库、中间件、其他数据源中进行处理。

### Bex配置说明

#### 文件路径

在WEB-INF\conf\metaData\bexConf路径下新建一个.xml文件。

命名均要以-bex结尾。例如，新建一个用户管理的bex文件，则命名为：uum-bex.xml。

#### bex语法说明及例子

Bex例子如下：

<?xml version="1.0" encoding="UTF-8"?>

<bexs>

<!--员工销户 start-->

<bex bex\_code="delUserInfoQuery" bex\_type="4" page\_mode="2" bex\_cate="1" attr1="delUserInfoQuery">

<param para\_code="USER\_CODE" para\_name="用户标识" data\_type="12" para\_type="0" def\_val="-1"/>

</bex>

<!--员工销户 start-->

</bexs>

**下面针对以上例子对bex进行说明。注意以上划线部分。**

* <bex>标签属性说明

|  |  |  |
| --- | --- | --- |
| **属性(是否必填)** | **属性说明** | **缺省值** |
| bex (Y) | 该标签为BE接口定义 | 无 |
| bex\_code (Y) | 对应service的business\_code | 无 |
| bex\_type(Y) | 类型 。  1-SP\_BEX 存储过程接口  2-LBM\_BEX KCBP LBM 接口  3-BEAN\_BEX JavaBean 接口  4-BATIS\_BEX ibatis/mybatis 接口  5-SQL\_BEX SQL 数据集接口  6-WEBSRV\_BEX Web Service 接口  7-KESB\_BEX KESB接口  11-META\_SP\_BEX 针对元数据处理的存储过程 | 无 |
| page\_mode(Y) | 分页方式。 0-不分页；1-后台分页；2-中间分页；3-前端分页 | 1 |
| bex\_cate(Y) | 分类。1：默认；2：不做查询只取view；20：调用工作流 | 1 |
| attr1(Y) | 对应mybatis的id | 1 |

* <param>标签属性说明

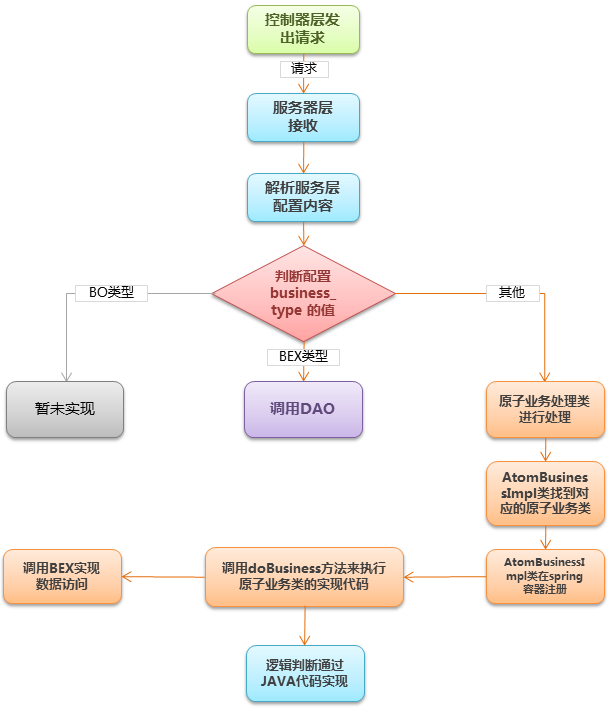
|  |  |  |
| --- | --- | --- |
| **属性(是否必填)** | **属性说明** | **缺省值** |
| param (Y) | 该标签为BE接口参数定义 | 无 |
| para\_code (Y) | 该参数对应数据库列字段名称 | 无 |
| para\_name (Y) | 该参数的名称 | 无 |
| para\_type (Y) | 参数类型 0-入参；1-出参；2-出入参；3-返回集 | 无 |
| data\_type (Y) | 数据类型。  100为组件，以java.sql.Types为参考  -7 BIT；-6 TINYINT；5 SMALLINT；4 INTEGER；-5 BIGINT；3 DECIMAL；2 NUMERIC；12 VARCHAR；91 DATE；2004 BLOB；2005 CLOB；1 CHAR；16 BOOLEAN；-10 CURSOR；100 COMPOSITE；0 UNDEFINED | 12 |
| def\_val (Y) | 该参数传到后台的缺省值 | 无 |

## 原子业务(ATOM)

### 原子业务概述

原子，一般指化学反应不可再分的基本微粒。在KJDP3.5平台中，指的是不可分割的业务功能。将业务功能细分为最小的模块，即原子业务。在业务开发过程中，通过拼装成整体业务，实现需要的业务功能。以此提高系统的高可用性和减少业务开发人员的工作量。

原子业务执行流程如下：



第一步、服务层GenericServiceImpl接收到控制器发出的请求，解析服务层中的Business配置内容。

第二步、判断配置business\_type的值

1：BEX类型，直接走DAO层调用完成处理，基础业务使用较多。

2：BO类型，暂时未实现。

3：即表示当前业务为原子业务，转发到原子业务处理类AtomBusinessImpl类的doBusiness方法进行处理。

第三步、AtomBusinessImpl类通过配置中的attr1找到对应的原子业务类，通过BeanRegister在spring容器中进行注册。

第四步、通过原子业务实例类的bean去调用原子业务基类AtomAbstract接口中doBusiness方法来执行原子业务类的实现代码。

第五步、原子业务类代码中可以通过DAO层接口去调用相应的BEX实现数据访问，逻辑判断通过JAVA代码来实现。

注：1）控制器接受前端请求调用服务层入口及过程与原子业务无关，不详细说明。

2）数据访问层处理过程与原子业务无关，不详细说明。

### 原子业务开发说明

以下标题顺序是建议开发步骤，仅供参考。

#### 前端请求

前端发ajax请求调用 service，对应service配置里的service\_code

例如：

ajaxRequest({

req:[{

**service:' uas\_custIdentify ',**

CUST\_ID:CUST\_ID

}],

error:function(){

alert('失败！');

},

func:function(data){

if(data[0]["service"]["flag"]=='0'){

alert('成功！');

}

}

});

#### 解析服务层配置

service判断business\_type的值，如果business\_type="3"，则是调用原子业务。

service配置里根据属性查找里面配置的类与方法。

配置说明：

* service中：

service\_code对应前端请求的service

* service\_business中：

business\_code对应bex的bex\_code；

business\_method对应原子业务的方法名称；

business\_name的名称必须唯一，对应该原子业务的类名

business\_type=3表示调用原子业务

例子如下：

<?xml version="1.0" encoding="UTF-8"?>

<services>

<!--客户信息识别 start-->

<service service\_code="uas\_custIdentify" service\_name="客户身份识别" attr1="">

<businesses>

<service\_business business\_code="custIdentifyBiz" business\_type="3"

business\_method="custIdentify" serial\_no="1" business\_name="CustInfoBiz" />

</businesses>

</service>

<!--客户信息识别 end-->

</services>

#### BEX层配置

BEX收到SERVICE层的请求后，根据请求的参数，找到与BEX\_CODE属性相对应的BEX服务工厂进行处理，将数据请求在mybatis进行数据处理，在java层进行业务逻辑处理。

配置说明：

* bex中：

bex\_code对应服务层请求的service

attr1对应mybatis的id；

attr2对应ds.properties文件中的名称，如下粗体：

**ecif**.jdbcUrl=jdbc:oracle:thin:@192.168.50.224:1521:cif

例子如下：

<?xml version="1.0" encoding="UTF-8"?>

<bexs>

<!--客户信息识别 start-->

<bex bex\_code="CustIdentifyBex" bex\_type="4" page\_mode="2" bex\_cate="1"

attr1="custIdentifyQuery" attr2="ecif">

</bex>

<!--客户信息识别 end-->

</bexs>

#### 原子业务代码

原子业务代码主要处理业务逻辑，写在java代码中。

配置说明：

原子类写在 com.szkingdom.business.atom 里面

代码中的setBexCode("xxx")会根据"xxx"去调用对用的bex\_code

代码例子如下：

package com.szkingdom.business.atom;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.szkingdom.frame.business.atom.AtomResult;

import com.szkingdom.frame.business.atom.DataExchangeAssembly;

import com.szkingdom.frame.service.model.GenericRequest;

import com.szkingdom.frame.service.model.GenericResult;

/\*\*

\* @author tanchangwei

\* @since 2014/02/26

\* 客户概要资料查询

\*/

public class CustInfoBiz extends AtomAbstract{

/\*\*

\* @客户身份识别

\* 表: CUSTID\_INFO

\* 入参: ID\_CODE,ID\_TYPE

\* 出参: CIF\_CODE,USER\_FNAME

\* 规则: id\_type,id\_code模糊匹配/cif\_code精确查询

\* 模块: custIdentify

\*/

public GenericResult **custIdentify**(Map params,

DataExchangeAssembly dataExchange) {

GenericRequest genericRequest = new GenericRequest();

genericRequest.setRequestParams(params);

**setBexCode("CustIdentifyBex");**

GenericResult result = business.doBusiness(genericRequest);

return new AtomResult(result);

}

}

#### Mybatis访问数据层

Mybatis配置主要用于对数据库的数据操作，常用的有insert、select、delete、update。

要注意：每新增一个mybatis配置文件，要在：

WEB-INF\conf\mybatis\mybaits-config-all.xml中配置新增文件的路径，例如：

<mapper resource="../conf/mybatis/mapper-frame.xml" />

mybatis例子如下：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.szkingdom.frame.dao.ibatis.mapper.ISystemMapper">

<!--客户身份识别 标识符查询 ID\_CODE与CIF\_CODE不能同时为空,客户识别.. -->

<select id="custIdentifyQuery" resultType="java.util.HashMap" parameterType="java.util.HashMap">

SELECT USER\_NAME,CUST\_TYPE,CIF\_CODE

FROM CUSTID\_INFO

<where>

<choose>

<when test="CIF\_CODE != null and CIF\_CODE != ''">

CIF\_CODE = ${CIF\_CODE}

</when>

<otherwise>

<if test="ID\_TYPE != null and ID\_TYPE != '' ">

AND ID\_TYPE LIKE '%${ID\_TYPE}%'

</if>

AND ID\_CODE LIKE '%${ID\_CODE}%'

</otherwise>

</choose>

</where>

</select>

</mapper>

# 前台页面开发

## 前台开发概述

前台开发主要使用业务视图VIEW配置、HTML、JQuery、CSS、KUI组件进行构建，通过AJAX方式提交到控制层。控制层需要配置几个文件，包括service、bex。控制层再提供给数据访问层，访问层包括mybatis文件。

## 页面开发

### View配置

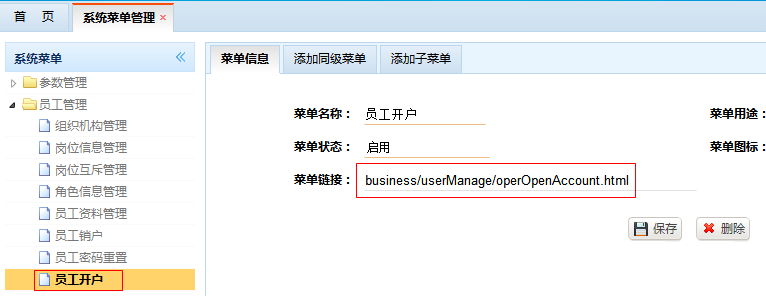
开发方法见3.2业务视图章节

### Html开发

1. 配置文件路径：

在系统菜单管理中，填写菜单在webapp下的路径，路径为business/userManage/operOpenAccount.html。

设置路径如下：



1. 页面开发注意事项：

* 每个页面都需要引进这3个文件：icon.css、jquery-1.8.3.min.js、kui.loader.js

路径如下：

<link rel="stylesheet" type="text/css" href="../../kui-base/themes/icon.css">

<script type="text/javascript" src="../../kui-base/js/core/jquery-1.8.3.min.js"></script>

<script type="text/javascript" src="../../kui-base/js/core/kui.loader.js"></script>

* 页面使用KUI组件时，需要动态加载组件。动态加载组件部分参见4.5.1.2Html

## Javascript开发

**语法**：由于客户端框架引入了jQuery，所以在编写JS代码时，要求优先使用jQuery的写法而少用甚至避免使用原生的JavaScript写法，特别是在访问和控制DOM对象、事件绑定、遍历元素等方面。但在封装一个独立的插件时除外（减少依赖）。示例如下：

var userName = document.getElementById("USER\_NAME").value; //避免

document.getElementById("DIV\_TIP").style.display = "block"; //避免

document.getElementById("DIV\_TIP").innerHTML = "test"; //避免

var userName = $("#USER\_NAME").val(); //正确

$("#DIV\_TIP").css("display", "block").html("test"); //正确

**JS程序入口：**在html文件中编写JS程序时，统一将执行的入口程序放在JS代码段落的末尾，做法类似main()函数。

<script type="text/javascript">

var ajax = new AjaxEngine();

function renderTree() {

//TODO

}

function loadChkObjectDDL() {

//TODO

}

/\*\*

\* 程序入口，初始化页面

\*/

window.onload = function() {

renderTree();

loadChkObjectDDL();

//Others

}

</script>

**函数还是对象**：在实现一个模块的过程中，通常都需要写一些JS函数用来处理不同的任务，那么到底是通篇都写成一个一个的函数还是按模块封装成对象，这需要权衡模块的复杂度和功能多少。推荐的做法是在开始编码前先弄清楚有哪些函数，当函数较多让你需要提取成JS文件时，那么你需要按功能进行划分，并封装成若干个对象。此外，当你写的JS函数有可能被其他模块重用时，你需要按类型进行划分并封装成对象以便复用。

**对象的写法：**JavaScript定义对象典型的有构造函数和原型等方式，在本框架下，基本都采用构造函数方式，示例如下：

function GridService(gridParam) {

this.gridService = null;

this.gridParam = gridParam || null;

this.setGridParam = function(gridParam) {

this.gridParam = gridParam;

};

this.loadProcess = function() {

};

this.saveProcess = function() {

var index = 0;

};

}

**代码保存位置规范**：通常为了简单方便和提高开发效率及运行效率，JS代码可以直接写在网页的<script/>标记里，但是为了避免在一个网页内写入过多的内容而降低可读性，所以当JS代码达到5屏（大约150行）时建议提取成一个JS文件，再通过链接的方式放在网页的<head/>内。

**优化选择器的执行速度：**在使用代码访问元素时，如果有id号，建议使用id号直接访问元素，这样速度是最快的；如果没有id号，可使用元素标记访问，其次就是使用类别（class）进行访问。

**减少对DOM元素的直接操作：**在网页中反复对DOM进行操作，尤其是循环或递归操作，其性能是很低的，会造成在网页上显示结果需要等待数秒甚至更长时间。因此为了减少对DOM元素直接操作的次数，有必要在操作前完善大部分的DOM操作，最后通过一次直接操作，更新其DOM结构。常用的做法是先在循环或递归中拼字符串，最后一次性完成DOM元素的增加。

## CSS开发

在开发过程中，统一样式由统一的CSS文件控制，每个页面都需引入以下文件：

kui-base/themes/icon.css。如特殊需要，在页面<head>内写CSS，如下所示：

<style type="text/css">

.tips{

color:red;

}

</style>

## KUI组件使用

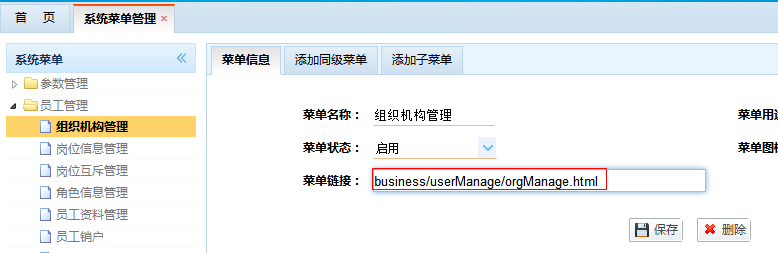
### 使用方法

#### View

开发方法建3.2.2.2VIEW结合KUI使用说明

#### Html

1.在系统菜单配置好菜单链接，如下所示：



2.动态加载组件：

* 每个页面要加载KUI，必须引入jquery-1.8.3.min.js 、kui.loader.js的文件，此文件会解析页面引入的组件进行渲染。

路径如下（以下3个文件是每个页面必须的）：

<link rel="stylesheet" type="text/css" href="../../kui-base/themes/icon.css">

<script type="text/javascript" src="../../kui-base/js/core/jquery-1.8.3.min.js"></script>

<script type="text/javascript" src="../../kui-base/js/core/kui.loader.js"></script>

* 当在触发事件的时候，用JavaScript调用的组件，要用using属性再进行动态加载。例子如下：

<script type="text/javascript">

function initTable(){

using('datagrid',function(){

$("#ageTraBusInfo").datagrid({

height:287,

fitColumns:true,

singleSelect: true,

showRowDetail:false,

pageSize:2,

pageList:[2,10,20,30,40,50],

required:true,

//req:[{bex\_codes:'demo\_dlkhywsz\_bex'}],

columns:[[

{title:'代理人代码',field:''},

{title:'客户代码',field:''},

{title:'用户角色',field:''},

{title:'交易板块',field:''},

{title:'交易账户',field:''},

{title:'证券类别',field:''},

{title:'证券业务',field:''},

{title:'开始日期',field:''},

{title:'结束日期',field:''}

]]

})

})

}

</script>

</head>

<body>

<div class="kui-panel" title="客户交易信息">

<table id='ageTraBusInfo' class="kui-datagrid"></table>

</div>

1. 在html中使用组件

* 使用组件：

在html中使用组件时，首先要加一个以“kui-类名”的类，例如使用datagrid，即“class="kui-datagrid"”。

例子如下：

<table id='ageTraBusInfo' class="kui-datagrid"></table>

* 设置组件属性：

给组件设置属性时，要把组件写在kui-optioins=””中，例子如下：

<table id="t1" class="kui-datagrid" title="临柜业务信息" style="width:400px;height:250px"

kui-options="border:true,singleSelect:true,showRowDetail:false,

req:[],fit:true">

<thead>

<tr>

<th kui-options="field:'',width:150,sortable:true,checkbox:false,resizable:true,title:'临柜业务'"></th>

<th kui-options="field:'',width:120,sortable:true,title:'业务数据'"></th>

<th kui-options="field:'',width:100,sortable:true,title:'客户名称'"></th>

<th kui-options="field:'',width:100,sortable:true,title:'客户代码'"></th>

<th kui-options="field:'',width:100,sortable:true,title:'任务序号'"></th>

<th kui-options="field:'',width:150,sortable:true,title:'当前状态'"></th>

<th kui-options="field:'',width:150,sortable:true,title:'发生时间'"></th>

<th kui-options="field:'',width:150,sortable:true,title:'备注'"></th>

</tr>

</thead>

</table>

### 基本组件

#### 一般拖动组件(kui- draggable)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| proxy（替代） | string,function | 拖动时的替代元素，当被设置为'clone'时，将使用该元素的一个复制元素来作为替代元素。如果指定了一个函数, 它将返回一个jquery对象。  $('.dragitem').draggable({  proxy: function(source){  var p = $('<div style="border:1px solid #ccc;width:80px"></div>');  p.html($(source).html()).appendTo('body');  return p;  }  }); | null |
| revert（复原） | boolean（布尔型） | 如果设置为true, 当拖动停止时元素将返回它的初始位置。 | false |
| cursor（指针） | string（字符串） | 拖动时的CSS指针。 | move |
| deltaX（水平增量，X轴） | number（数字） | 被拖动元素对应于当前指针的水平位置。 | null |
| deltaY（垂直增量，y轴） | number（数字） | 被拖动元素对应于当前指针的垂直位置。 | null |
| handle（句柄） | selector（选择器） | 开始拖动的句柄。 | null |
| disabled  （停止拖动） | boolean（布尔型） | 当设置为true时停止拖动。 | false |
| edge（边缘） | number（数字） | 可以在其中拖动的容器的宽度 | 0 |
| axis（拖动轴） | string（字符串） | 可用值为'v'或者'h',当设置为空时，元素可以同时在水平和垂直方法拖动。 | null |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onBeforeDrag | e | 在拖动之前触发，返回false将取消拖动。 |
| onStartDrag | e | 当目标对象开始被拖动时触发。 |
| onDrag | e | 在拖动过程中触发，当不能再拖动时返回false。 |
| onStopDrag | e | 当拖动停止时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| proxy | none | 如果替代元素被设置，返回将被用于拖动的替代元素。 |
| enable | none | 允许拖动。 |
| disable | none | 禁止拖动。 |

#### 拖动至容器组件(kui- droppable)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| accept | selector | 决定哪个可拖动元素将会被接收。 | null |
| disabled | boolean | 设置true将无法拖动元素 | false |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onDragEnter | e,source | 当可拖动元素被拖入目标容器的时候触发，参数source是被拖动的DOM元素。 |
| onDragOver | e,source | 当可拖动元素被拖至某个元素之上的时候触发，参数source是被拖动的DOM元素。 |
| onDragLeave | e,source | 当可拖动元素被拖离目标容器的时候触发，参数source是被拖动的DOM元素。 |
| onDrop | e,source | 当可拖动元素被拖入目标容器并放置的时候触发，参数source是被拖动的DOM元素。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回拖动至容器的属性 |
| enable | none | 设置拖动至容器可用 |
| disable | none | 设置拖动至容器不可用 |

#### 缩放组件(kui- resizable)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| disabled（禁止缩放） | boolean（布尔型） | 如果设置为true将禁止缩放。 | false |
| handles（句柄） | string（字符串） | 说明缩放的方向，'n'表示向上（北）,'e'表示向右（东）等。 | n, e, s, w, ne, se, sw, nw, all |
| minWidth（最小宽度） | number（数字） | 缩放时所允许的最小宽度。 | 10 |
| minHeight（最小高度） | number（数字） | 缩放时所允许的最小高度。 | 10 |
| maxWidth（最大宽度） | number（数字） | 缩放时所允许的最大宽度。 | 10000 |
| maxHeight | number（数字） | 缩放时所允许的最大高度。 | 10000 |
| edge（边界） | number（数字） | 将被缩放的边框的边界。 | 5 |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onStartResize | e | 当开始缩放时触发。 |
| onResize | e | 在缩放过程中触发，当返回false时, DOM元素将不再缩放。 |
| onStopResize | e | 当缩放停止是触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回缩放属性对象。 |
| enable | none | 设置缩放可用 |
| disable | none | 设置缩放不可用。 |

#### 分页组件(kui- pagination)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| total（总数） | number  （数字） | 总记录数，当创建分页时必须设置。 | 1 |
| pageSize（每页记录数） | number  （数字） | 每页显示的记录数。 | 10 |
| pageNumber  （当前页码） | number  （数字） | 当分页创建完毕时显示当前页码。 | 1 |
| pageList  （可选择的每页记录数） | array  （数组） | 用户能够改变每页显示的记录数。  $('#pp').pagination({  pageList: [10,20,50,100]  }); | [10,20,30,50] |
| loading  （是否正在载入） | boolean  （布尔型） | 定义数据是否正在载入。 | false |
| buttons（按钮） | array  （数组） | 自定义按钮，每个按钮包含2个属性: iconCls： 显示背景图片的CSS类 handler： 当按钮被点击时调用的一个句柄函数 定义在属性中。  <div class="kui-pagination" style="border:1px solid #ccc" kui-options="  total: 114,  buttons: [{  iconCls:'icon-add',  handler:function(){alert('add')}  },'-',{  iconCls:'icon-save',  handler:function(){alert('save')}  }]">  </div>  用javascript创建。  $('#pp').pagination({  total: 114,  buttons: [{  iconCls:'icon-add',  handler:function(){alert('add')}  },'-',{  iconCls:'icon-save',  handler:function(){alert('save')}  }]  }); | null |
| showPageList（显示可选择的每页记录数） | boolean  （布尔型） | 定义是否显示可选择的每页记录数。 | true |
| showRefresh  （显示刷新） | boolean  （布尔型） | 定义是否显示刷新按钮。 | true |
| beforePageText（输入框之前的文本） | string  （字符串） | 在输入框之前显示的符号。 | Page |
| afterPageText（输入框之后的文本） | string  （字符串） | 在输入框之后显示的符号。 | of {pages} |
| displayMsg（显示信息） | string  （字符串） | 在插件右上方显示分页信息。 | Displaying {from} to {to} of {total} items |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onSelectPage | pageNumber, pageSize | 当用户进行翻页时触发，回调函数包含2个参数: pageNumber： 下一页的页码 pageSize： 下一页的显示记录数  $('#pp').pagination({  onSelectPage:function(pageNumber, pageSize){  $(this).pagination('loading');  alert('pageNumber:'+pageNumber+',pageSize:'+pageSize);  $(this).pagination('loaded');  }  }); |
| onBeforeRefresh | pageNumber, pageSize | 刷新之前触发, 返回false将取消刷新。 |
| onRefresh | pageNumber, pageSize | 刷新之后触发。 |
| onChangePageSize | pageSize | 当用户修改每页显示记录数时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回分页属性对象。 |
| loading | none | 提醒分页插件正在载入。 |
| loaded | none | 提醒分页插件已经载入。 |
| refresh | options | 刷新和显示页面信息。  $('#pp').pagination('refresh'); // 刷新id为pp的分页信息  $('#pp').pagination('refresh',{ // 改变属性刷新id为pp的分页信息  total: 114,  pageNumber: 6  }); |
| select | page | 选中一个新的页面，页面索引从1开始。  $('#pp').pagination('select'); // 选中当前分页  $('#pp').pagination('select', 2); // 选中第二个分页页面 |

#### 搜索框组件(kui- searchbox)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | **类型** | **描述** | **默认值** |
| width（宽度） | number（数字） | 设置组建的宽度。 | auto |
| prompt  （提醒） | string（字符串） | 显示在搜索框的提醒信息。 | '' |
| value（值） | string（字符串） | 搜索框的默认值。 | '' |
| menu（菜单） | selector（选择器） | 搜索类型下拉菜单。  <input class="kui-searchbox" style="width:300px" kui-options="menu:'#mm'" />  <div id="mm" style="width:150px">  <div kui-options="name:'item1'">Search Item1</div>  <div kui-options="name:'item2',selected:true">Search Item2</div>  <div kui-options="name:'item3'">Search Item3</div>  </div> | null |
| searcher  （搜索器） | function(value,name) | 当用户点击搜索按钮或者按下ENTER键的时候搜索函数将被调用。 | null |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回搜索框属性对象。 |
| menu | none | 返回搜索类型菜单对象。  var m = $('#ss').searchbox('menu'); // get the menu object  var item = m.menu('findItem', 'Sports News'); // find the menu item  // change the menu item icon  m.menu('setIcon', {  target: item.target,  iconCls: 'icon-save'  });  // select the searching type name  $('#ss').searchbox('selectName', 'sports'); |
| textbox | none | 返回文本框对象。 |
| getValue | none | 返回当前搜索关键字。 |
| setValue | value | 设置新的搜索关键字。 |
| getName | none | 返回当前搜索类型。 |
| selectName | name | 选中当前搜索类型。  $('#ss').searchbox('selectName', 'sports'); |
| destroy | none | 清除搜索框组件 |
| resize | width | 重置搜索框组建的宽度。 |

#### 进度条组件(kui- progressbar)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| width（宽度） | string（字符串） | 设置进度条的宽度。 | auto |
| value（值） | number（数字） | 当前进度，百分比数值。 | 0 |
| text（文本） | string（字符串） | 显示在组件中的文本模板。 | {value}% |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onChange | newValue,oldValue | 当进度发生改变时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回进度条属性对象。 |
| resize | width | 重置进度条。 |
| getValue | none | 返回当前进度值。 |
| setValue | value | 设置一个新的进度值。 |

### 布局管理器组件

#### 控制面板组件(kui-panel)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| id | string（字符串） | 控制面板的id属性。 | null |
| title（标题） | string（字符串） | 显示在控制面板顶部的标题文本。 | null |
| iconCls  （图标CSS类） | string（字符串） | 在控制面板标题前显示一个16x16大小图标的CSS类。 | null |
| width（宽度） | number（数字） | 设置控制面板的宽度。 | auto |
| height（高度） | number（数字） | 设置控制面板的高度。 | auto |
| left（左边距） | number（数字） | 设置控制面板的左边距。 | null |
| top（顶边距） | number（数字） | 设置控制面板的顶边距。 | null |
| cls（类） | string（字符串） | 对控制面板引用一个CSS类。 | null |
| headerCls  （头部css类） | string（字符串） | 对控制面板头部引用一个CSS类。 | null |
| bodyCls（主体类） | string（字符串） | 对控制面板主体引用一个CSS类。 | null |
| style（样式） | object（对象） | 给控制面板添加一个自定义的样式。  <div class="kui-panel" style="width:200px;height:100px" kui-options="style:{borderWidth:2}"> </div> | {} |
| fit(铺满浏览器) | boolean（布尔型） | 设置为true时，控制面板的大小将铺满它所在的容器（浏览器）。  <div style="width:200px;height:100px;padding:5px"> <div class="kui-panel" style="width:200px;height:100px" kui-options="fit:true,border:false"> Embedded Panel </div> </div> | false |
| border（边框） | boolean（布尔型） | 定义是否显示控制面板边框。 | true |
| doSize  （调整大小） | boolean（布尔型） | 如果设置为true，在控制面板被创建时将被重置大小并且自动布局。 | true |
| noheader  （无头部） | boolean（布尔型） | 如果设置为true，控制面板头部将不被创建。 | false |
| content（内容） | string（字符串） | 控制面板主体的内容。 | null |
| fieldset (类似于fieldset标签) | boolean (布尔型) | 配置fieldset，noheader属性为true,控制面板显示样式类似于fieldset标签  <div class="kui-panel" title="合同情况" kui-options="fieldset:true,noheader:true">  </div> | null |
| collapsible  （可折叠） | boolean（布尔型） | 定义是否显示可折叠按钮。 | false |
| minimizable  （最小化） | boolean（布尔型） | 定义是否显示最小化按钮。 | false |
| maximizable  （最大化） | boolean（布尔型） | 定义是否显示最大化按钮。 | false |
| closable（关闭） | boolean（布尔型） | 定义是否显示关闭按钮。 | false |
| tools（工具栏） | array（数组） | 自定义工具栏，每一个工具都可以包含2个属性：图标类 和句柄。  工具栏定义在已经存在的<div>标签上    <div class="kui-panel" style="width:300px;height:200px"  title="My Panel" kui-options="iconCls:'icon-ok',tools:'#tt'">  </div>  <div id="tt">  <a href="#" class="icon-add" onclick="javascript:alert('add')"></a>  <a href="#" class="icon-edit" onclick="javascript:alert('edit')"></a>  </div>      工具栏也可以定义在数组中    <div class="kui-panel" style="width:300px;height:200px"  title="My Panel" kui-options="iconCls:'icon-ok',tools:[  {  iconCls:'icon-add',  handler:function(){alert('add')}  },{  iconCls:'icon-edit',  handler:function(){alert('edit')}  }]">  </div> | [] |
| collapsed  （已折叠） | boolean（布尔型） | 定义控制面板初始化时是否折叠。 | false |
| minimized  （已最小化） | boolean（布尔型） | 定义控制面板初始化时是否最小化。 | false |
| maximized  （已最大化） | boolean（布尔型） | 定义控制面板初始化时是否最大化。 | false |
| closed（已关闭） | boolean（布尔型） | 定义控制面板初始化时是否关闭。 | false |
| href（超链接） | string（字符串） | 载入远程数据的超链接，载入的数据将显示在控制面板中。  <div id="pp" class="kui-panel" style="width:300px;height:200px"  kui-options="href='get\_content.php',closed:true">  </div>  <a href="#" onclick="javascript:$('#pp').panel('open')">Open</a> | null |
| cache（缓存） | boolean（布尔型） | 如果设置为true，从超链接载入的数据将被缓存。 | true |
| loadingMessage（载入时信息） | string（字符串） | 在正在载入远程数据时显示在控制面板中的信息。 | Loading… |
| extractor  （提取器） | function（函数） | 定义如何从ajax返回值中提取内容,返回被提取的数据。  extractor: function(data){  var pattern = /<body[^>]\*>((.|[\n\r])\*)<\/body>/im;  var matches = pattern.exec(data);  if (matches){  return matches[1]; // only extract body content  } else {  return data;  }  } |  |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onLoad | none | 在远程数据被载入时触发。 |
| onBeforeOpen | none | 在控制面板被打开之前触发，返回false将停止打开。 |
| onOpen | none | 在控制面板被打开之后触发。 |
| onBeforeClose | none | 在控制面板被关闭之前触发，返回false将取消关闭。 |
| onClose | none | 在控制面板被关闭后触发。 |
| onBeforeDestroy | none | 在控制面板被注销前触发，返回false将取消注销。 |
| onDestroy | none | 在控制面板被注销后触发。 |
| onBeforeCollapse | none | 在控制面板被折叠之前触发，返回false将停止折叠。 |
| onCollapse | none | 在控制面板被折叠之后触发。 |
| onBeforeExpand | none | 在控制面板被扩展之前触发，返回false将停止扩展（这里应该是指扩展区域，宽、高等）。 |
| onExpand | none | 在控制面板被扩展之后触发。 |
| onResize | width, height | 在控制面板被缩放后触发。 width： 新的控制面板宽度 height：新的控制面板高度 |
| onMove | left,top | 在控制面板被移动后触发。 left：新的控制面板左边距 top：新的控制面板顶边距 |
| onMaximize | none | 在控制面板被最大化后触发 |
| onRestore | none | 在控制面板被重置为初始大小后触发。 |
| onMinimize | none | 在控制面板被最小化后触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回选项属性。 |
| panel | none | 返回控制面板对象。 |
| header | none | 返回控制面板头对象。 |
| body | none | 返回控制面板主体对象。 |
| setTitle | title | 设置控制面板头部的标题文本。 |
| open | forceOpen | 当参数forceOpen设置为true时，控制面板将被打开，不受onBeforeOpen回调函数的约束。 |
| close | forceClose | 当参数forceClose设置为true时, 控制面板将被打开，不受onBeforeClose回调函数的约束。 |
| destroy | forceDestroy | 当参数forceDestroy设置为true时,控制面板将被打开，不受onBeforeDestroy回调函数的约束。 |
| refresh | href | 当href属性被设置时，刷新控制面板以载入远程数据。 |
| resize | options | 这是控制面板的尺寸并且进行布局。options对象包含以下2个属性: width： 新的控制面板宽度 height： 新的控制面板高度 left： 新的控制面板左边距 top： 新的控制面板顶边距 |
| move | options | 移动控制面板到一个新的位置。options对象包含以下2个属性: left： 新的控制面板左边距 top： 新的控制面板顶边距 |
| maximize | none | 使控制面板铺满整个容器。 |
| minimize | none | 最小化控制面板。 |
| restore | none | 使最大化的控制面板重置为其初始化时的大小和位置。 |
| collapse | animate | 折叠控制面板主体。 |
| expand | animate | 扩展控制面板主体。 |

#### 选项卡组件(kui-tabs)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| width（宽度） | number（数字） | 选项卡所在容器（控制面板）的宽度。 | auto |
| height（高度） | number（数字） | 选项卡所在容器（控制面板）的高度。 | auto |
| plain（简洁模式） | boolean（布尔型） | 如果设置为true，将不显示控制面板背景。 | false |
| fit（铺满浏览器） | boolean（布尔型） | 设置为true时，选项卡的大小将铺满它所在的容器（浏览器）。 | false |
| border（边框） | boolean（布尔型） | 如果设置true，将显示选项卡所在容器（控制面板）的边框。 | true |
| scrollIncrement（滚动增量） | number（数字） | 选项卡滚动条每次滚动的像素值。 | 100 |
| scrollDuration（滚动时间） | number（数字） | 每次滚动持续的时间，单位为毫秒。 | 400 |
| tools（工具栏） | array（数组）,selector | 控制面板右侧的工具栏，每个工具选项都跟链接按钮一样。 工具栏放在array（数组）中。  $('#tt').tabs({  tools:[{  iconCls:'icon-add',  handler:function(){  alert('add')  }  },{  iconCls:'icon-save',  handler:function(){  alert('save')  }  }]  });  定义工具栏在已经存在的<div> 标签上。  $('#tt').tabs({  tools:'#tab-tools'  });  <div id="tab-tools">  <a href="#" class="kui-linkbutton" plain="true" iconCls="icon-add"></a>  <a href="#" class="kui-linkbutton" plain="true" iconCls="icon-save"></a>  </div> | null |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onLoad | panel | 当一个选项卡完成ajax远程载入对象时触发。 |
| onSelect | title,index | 当用户选择一个选项卡时触发。 |
| onBeforeClose | title,index | 在一个选项卡被关闭之前触发，返回false将取消关闭。 |
| onClose | title,index | 在用户关闭一个选项卡面板后触发。 |
| onAdd | title,index | 在一个选项卡面板被添加后触发。 |
| onUpdate | title,index | 在一个选项卡面板被更新后触发。 |
| onContextMenu | e,title,index | 在一个选项卡面板被鼠标右键单击后触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回选项卡属性对象。 |
| tabs | none | 返回所有的选项卡面板。 |
| resize | none | 重置选项卡容器（控制面板）大小并且自动布局。 |
| add | options | 添加一个新的选项卡面板，options参数是一个可以配置的对象，查看选项卡控制面板属性获取更多信息。 $('#tt').tabs('add',{ title: 'new tab', selected: false //... }); |
| close | which | 关闭一个选项卡面板，which包括title或index参数，表示哪个选项卡将被关闭。 |
| getTab | which | 获取特定的选项卡面板名称。which包括title或index。 |
| getTabIndex | which | 获取特定的选项卡面板索引。which包括title或index。 |
| getSelected | none | 获取被选择的选项卡面板名称。 var tab = $('#tt').tabs('getSelected'); var index = $('#tt').tabs('getTabIndex',tab); alert(index); |
| select | which | 选择一个选项卡面板。which包括title或index。 |
| exists | which | 验证一个特定的选项卡面板是否存在。which包括title或index。 |
| update | param | 更新特定的选项卡面板，param参数包含2个属性: tab: 将被更新的选项卡。  options: 选项卡相关配置项。 var tab = $('#tt').tabs('getSelected'); // 获取选中的tab $('#tt').tabs('update', { tab: tab, options: { title: 'New Title', href: 'get\_content.php' } });  var tab = $('#tt').tabs('getSelected');  tab.panel('refresh', 'get\_content.php'); |
| enableTab | which | 启用选项卡，which 指选项卡的index或title  $('#tt').tabs('enableTab', 1); // 禁用第二个选项卡 $('#tt').tabs('enableTab', 'Tab2'); //禁用标题Tab2的选项 |
| disableTab | which | 禁用选项卡，which 指选项卡的index或title  $('#tt').tabs('enableTab', 1); // 禁用第二个选项卡 $('#tt').tabs('enableTab', 'Tab2'); //禁用标题Tab2的选项 |

#### 可伸缩面板组件(kui-accordion)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| width（宽度） | number（数字） | 可伸缩面板所在容器的宽度。 | auto |
| height（高度） | number（数字） | 可伸缩面板所在容器的高度。 | auto |
| fit（铺满浏览器） | boolean（布尔型） | 如果设置为true，可伸缩面板所在容器将铺满它所在的父容器（浏览器） | false |
| border（边框） | boolean（布尔型） | 定义是否显示边框。 | true |
| animate（动画效果） | boolean（布尔型） | 定义当延伸或者折叠面板时是否显示动画效果。 | true |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onSelect | title,index | 当一个可伸缩面板被选择时触发。 |
| onAdd | title,index | 在一个新面板被添加时触发。 |
| onBeforeRemove | title,index | 在可伸缩面板被移除之前触发，返回false将取消移除。 |
| onRemove | title,index | 在一个可伸缩面板被移除时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回容器属性对象。 |
| panels | none | 获取所有的面板。 |
| resize | none | 重置可伸缩面板。 |
| getSelected | none | 获取被选择的面板。 |
| getPanel | which | 获取特定的可伸缩面板。which包括title或index |
| getPanelIndex | panel | 获取特定的可伸缩面板索引。  var p = $('#aa').accordion('getSelected');  if (p){  var index = $('#aa').accordion('getPanelIndex', p);  alert(index);  } |
| select | which | 选择特定的面板。which包括title或index |
| add | options | 添加一个先的可伸缩面板。  $('#aa').accordion('add', {  title: 'New Title',  content: 'New Content',  selected: false  }); |
| remove | which | 移除特定的面板。which包括title或index |

#### 布局面板组件(kui-layout)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| title（标题） | string（字符串） | 布局面板的标题。 | null |
| region（区域） | string（字符串） | 定义布局面板的位置（方向），可以取下列值的其中之一：north, south, east, west, center。 |  |
| border（边框） | boolean（布尔型） | 设置为true将显示布局面板的边框。 | true |
| split（分隔条） | boolean（布尔型） | 如果设置为true将显示一个分隔条，用户可以拖动分隔条来改变布局面板的尺寸。 | false |
| iconCls（图标CSS类） | string（字符串） | 一个用来显示布局面板头部图标的css类。 | null |
| href（超链接） | string（字符串） | 一个用来从远程站点载入数据的超链接。 | null |
| fit | boolean(布尔型) | 设置为true时，布局面板的大小将铺满所在的容器，创建布局面板是在body标签上时，布局面板将在整个页面上最大化显示。 | null |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| resize | none | 设置布局面板的尺寸大小。 |
| panel | region | 返回特定的布局面板，'region'参数的可取值为：'north','south','east','west','center'。 |
| collapse | region | 折叠特定的布局面板，'region'参数的可取值为：'north','south','east','west'。 |
| expand | region | 延伸特定的布局面板，'region'参数的可取值为：'north','south','east','west'。 |

### 菜单与按钮组件

#### 菜单组件(kui-menu)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| zIndex（堆叠顺序） | number（数字） | 菜单的z-index样式,从第一个菜单开始递增。 | 110000 |
| left（左边距） | number（数字） | 菜单的左边距。 | 0 |
| top（顶边距） | number | 菜单的顶边距。 | 0 |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onShow | none | 当一个菜单被显示后触发。 |
| onHide | none | 在一个菜单被隐藏后触发。 |
| onClick | item | 当一个菜单被点击时触发。  <div class="kui-menu" kui-options="onClick:menuHandler" style="width:120px;">  <div kui-options="name:'new'">New</div>  <div kui-options="name:'save',iconCls:'icon-save'">Save</div>  <div kui-options="name:'print',iconCls:'icon-print'">Print</div>  <div class="menu-sep"></div>  <div kui-options="name:'exit'">Exit</div>  </div>  <script type="text/javascript">  function menuHandler(item){  alert(item.name)  }  </script> |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| show | pos | 在一个特定的位置显示菜单。 pos参数有2个属性： left：新的左边距。 top：新的顶边距。 |
| hide | none | 隐藏一个菜单。 |
| destroy | none | 清除一个菜单。 |
| getItem | itemEl | 获取菜单项数据并将其返回, 数据包含以下属性： target：DOM对象，菜单项目。 id：字符串，分配给元素的ID。 text：字符串，菜单项的文本。 href：字符串，超链接地址。 disabled：布尔型，菜单项是启用还是禁用。 onclick：函数, 当用户点击菜单时将要执行的函数。 iconCls：字符串，图标css类。  <div id="mm" class="kui-menu" style="width:120px">  <div>New</div>  <div id="m-open">Open</div>  <div>Save</div>  </div>  var itemEl = $('#m-open')[0]; // the menu item element  var item = $('#mm').menu('getItem', itemEl);  console.log(item); |
| setText | param | 设置特定的菜单文本，'param'参数包含2个属性： target：DOM对象，将要被设置的菜单项。 text：字符串，新的文本值。  var item = $('#mm').menu('findItem', 'Save');  $('#mm').menu('setText', {  target: item.target,  text: 'Saving'  }); |
| setIcon | param | 设置特定的菜单项图标，'param'参数包含2个属性： target：DOM对象，菜单项目。 iconCls: 新的图标css类。  $('#mm').menu('setIcon', {  target: $('#m-open')[0],  iconCls: 'icon-closed'  }); |
| findItem | text | 查找特定的菜单项，和getItem方法返回的对象相同。  // find 'Open' item and disable it  var item = $('#mm').menu('findItem', 'Open');  $('#mm').menu('disableItem', item.target); |
| appendItem | options | 添加菜单，'param'参数包含以下属性： parent: 待添加新菜单的DOM对象，如果没有被设置，新菜单项将会被作为顶级菜单添加。 text：字符串，菜单项文本。 href：字符串，超链接地址。 onclick：字符串或者函数，当用户点击菜单项时将要被执行的脚本代码或者函数。 iconCls: string, the icon class.  // append a top menu item  $('#mm').menu('appendItem', {  text: 'New Item',  iconCls: 'icon-ok',  onclick: function(){alert('New Item')}  });  // append a sub menu item  var item = $('#mm').menu('findItem', 'Open'); // find 'Open' item  $('#mm').menu('appendItem', {  parent: item.target, // the parent item element  text: 'Open Excel',  iconCls: 'icon-excel',  onclick: function(){alert('Open Excel')}  }); |
| removeItem | itemEl | 移除特定的菜单项。 |
| enableItem | itemEl | 启用菜单项。 |
| disableItem | itemEl | 禁用菜单项。 |

#### 链接按钮组件(kui-linkbutton)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| id（ID） | string（字符串） | 分配给该组件的ID。 | null |
| disabled（禁用） | boolean（布尔型） | 设置为true将禁用按钮。 | false |
| plain（简洁模式） | boolean（布尔型） | 设置为true将显示简洁效果。 | false |
| text（文本） | string（字符串） | 显示在按钮上的文本。 | '' |
| iconCls（图标CSS类） | string（字符串） | 用来在左边显示一个16x16大小图标的css类。 | null |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| disable | none | 禁用按钮。  $('#btn').linkbutton('disable'); |
| enable | none | 启用按钮。  $('#btn').linkbutton('enable'); |

#### 菜单按钮组件(kui-menubutton)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| plain（简洁模式） | boolean（布尔型） | 设置为true将显示简洁效果。 | true |
| menu（菜单） | string（字符串） | 用来创建一个相应菜单的选择器。 | null |
| duration（持续时间） | number（数字） | 定义鼠标划过按钮时显示菜单所持续的时间，单位为毫秒。 | 100 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| disable | none | 禁用菜单按钮。 |
| enable | none | 启用菜单按钮。 |
| destroy | none | 清除菜单按钮。 |

#### 分隔按钮组件(kui-splitbutton)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| plain（简洁模式） | boolean（布尔型） | 设置为true将显示简洁效果。 | true |
| menu（菜单） | string（字符串） | 用来创建一个相应菜单的选择器。 | null |
| duration（持续时间） | number（数字） | 定义鼠标划过按钮时显示菜单所持续的时间，单位为毫秒。 | 100 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| disable | none | 禁用分隔按钮。  $('#sb').splitbutton('disable'); |
| enable | none | 启用分隔按钮。  $('#sb').splitbutton('enable'); |
| destroy | none | 清除分隔按钮。 |

### 表单组件

#### 表单组件(kui-form)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| url（目标网址） | string（字符串） | 表单提交的目标网址，在调用提交表单的方法时，请传入一个url。 | null |
| req | Array（数组） | 表单提交的目标Service请求，在调用提交表单的方法时，请传入一个req。 | [] |
| queryCols | Array（数组） | 表单的输入框对象数组 | [] |
| toolbar | Array（数组） | 表单的按钮对象数组 | [] |
| colNumbers | int | 表单的输入框每一行对象列数 | 3 |
| requiredGrp | String | 必须同时输入对象名称串 | "" |
| tipMessage | String | 必须同时输入不满足提示 | "" |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onSubmit | none | 在表单提交之前触发，返回false将阻止表单提交。 |
| success | data | 当表单成功提交时触发。 |
| onBeforeLoad | param | 在发送载入远程数据请求之前触发，返回false将取消请求。 |
| onLoadSuccess | data | 在表单数据被载入时触发。 |
| onLoadError | none | 在载入表单数据发生错误时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| submit | options | 执行表单提交，options参数是一个包含以下属性的对象： url：表单提交的目标网址 req: 表单提交的目标req onSubmit：在表单提交之前执行的回调函数 success: 在表单成功提交后执行的回调函数  下面这个例子展示了如何提交一个有效的表单以及避免重复提交表单。  $.messager.progress(); // 显示进度条  $('#ff').form('submit', {  req: [{service:'GeneralPojoService',bex\_codes:'LBM\_userInfoQuery,  LBM\_userBaseInfoQuery',OP\_CODE:userCode,USER\_CODE:userCode}],  onSubmit: function(){  var isValid = $(this).form('validate');  if (!isValid){  $.messager.progress('close');  // hide progress bar while the form is invalid  }  return isValid; // 返回false将停止表单的提交  },  success: function(){  $.messager.progress('close');  // 表单提交成功时隐藏进度条。  }  }); |
| load | data | 载入记录填充表单。 从配置的loadUrl或req加载表单数据，后面不带参数默认为配置的queryParam参数，如果后面带参数则会覆盖queryParam中的参数。  $('#ff').form('load'); // load from loadurl or req  $('#ff').form('load',{id:1}); |
| clear | none | 清除表单数据 |
| reset | none | 表单还原默认值函数 |
| validate | none | 执行表单域验证，如果全部表单域有效返回true，该方法与表单验证插件一起使用。 |

#### 组合框组件(kui-combo)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| width（宽度） | number（数字） | 组合框的宽度。 | auto |
| panelWidth  （下拉框宽度） | number（数字） | 下拉框的宽度。 | null |
| panelHeight（下拉框高度） | number（数字） | 下拉框的高度。 | 200 |
| multiple  （可多选） | boolean（布尔型） | 定义是否支持多选。 | false |
| separator  （分隔条） | string（字符串） | 当支持多选时，用以分隔文本的字符。 | , |
| editable  （可编辑） | boolean（布尔型） | 定义用户是否可以直接向组合框输入内容。 | true |
| disabled（禁用） | boolean（布尔型） | 定义是否禁用输入框。 | false |
| hasDownArrow（下拉图标） | boolean（布尔型） | 定义是否显示下拉按钮。 | true |
| delay（延迟） | number（数字） | 最后一次输入事件与搜索之间的延迟间隔。 | 200 |
| keyHandler  （按键助手） | object（对象） | 当用户按下一个键位时调用的函数，默认的按键助手定义如下：  keyHandler: {  up: function(){},  down: function(){},  enter: function(){},  query: function(q){}  } |  |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onShowPanel | none | 当显示下拉面板的时候触发。 |
| onHidePanel | none | 当隐藏下拉面板的时候触发。 |
| onChange | newValue, oldValue | 当组合框的值发生改变时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| panel | none | 返回下拉面板对象。 |
| textbox | none | 返回文本框对象。 |
| destroy | none | 销毁组件。 |
| resize | width | 调整组件的宽度。 |
| showPanel | none | 显示下拉面板。 |
| hidePanel | none | 隐藏下拉面板。 |
| disable | none | 禁用组合框。 |
| enable | none | 启用组合框。 |
| validate | none | 验证输入的值。 |
| isValid | none | 返回验证结果。 |
| clear | none | 清除组件的值。 |
| getText | none | 获取组合框的值包括代码和文字部分。 |
| setText | text | 设置输入框的值。 |
| getValues | none | 获取选中项前面的code值（多选时使用），结果为数组。 |
| setValues | values | 设置下拉面板选中的值（多选使用），参数values是一个数组。 |
| getValue | none | 获取选中项前面的code值。 |
| setValue | value | 设置选中的值，参数为下拉面板中选项的代码号。 |
| getName | none | 获取选中项后面的文字部分。 |
| getNames | none | 获取选中项后面的文字部分（多选），结果为数组。 |

#### 下拉组合框组件(kui-combobox)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| valueField（值域） | string（字符串） | 绑定到组合框下的数据的值域的名称。 | value |
| textField（文本域） | string（字符串） | 绑定到组合框下数据的文本域的名称。 | text |
| url（超链接地址） | string（字符串） | 可供载入数据列表的远程地址。 | null |
| method（方法） | string（字符串） | 获取数据的HTTP方法。 | post |
| data（数据） | array（数组） | 将被载入的数据列表。  <input class="kui-combobox" kui-options="  valueField: 'label',  textField: 'value',  data: [{  label: 'java',  value: 'Java'  },{  label: 'perl',  value: 'Perl'  },{  label: 'ruby',  value: 'Ruby'  }]" /> | null |
| dict（数据字典） | string（字符串） | 将被载入数据的数据字典。 |  |
| req（后台请求对象数组） | array（数组） | 请求对象数组。 | [] |
| sort（排序） | function | 自定义排序方法。 |  |
| autoFill（默认选中数据） | boolean | 初始化时默认选中第一条数据。 | false |
| editable | string（字符串） | 是否可编辑。 |  |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onBeforeLoad | param | 在加载数据请求前触发，反回false将取消加载行为。 |
| onLoadSuccess | none | 当远程数据载入成功时触发。 |
| onLoadError | none | 当远程数据载入错误时触发。 |
| onSelect | record | 当用户选择一个列表项时触发。 |
| onUnselect | record | 当用户取消选择一个列表项时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| getData | none | 返回已载入的数据。 |
| loadData | data | 载入本地数据列表。 |
| reload | url | 请求远程数据列表。 |
| setValues | values | 设置组合框的值，参数values是一个数组。  $('#cc').combobox('setValues', ['001','002']); |
| setValue | value | 设置组合框的值。  $('#cc').combobox('setValues', '001'); |
| clear | none | 清除组合框的值。 |
| select | value | 选择特定的项目。 |
| unselect | value | 取消选择特定的项目。 |
| setDict | value | 设置组合框的数据字典。 |

#### 下拉组合树组件(kui-combotree)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| editable（可编辑） | boolean（布尔型） | 定义用户是否可以直接输入。 | false |

* **事件**

事件继承组合框和树形菜单

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| tree | none | 返回树形菜单对象。  var t = $('#cc').combotree('tree'); // 得到树对象  var n = t.tree('getSelected'); // 得到选中的节点  alert(n.text); |
| loadData | data | 载入本地树形菜单数据。  $('#cc').combotree('loadData', [{  id: 1,  text: 'Languages',  children: [{  id: 11,  text: 'Java'  },{  id: 12,  text: 'C++'  }]  }]); |
| reload | url | 再次向远程树形菜单数据发起请求。 |
| clear | none | 清除组件值。 |
| setValues | values | 设置组合框的值，参数values是一个数组。  $('#cc').combotree('setValues', [1,3,21]); |
| setValue | value | 设置组合框的值。  $('#cc').combotree('setValues', 1); |

#### 下拉组合表格(kui-combogrid)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| loadMsg（载入时信息） | string（字符串） | 当数据表格载入远程数据时的提示信息。 | null |
| idField（id字段） | string（字符串） | 标识表格行数据唯一性的字段名。 | null |
| textField（文本域） | string（字符串） | 显示在文本框中的文本域的名称。 | null |
| mode（模式） | string（字符串） | 定义当文本发生改变时如何载入组合表格的数据。如果组合表格要从远程服务器载入数据，设置为'remote'。 | local |
| filter（过滤器） | function(q, row) | 定义当模式设置为'local'时如何选择本地数据，返回true将选择一行。  $('#cc').combogrid({  filter: function(q, row){  var opts = $(this).combogrid('options');  return row[opts.textField].indexOf(q) == 0;  }  }); |  |

* **事件**

事件继承组合框和数据表格。

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| grid | none | 返回数据表格对象。 |
| setValues | values | 设置组合框的值，参数values是一个数组。 |
| setValue | value | 设置组合框的值 |
| clear | none | 清除组合框的值。 |

### 窗口组件

#### 窗口组件(kui-window)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| title（标题） | string（字符串） | 窗口的标题。 | New Window |
| collapsible（可折叠） | boolean（布尔型） | 定义是否显示折叠按钮。 | true |
| minimizable（最小化） | boolean（布尔型） | 定义是否显示最小化按钮。 | true |
| maximizable（最大化） | boolean（布尔型） | 定义是否显示最大化按钮。 | true |
| closable（关闭） | boolean（布尔型） | 定义是否显示关闭按钮。 | true |
| closed（已关闭） | boolean（布尔型） | 定义是否在初始化组件时关闭（隐藏）窗口 | false |
| zIndex（堆叠顺序） | number（数字） | 窗口的堆叠顺序，从第一个窗口的zIndex值开始递增。 | 9000 |
| draggable（可拖放） | boolean（布尔型） | 定义窗口是否可以被拖放。 | true |
| resizable（可缩放） | boolean（布尔型） | 定义窗口是否可以被缩放。 | true |
| shadow（阴影） | boolean（布尔型） | 如果设置为true，显示窗口的时候将显示阴影。 | true |
| inline（行内显示） | boolean（布尔型） | 定义如何布局窗口，如果设置为true，窗口将显示在它的父容器中，否则将显示在所有元素的上面。 | false |
| modal（遮罩） | boolean（布尔型） | 定义窗口是否带有遮罩效果。 | true |

* **事件**

事件继承控制面板。

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| window | none | 返回窗口对象。 |
| resize | param | 设置窗口大小。 |
| move | param | 移动窗口位置。 |

#### 对话窗口(kui-dialog)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| title（标题） | string  （字符串） | 对话标题。 | New Dialog |
| collapsible  （可折叠） | boolean  （布尔型） | 定义是否显示折叠按钮。 | false |
| minimizable  （最小化） | boolean  （布尔型） | 定义是否显示最小化按钮。 | false |
| maximizable  （最大化） | boolean  （布尔型） | 定义是否显示最大化按钮。 | false |
| resizable（可缩放） | boolean  （布尔型） | 定义对话窗口是否可以被缩放。 | false |
| toolbar（工具栏） | array（数组） | 对话窗口顶部的工具栏，每个工具的属性都跟链接按钮的属性一样。  对话框的工具栏可以用  标签声明：  <div class="kui-dialog" style="width:600px;height:300px"  kui-options="title:'My Dialog',toolbar:'#tb',modal:true">  Dialog Content.  </div>  <div id="tb">  <a href="#" class="kui-linkbutton" kui-options="iconCls:'icon-edit',plain:true"/a>  <a href="#" class="kui-linkbutton" kui-options="iconCls:'icon-help',plain:true"/a>  </div>  对话框的工具栏也可以通过数组来定义：  <div class="kui-dialog" style="width:600px;height:300px"  kui-options="title:'My Dialog',modal:true,  toolbar:[{  text:'Edit',  iconCls:'icon-edit',  handler:function(){alert('edit')}  },{  text:'Help',  iconCls:'icon-help',  handler:function(){alert('help')}  }]">  Dialog Content.  </div> | null |
| buttons（按钮） | array（数组） | 对话窗口底部的按钮，每个按钮的属性都跟链接按钮一样。  按钮可以通过div标签来定义：  <div class="kui-dialog" style="width:600px;height:300px"  kui-options="title:'My Dialog',buttons:'#bb',modal:true">  Dialog Content.  </div>  <div id="bb">  <a href="#" class="kui-linkbutton">Save</a>  <a href="#" class="kui-linkbutton">Close</a>  </div>  按钮也可以通过数组来定义：  <div class="kui-dialog" style="width:600px;height:300px"  kui-options="title:'My Dialog',modal:true,  buttons:[{  text:'Save',  handler:function(){...}  },{  text:'Close',  handler:function(){...}  }]">  Dialog Content.  </div> | null |

* **事件**

事件继承窗口。

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| dialog | none | 返回对话窗口对象。 |
| refresh | none | 刷新对话窗口。 |
| getContent | none | 得到对话框中第一个子元素对象。 |

#### 消息窗口(kui-message)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| ok（确认） | string（字符串） | 显示在“确认”按钮上的文本。 | Ok |
| cancel（取消） | string（字符串） | 显示在“取消”按钮上的文本。 | Cancel |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| $.messager.show | options | 在屏幕的右下方显示一个消息窗口 ，options参数是一个可配置的对象: showType：定义如何显示消息窗口，可选值: null,slide,fade,show，默认为slide。 showSpeed：定义窗口显示的时间，单位毫秒，默认为 600。 width：定义消息窗口的宽度，默认为250。 height：定义消息窗口的高度，默认为100。 msg：显示在消息窗口的文本。 title：显示在窗口头部的标题。 timeout：如果定义为0，消息窗口将不会关闭直到用户点击关闭为止。如果定义为非0，  消息窗口在超时后将自动关闭。 示例代码：  $.messager.show({  title:'My Title',  msg:'Message will be closed after 5 seconds.',  timeout:5000,  showType:'slide'  });  // show message window on top center  $.messager.show({  title:'My Title',  msg:'Message will be closed after 4 seconds.',  showType:'show',  style:{  right:'',  top:document.body.scrollTop+  document.documentElement.scrollTop,  bottom:''  }  }); |
| $.messager.alert | title, msg, icon, fn | 显示警告窗口。参数如下: title：显示在窗口头部的标题文本。 msg：显示在窗口中的文本。 icon：显示的图片，可选值：error,question,info,warning。 fn：当窗口关闭时触发的回调函数。  $.messager.alert('My Title','Here is a info message!','info'); |
| $.messager.confirm | title, msg, fn | 显示一个带有确认和取消按钮的确认信息窗口。参数如下： title：显示在窗口头部的文本。 msg：显示在窗口中的文本。 fn(b)：回调函数，当用户点击确认按钮时，传递一个true值给回调函数，否则传递一个false值。  $.messager.confirm('Confirm', 'Are you sure to exit this system?', function(r){  if (r){  // exit action;  }  }); |
| $.messager.prompt | title, msg, fn | 显示一个带有确认和取消的输入信息窗口。 参数如下： title：显示在窗口头部的标题文本。 msg：显示在窗口中的信息 fn(val)：接受用户输入作为参数的回调函数。 |
| $.messager.progress | options or method | 显示一个带进度条信息的窗口。 属性定义如下： title：显示在面板头部的标题文本，默认为''。 msg：显示在主窗体的文本，默认为''。 text：显示在进度条中的信息默认为undefined。 interval：每次进度增加所耗费的时间，单位为毫秒，默认为300.  方法定义如下： bar：获取进度条对象。 close：关闭进程窗口。  **示例代码**  显示进程消息窗口。  $.messager.progress();  使用如下代码关闭窗口。  $.messager.progress('close'); |

### 数据表格和树形菜单组件

#### 数据表格组件(kui-datagrid)

* **数据表格属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| columns（列） | array（数组） | 数据表格列配置对象，查看列属性以获取更多细节。 | null |
| frozenColumns（固定列） | array（数组） | 跟列属性一样，但是这些列固定在左边，不会滚动。 | null |
| fitColumns  （自适应列宽） | boolean（布尔型） | 设置为true将自动使列适应表格宽度以防止出现水平滚动。 | false |
| striped（显示条纹） | boolean（布尔型） | 设置为true将交替显示行背景。 | false |
| method（方法） | string（字符串） | 请求远程数据的方法类型。 | post |
| nowrap（截取） | boolean（布尔型） | 设置为true，当数据长度超出列宽时将会自动截取。 | true |
| idField（id字段） | string（字符串） | 表明该列是一个唯一列。 | null |
| url（超链接） | string（字符串） | 一个用以从远程站点请求数据的超链接地址。 | null |
| req | array | 数据请求对象数组。 | [] |
| loadMsg  （载入时信息） | string（字符串） | 当从远程站点载入数据时，显示的一条快捷信息。 | Processing, please wait … |
| pagination  （分页） | boolean（布尔型） | 设置true将在数据表格底部显示分页工具栏。 | false |
| pagePosition（分页工具栏位置） | string（字符串） | 设置分页工具栏显示的位置。值可以为：'top','bottom','both' | bottom |
| toolbar  （工具栏） | 数组，选择器 | 数据表格顶部的工具栏。可以的值：数组，每个工具参数和linkbutton一样。能指定工具栏的选择器。  用div标签定义工具栏：  $('#dg').datagrid({  toolbar: '#tb'  });  <div id="tb">  <a href="#" class="kui-linkbutton" kui-options="iconCls:'icon-edit',plain:true"/a>  <a href="#" class="kui-linkbutton" kui-options="iconCls:'icon-help',plain:true"/a>  </div>  通过数组定义工具栏：  $('#dg').datagrid({  toolbar: [{  iconCls: 'icon-edit',  handler: function(){alert('edit')}  },'-',{  iconCls: 'icon-help',  handler: function(){alert('help')}  }]  }); | null |
| checkOnSelect（选勾） | boolean（布尔型） | 为true时，当用户选中/不选中一行时复选框会同时选中/不选中。当为false时，复选框只有在用户单击了复选框的时候才会选中/不选中。 | true |
|  |  |  |  |
| selectOnCheck（勾选） | boolean（布尔型） | 为true时，当用户选中/不选中复选框会同时选中/不选中该行。当为false时，当用户选中/不选中复选框不会同时选中/不选中该行。 | true |
| rownumbers（行数） | boolean（布尔型） | 设置为true将显示行数。 | false |
| singleSelect  （单选模式） | boolean（布尔型） | 设置为true将只允许选择一行。 | false |
| pagePosition | string | 定义分页按钮的位置。可用值：“top”，“bottom”，“both” | bottom |
| pageNumber（当前页码） | number（数字） | 当设置分页属性时，初始化分页码。 | 1 |
| pageSize  （每页记录数） | number（数字） | 当设置分页属性时，初始化每页记录数。 | 10 |
| pageList（可选择的每页记录数） | array | 当设置分页属性时，初始化每页记录数列表。 | [10,20,30,40,50] |
| queryParams（查询参数） | object（对象） | 当请求远程数据时，发送的额外参数。 | {} |
| sortName  （默认排序） | string（字符串） | 当数据表格初始化时以哪一列来排序。 | null |
| sortOrder  （排序顺序） | string（字符串） | 定义排序顺序，可以是'asc'或者'desc'（正序或者倒序）。 | asc |
| remoteSort  （远程排序） | boolean（布尔型） | 定义是否通过远程服务器对数据排序。 | true |
| showHeader（显示表头） | boolean（布尔型） | 定义是否显示表头。 | true |
| showFooter  （显示行底） | boolean（布尔型） | 定义是否显示行底（如果是做统计表格，这里可以显示总计等）。 | false |
| scrollbarSize（滚动条大小） | number（数字） | 滚动条的宽度（当垂直滚动条显示的时候）或高度（当水平滚动条显示的时候）。 | 18 |
| allowMove  （移动） | boolean（布尔型） | 是否可以移动表格，true可以，false不可以。 | false |
| rowStyler  （行样式） | function（函数） | 返回样式，如：'background:red'，function有2个参数： index：行索引，从0开始. row：对应于该行记录的对象。 |  |
| loader（加载器） | function（函数） | 定义如何从远程加载数据，返回false可以终止该行为。函数可以有以下参数： param:传到远程服务器的参数对象。 success(data):成功返回数据时调用的回调函数。 error():当返回数据失败时将调用的回调函数。 | json loader |
| loadFilter  （载入过滤器） | function（函数） | 返回用以显示的已过滤数据，function有一个参数'data'表示原始数据，你可以将原始数据改变为规范的数据格式，该函数必须返回包含 'total'和'rows'属性的标准数据对象。 |  |
| editors  （编辑模式） | object（对象） | 定义当编辑一行时的编辑模式。 | predefined editors |
| view（视图） | object（对象） | 定义数据表格的视图。 | default view |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onLoadSuccess | data | 当数据载入成功时触发。 |
| onLoadError | none | 当载入远程数据发生错误时触发。 |
| onBeforeLoad | param | 在请求载入数据之前触发，如果返回false将取消载入。 |
| onClickRow | rowIndex, rowData | 当用户点击行时触发，参数如下： rowIndex：被点击的行索引，从0开始。 rowData：对应于被点击的行的记录。 |
| onDblClickRow | rowIndex, rowData | 当用户双击一行时触发，参数如下： rowIndex：被点击的行索引，从0开始。 rowData：对应于被点击的行的记录。 |
| onClickCell | rowIndex, field, value | 当用户点击单元格时触发。 |
| onDblClickCell | rowIndex, field, value | 当用户双击单元格时触发。 |
| onSortColumn | sort, order | 当用户对列排序时触发，参数如下： sort：排序字段名称。 order：排序顺序。 |
| onResizeColumn | field, width | 当用户调整列宽时触发。 |
| onSelect | rowIndex, rowData | 当用户选择一行是触发，参数如下： rowIndex：被选择的行索引，从0开始。 rowData：对应于被选择行的记录。 |
| onUnselect | rowIndex, rowData | 当用户取消选择一行时触发，参数如下： rowIndex：被取消选择的行索引，从0开始。 rowData：对应于被取消选择行的记录。 |
| onSelectAll | rows | 当用户选择所有行时触发。 |
| onUnselectAll | rows | 当用户取消选择所有行时触发。 |
| onCheck | rowIndex,rowData | 当用户选中一行时触发。参数： rowIndex:选中行的索引，0开始。 rowData:被选中行对应的记录。 |
| onUncheck | rowIndex,rowData | 当用户取消选中一行时触发。参数： rowIndex:取消选中行的索引，0开始。 rowData:取消选中行对应的记录。 |
| onCheckAll | rows | 当用户选中全部行时触发。 |
| onUncheckAll | rows | 当用户取消选中全部行时触发。 |
| onBeforeEdit | rowIndex, rowData | 当用户开始编辑一行时触发，参数如下： rowIndex：正在编辑的行索引，从0开始。 rowData：对应于正在编辑的行的记录。 |
| onAfterEdit | rowIndex, rowData, changes | 当用户编辑完成时触发，参数如下： rowIndex：正在编辑的行索引，从0开始。 rowData：对应于正在编辑的行的记录。 changes：被改变的字段内容，对应方式为字段：值。 |
| onHeaderContextMenu | e, field | 当数据表格的列标题被鼠标右键单击时触发。 |
| onRowContextMenu | e, rowIndex, rowData | 当一行被鼠标右键单击时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回属性对象。 |
| getPager | none | 返回页面对象。 |
| getPanel | none | 返回控制面板对象。 |
| getColumnFields | frozen | 返回列字段，如果设置了frozen属性为true，将返回固定列的字段名。 |
| getColumnOption | field | 返回特定的列属性。 |
| resize | param | 缩放和布局。 |
| loading | none | 显示载入状态。 |
| loaded | none | 隐藏载入状态。 |
| fitColumns | none | 让列宽自动适应数据表格的宽度。 |
| fixColumnSize | none | 固定列尺寸。 |
| fixRowHeight | index | 固定特定列的高度。 |
| autoSizeColumn | field | 列宽自适应内容长度。 |
| getRows | none | 返回当前页的记录。 |
| getRowIndex | row | 返回指定行的索引，row参数可以是行记录或者是一个id字段的值。 |
| getSelected | none | 返回第一个被选择的行记录或null。 |
| getSelections | none | 返回所有被选择的行，当没有记录被选择时，将返回一个空数组。 |
| clearSelections | none | 取消所有的已选择项。 |
| selectAll | none | 全选。 |
| unselectAll | none | 取消全选。 |
| selectRow | index | 选择一行，行索引从0开始。 |
| selectRecord | idValue | 通过传递id参数来选择一行。 |
| unselectRow | index | 取消选择一行。 |
| beginEdit | index | 开始编辑一行。 |
| endEdit | index | 结束编辑。 |
| refreshRow | index | 刷新一行。 |
| validateRow | index | 校验指定的行，如果有效返回true。 |
| updateRow | param | 更新指定的行，param参数包含如下属性： index：要更新的行索引。 row：新的行数据。 |
| appendRow | row | 添加一行。 |
| insertRow | param | 插入一个新行，param参数包含如下属性： index：要插入的行索引，如果没有定义则在最后面添加一个新行。 row：行数据。 |
| deleteRow | index | 删除一行。 |
| getChanges | type | 获取从最后一次提交开始的被修改的所有行，type参数表明修改的类型，可选值：inserted，deleted，updated等 。当没有传递type参数时，返回所有被修改的行。 |
| acceptChanges | none | 提交所有修改的数据，提交后的数据将不能再修改或者回滚。 |
| rejectChanges | none | 回滚所有被删除的行。 |
| mergeCells | options | 合并单元格，options参数包含如下属性： index：行索引。 field：字段名。 rowspan：整合单元格要跨的行数。 colspan：整合单元格要跨的列数。 |
| showColumn | field | 显示特定的列。 |
| hideColumn | field | 隐藏特定的列。 |

#### 树形菜单组件(kui-tree)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| url（超链接） | string（字符串） | 用以载入远程数据的超链接地址。 | null |
| method（方法） | string（字符串） | 获取数据的HTTP方法。 | post |
| animate（动画） | boolean（布尔型） | 定义当节点打开或关闭时是否显示动画效果。 | false |
| checkbox  （复选框） | boolean（布尔型） | 定义是否在每个节点之前显示复选框。 | false |
| cascadeCheck（级联选择） | boolean（布尔型） | 定义是否支持级联选择。 | true |
| onlyLeafCheck（只选叶子节点） | boolean（布尔型） | 定义是否只在叶子节点之前显示复选框。 | false |
| dnd（拖放） | boolean（布尔型） | 定义是否支持拖放。 | false |
| data（数据） | array（数组） | 将被载入的节点数据。  $('#tt').tree({  data: [{  text: 'Item1',  state: 'closed',  children: [{  text: 'Item11'  },{  text: 'Item12'  }]  },{  text: 'Item2'  }]  }); | null |
| lines（节点间连接的虚线） | boolean（布尔型） | 图标显示为加减，并显示节点间相连的虚线。 | false |
| loader | function(param,success,error) | 定义如何从远程服务器获取数据。函数参数： param:传到服务器的参数对象。 success(data):成功得到数据后执行的回调函数。 error():失败时执行的回调函数。 | json loader |
| loadFilter | function(data,parent) | 返回过滤后用以显示的数据，返回的数据是标准的树格式。函数参数： data:用以加载的原始数据。 parent:DOM对象，指明父节点。 | json loader |
| req | array | 请求后台的req对象数组 | "" |
| field | object | field:{'id':'id','parentid':'parentid','text':'text'} id为当前节点的value值 parentid为当前节点的父节点 text为当前节点显示的文本 | {} |

* **事件**

多数事件回调函数都有'node'参数，该参数包含如下属性：

* id：节点的唯一标识。
* text：显示在节点上的文本。
* checked：节点是否被选择。
* attributes：节点的自定义属性。
* target：目标DOM对象。

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onClick | node | 当用户点击节点时触发，node参数包含如下属性： id：节点id。 text：显示在节点上的文本。 checked：节点是否被选择。 attributes：节点的自定义属性。 target：被点击的目标DOM对象。 |
| onDblClick | node | 当用户双击节点时触发。 |
| onBeforeLoad | node, param | 在请求载入数据之前触发，返回false将取消载入。 |
| onLoadSuccess | node, data | 当数据载入成功时触发。 |
| onLoadError | arguments | 当数据载入失败时触发，arguments参数跟jQuery.ajax的'error'函数一样。 |
| onBeforeExpand | node | 在节点打开之前触发，返回false将取消打开。 |
| onExpand | node | 在节点被打开时触发。 |
| onBeforeCollapse | node | 在节点被关闭之前触发，返回false将取消关闭。 |
| onCollapse | node | 当节点被关闭时触发。 |
| onCheck | node, checked | 当用户点击复选框时触发。 |
| onBeforeSelect | node | 在节点被选择之前触发，返回false将取消选择。 |
| onSelect | node | 当节点被选择时触发。 |
| onContextMenu | e, node | 当节点被鼠标右键点击时触发。 |
| onDrop | target, source, point | 当节点位置被（拖动）更换时触发。 target：DOM对象，需要被拖动动的目标节点。 source：原始节点。 point：指明拖动方式，可选值：'append'，'top'或者'bottom'。 |
| onBeforeEdit | node | 在编辑节点之前触发。 |
| onAfterEdit | node | 在编辑节点之后触发。 |
| onCancelEdit | node | 当取消编辑时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回树形菜单属性对象。 |
| loadData | data | 载入树形菜单数据。 |
| getNode | target | 获取特定的节点对象。 |
| getData | target | 获取特定的节点数据，包括它的子节点。 |
| reload | target | 重新载入树形菜单数据。 |
| getRoot | none | 获取根节点，返回节点对象。 |
| getRoots | none | 获取根节点，返回节点数组。 |
| getParent | target | 获取父节点，target是一个节点DOM对象。 |
| getChildren | target | 获取子节点，target参数是一个节点DOM对象。 |
| getChecked | none | 获取所有被选择的节点。 |
| getSelected | none | 获取被选择的节点并返回，如果没有节点被选择则返回null。 |
| isLeaf | target | 判断指定的节点是否是叶子节点，target参数是一个节点DOM对象。 |
| find | id | 查找指定的节点并返回节点对象。 |
| select | target | 选择一个节点，target参数是一个节点DOM对象。 |
| check | target | 设置指定的节点为已选择状态。 |
| uncheck | target | 设置指定的节点为未选择状态。< |
| collapse | target | 关闭节点，target参数是一个节点DOM对象。 |
| expand | target | 打开节点，target参数是一个节点DOM对象。 |
| collapseAll | target | 关闭所有的节点。 |
| expandAll | target | 打开所有的节点。 |
| expandTo | target | 打开从根节点到指定节点之间的所有节点。 |
| append | param | 添加若干子节点到一个父节点，param参数有2个属性： parent：DOM对象，将要被添加子节点的父节点，如果未指定，子节点将被添加至根节点。 data：数组，节点数据。 |
| toggle | target | 打开或关闭节点的触发器，target参数是一个节点DOM对象。 |
| insert | param | 在一个指定节点之前或之后插入节点，'param'参数包含如下属性： before：DOM对象，在某个节点之前插入。 after: DOM对象，在某个节点之后插入。 data：对象，节点数据。 |
| remove | target | 移除一个节点和它的子节点，target参数是一个节点DOM对象。 |
| pop | target | 移除一个节点和它的子节点，该方法跟remove方法一样，不同的是它将返回被移除的节点数据。 |
| update | param | 更新指定的节点，param参数有如下属性： target(DOM对象，将被更新的目标节点)，id，text，iconCls，checked等。 |
| enableDnd | none | 启用拖动特性。 |
| disableDnd | none | 禁用拖动特性。 |
| beginEdit | nodeEl | 开始编辑节点。 |
| endEdit | nodeEl | 结束编辑节点。 |
| cancelEdit | nodeEl | 取消编辑节点。 |

#### 树形表格(kui-treegird)

* **属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **描述** | **默认值** |
| idfield | string（字符串） | 定义了关键字段来确定一个树节点。必填 | null |
| treeField（树形字段） | string（字符串） | 定义作为树节点的字段。必填 | null |
| loader | function(param,success,error) | 定义如何从远程服务器加载数据。返回false可以中止这一行动。这个函数接受以下参数： param: 这个参数对象传递到远程服务器. success(data):这个回调函数,它将检索数据时调用成功. error(): 这个回调函数,它将检索数据时调用失败. | json loader |
| loadFilter | function(data,parentId) | 返回过滤数据来显示 |  |
| animate（动画） | boolean（布尔型） | 定义当打开或关闭节点时是否显示动画效果。 | false |
| singleSelect（单选） | boolean（布尔型） | 为true时可以选中多行，false只能选中一行。 | true |

* **事件**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| onClickRow | row | 当用户点击节点时触发 。 |
| onDblClickRow | row | 当用户双击节点时触发 。 |
| onClickCell | field,row | 当用户点击一个单元格时触发 。 |
| onDblClickCell | field,row | 当用户双击一个单元格时触发 。 |
| onBeforeLoad | row, param | 在请求载入数据之前触发，返回false将取消载入。 |
| onLoadSuccess | row, data | 当数据载入成功时触发。 |
| onLoadError | arguments | 当数据载入失败时触发，arguments参数和jQuery.ajax的'error'函数一样。 |
| onBeforeExpand | row | 在节点打开之前触发 ，返回false将取消打开。 |
| onExpand | row | 当节点打开时触发。 |
| onBeforeCollapse | row | 在节点关闭之前触发，返回false将取消关闭。 |
| onCollapse | row | 当节点关闭时触发。 |
| onContextMenu | e, row | 当节点被鼠标右键点击时触发 。 |
| onBeforeEdit | row | 当用户开始编辑节点时触发。 |
| onAfterEdit | row,changes | 当用户结束编辑时触发。 |
| onCancelEdit | row | 当用户取消编辑节点时触发。 |

* **方法**

|  |  |  |
| --- | --- | --- |
| **名称** | **参数** | **描述** |
| options | none | 返回树形表格的属性对象。 |
| resize | options | 设置树形表格的大小，options包含2个属性： width：新的树形表格宽度 height：新的树形表格高度。 |
| fixRowHeight | id | 固定特定行的高。 |
| loadData | data | 载入树形表格数据。 |
| reload | id | 重新载入树形表格数据。  $('#tt').treegrid('reload', 2); // reload the row which value is equals to 2  $('#tt').treegrid('reload'); // reload the all rows |
| reloadFooter | footer | 重新载入树形表格的行底数据。 |
| getData | none | 获取已载入的数据。 |
| getFooterRows | none | 获取行底数据。 |
| getRoot | none | 获取根节点，返回节点对象。 |
| getRoots | none | 获取根节点，返回节点数组。 |
| getParent | id | 获取父节点。 |
| getChildren | id | 获取子节点。 |
| getSelected | none | 获取被选择节点并返回，如果没有节点被选择则返回null。 |
| getSelections | none | 获取所有被选择的节点。 |
| getLevel | id | 获取指定的节点的层级。 |
| find | id | 查找指定的节点并返回节点数据。 |
| select | id | 选择一个节点。 |
| unselect | id | 取消选择一个节点。 |
| selectAll | none | 选择所有的节点。 |
| unselectAll | none | 取消选择所有的节点。 |
| collapse | id | 关闭节点。 |
| expand | id | 打开节点。 |
| collapseAll | id | 关闭所有节点。 |
| expandAll | id | 打开所有节点。 |
| expandTo | id | 打开从根节点到指定节点直接的所有节点。 |
| toggle | id | 打开或关闭节点的触发器。 |
| append | param | 添加若干子节点到一个父节点，'param'参数包含如下属性： parent：DOM对象，将要被添加子节点的父节点，如果未指定，子节点将被添加至根节点。 data：数组，节点数据。  // append some nodes to the selected row  var node = $('#tt').treegrid('getSelected');  $('#tt').treegrid('append',{  parent: node.id, // the node has a 'id' value that defined through 'idField' property  data: [{  id: '073',  name: 'name73'  }]  }); |
| remove | id | 移除一个节点和它的子节点。 |
| refresh | id | 属性指定的节点。 |
| beginEdit | id | 开始编辑节点。 |
| endEdit | id | 结束编辑节点。 |
| cancelEdit | id | 取消编辑节点。 |
| getEditors | id | 获取指定行的编辑器，每个编辑器都有如下属性： actions：编辑器可以做的行为。 target：目标编辑器jQuery对象。 field：字段名。 type：编辑类型。 |
| getEditor | options | 获取指定的编辑器，options包含2个属性： id：行节点 id。 field：字段名。 |

# 业务流程设计

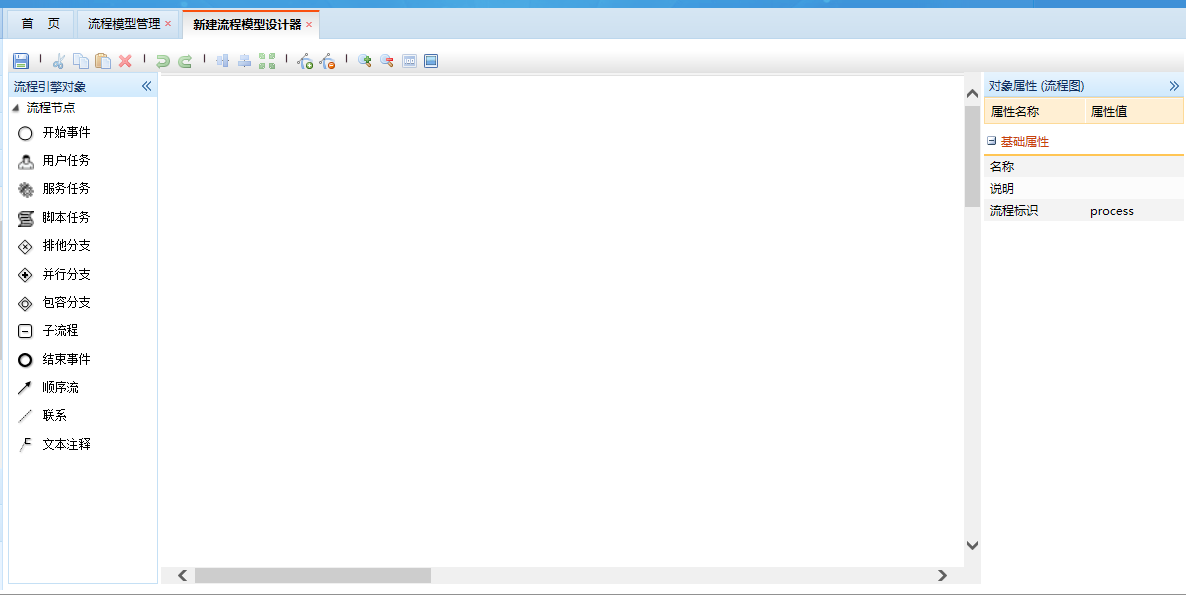
## Web流程设计器

### 设计器入口

登陆管理平台，选择流程管理—》选择流程模型管理—》创建—》输入流程标识、流程名称、描述—》创建，如下所示：



创建成功后，自动进入流程设计器：



流程设计器工作区包括以下四部分：

顶部区域：工作流常用工具

左边区域：流程节点对象

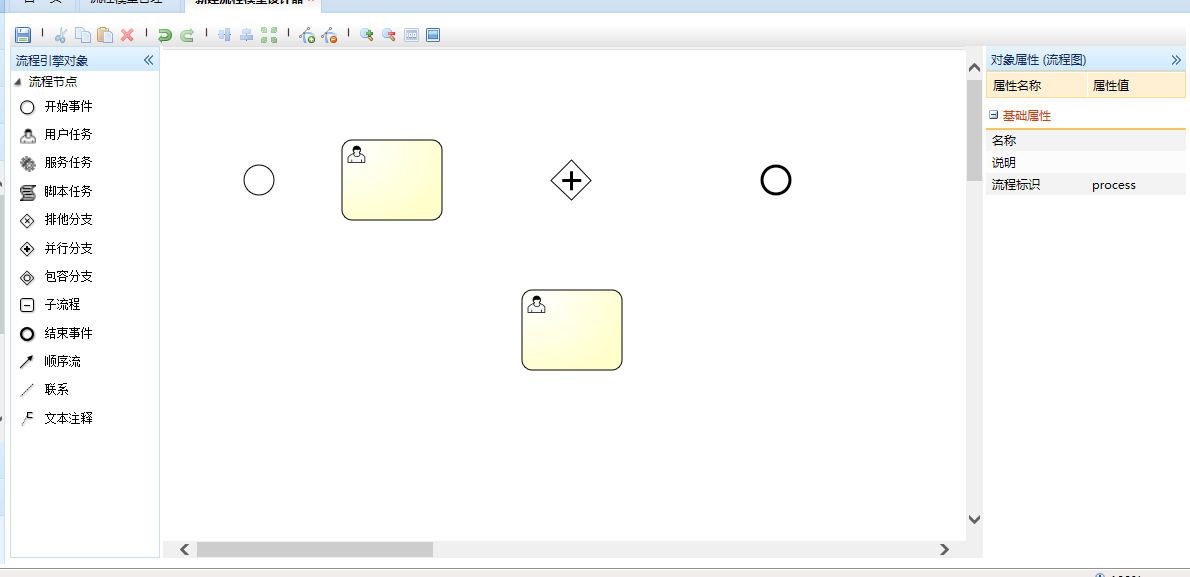
中间区域：流程编排区

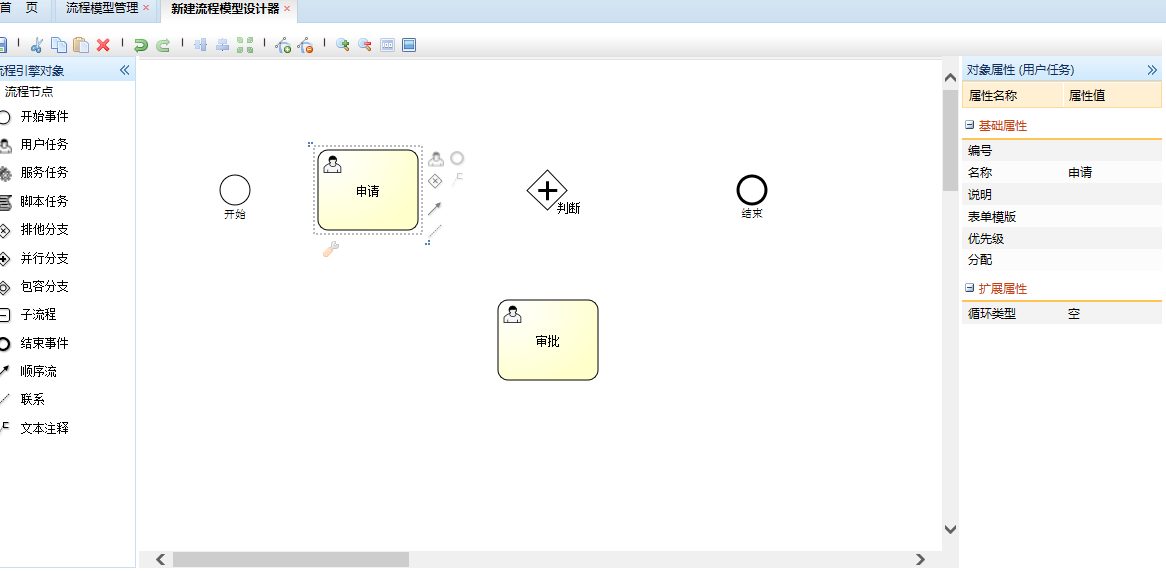
右边区域：流程节点属性编辑区

可以在以上描述区域内进行流程的设计。

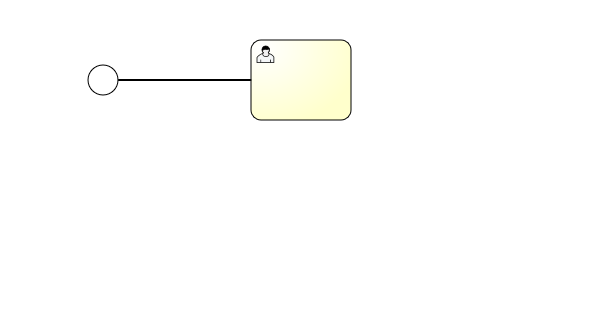
### 设计器使用

第一步：选择左边的流程节点，并拖拽到流程编排区

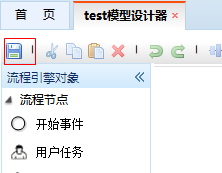
第二步：依次给每个节点输入属性



第三步：连接流程：鼠标点击—》出现流程连线—》连接节点，如下所示：



编辑完成后，点击右上角的保存图标即可保存此流程模型。如下所示：



保存成功后，可以对此流程进行部署、发起、导出等操作。

### 流程节点属性

* 开始事件

描述：指创建一个不包含触发器的开始事件

基础属性：

|  |  |
| --- | --- |
| **名称** | 该开始事件的名称 |
| **说明** | 对该开始时间的说明 |

* 用户任务

描述：指将任务手动分配给一个指定的人

基础属性：

|  |  |
| --- | --- |
| **说明** | 该用户任务节点对应service的service\_code |
| **表单模版** | 该用户任务节点对应的配置表单名称。可以是3种表单形式：如果是view，则对应id；  如果是html，则对应html名称；  如果是form表单，则对于其名称。 |
| **编号** | 自动生成后隐藏 |
| **名称** | 该用户任务节点的名称 |
| **分配** | 该用户任务处理人 |
| **优先级** | 该用户任务处理时的优先级 |

扩展属性：

|  |  |
| --- | --- |
| **循环类型** | （还没放开）暂时不支持 |

* 服务任务

描述：指业务逻辑自动任务，主要执行调用服务接口的功能

基础属性：

|  |  |
| --- | --- |
| **名称** | 该服务任务节点的名称 |
| **类字段** | 该服务任务节点调用service时的输入参数列表(对应Bex的参数) |
| **表达式** | 调用服务的具体方法 |
| **结果变量**  **名称** | 调用服务执行的返回结果 |

扩展属性：

|  |  |
| --- | --- |
| **循环类型** | 暂时没用 |

* 脚本任务

描述：指自动脚本逻辑任务。对调用接口的结果或数据进行处理

基础属性：

|  |  |
| --- | --- |
| **名称** | 该脚本任务节点的名称 |
| **说明** | 该脚本任务节点的说明 |
| **脚本格式** | 该脚本任务节点的脚本格式 |
| **脚本** | 该脚本任务节点的脚本内容 |

扩展属性：

|  |  |
| --- | --- |
| **循环类型** | 暂时没用 |

* 排他分支

描述：用来对流程中的决定进行建模。每一次只能选择一个流程执行

基础属性：

|  |  |
| --- | --- |
| **名称** | 该排他分支节点的名称 |
| **说明** | 该排他分支节点的说明 |

* 并行分支

描述：用来对流程中的并发进行建模。每一次必须选择所有并行的流程执行

基础属性：

|  |  |
| --- | --- |
| **名称** | 该并行分支节点的名称 |
| **说明** | 该并行分支节点的说明 |

* 包容分支

描述：排他分支和并行分支的结合，可以选择一个流程或多个流程执行

基础属性：

|  |  |
| --- | --- |
| **名称** | 该包容分支节点的名称 |
| **说明** | 该包容分支节点的说明 |

* 子流程

基础属性：

|  |  |
| --- | --- |
| **编号** | 该子流程节点的编号，自动生成并隐藏 |
| **名称** | 该子流程节点的名称 |
| **说明** | 该子流程节点的说明 |
| **异步** | 使用默认值 |
| **独家** | 使用默认值 |

扩展属性：

|  |  |
| --- | --- |
| **循环类型** | 暂时没用 |

* 结束事件

描述：没有一个具体的触发结束事件，即自动结束

基础属性：

|  |  |
| --- | --- |
| **名称** | 该结束事件节点的名称 |
| **说明** | 该结束事件节点的说明 |

* 顺序流

描述：顺序流定义活动的执行顺序

基础属性：

|  |  |
| --- | --- |
| **说明** | 该顺序流的名称 |
| **流条件** | 判断是否执行该顺序流的条件 |
| **默认流** | 标准说明只用满足条件才执行，默认流指如果其他分支条件不满足，则默认执行该顺序流 |
| **条件流** | 可以默认为标准 |

* 联系

描述：联系文本注释和流程节点对象

基础属性：

|  |  |
| --- | --- |
| **名称** | 该联系节点的名称 |
| **说明** | 该联系节点的说明 |

* 文本注释

描述：说明文字标注元素

基础属性：

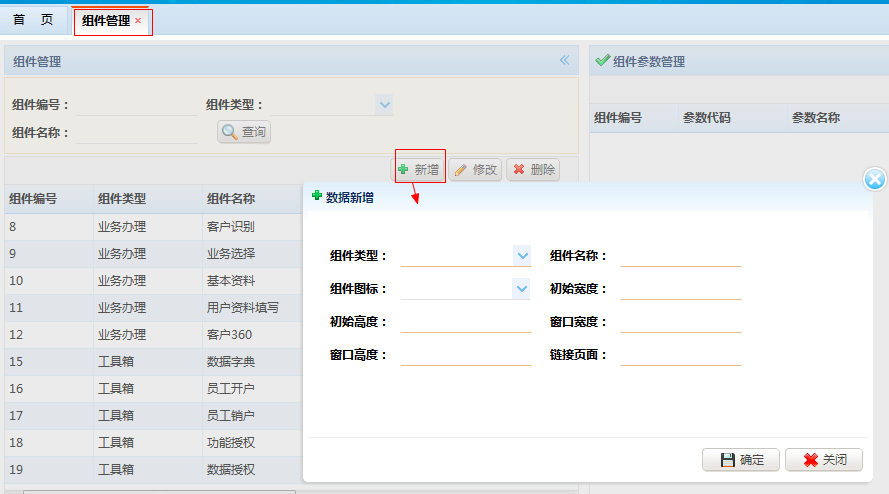
|  |  |
| --- | --- |
| **名称** | 该文本注释节点的名称内容 |
| **说明** | 该文本注释节点的说明 |

# 门户设计

## 门户对象说明

### 组件

组件是用于新建单独的业务功能模块，可存放在容器里面，也可直接放在布局中。一个组件对应一个业务功能模块。可以在管理平台—》门户管理—》组件管理中，新增组件。



### 容器

容器是用于存放组件。其中包括面板容器和TAB容器。面板容器和TAB容器的区别是TAB容器下面可新建多个页标签，将组件拖放在不同的页标签内，而面板容器没有。一个容器可存放多个组件

### 布局

布局是指界面的展示格局，用于设置选定菜单的布局格式，包括以下几种：

* 自由
* 1：3
* 1：3：1
* 1：2：2
* 1：2：1
* 1：4
* 2：2
* 3：2
* 1：1：2
* 1：1：3

后期需要，会适当增加。

## 门户界面设计

### 门户界面说明

门户界面提供给业务系统工作人员使用，界面如下：



左边栏是菜单列表，右边是选择的菜单的展示。

左下角还有系统设置功能。

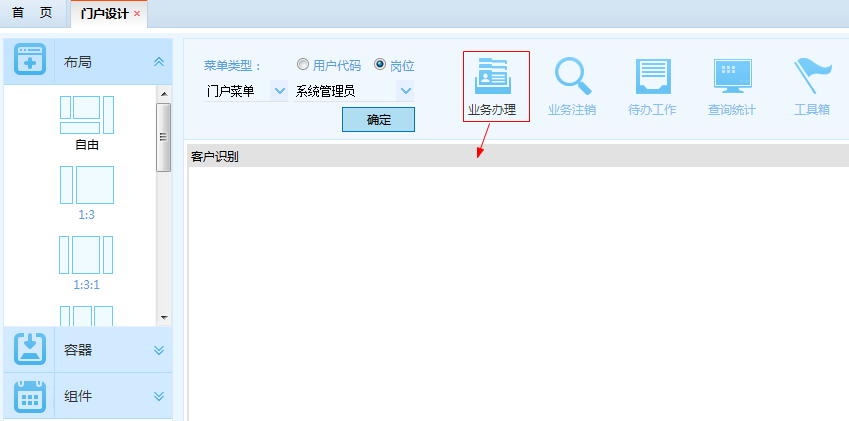
右下角提供搜索功能。

### 门户设计使用

在门户界面的菜单都属于门户菜单。

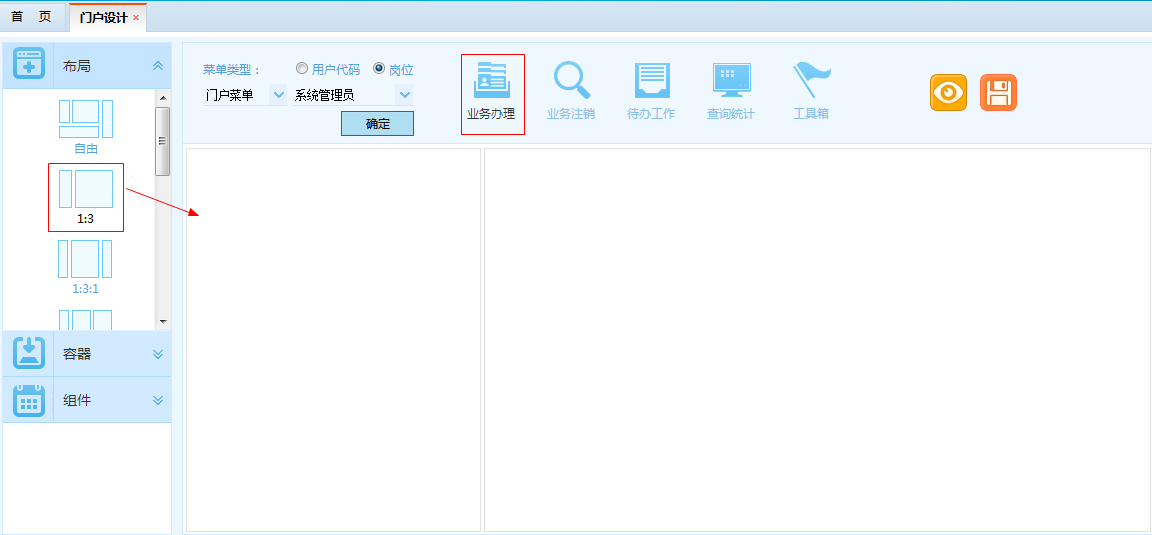
第一步：选择菜单

进入管理平台—》门户管理—》门户设计，选择某个菜单，例如选择业务办理：



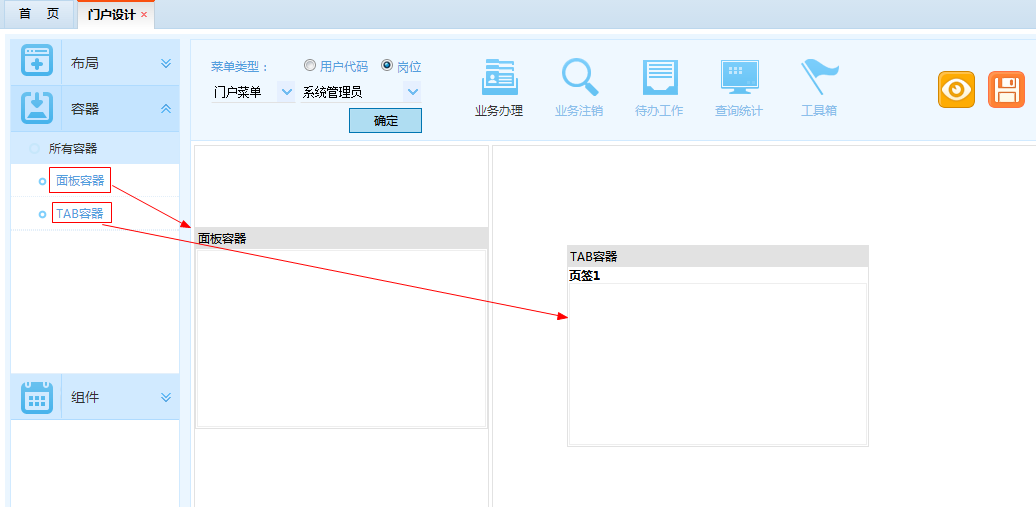
第二步：设置菜单的布局

选择新的布局，如1：3的布局比例，更换新的布局后要重新设置组件：



第三步：添加容器

从左边的容器中拖动面板容器或TAB容器到右边的布局中，如下所示：

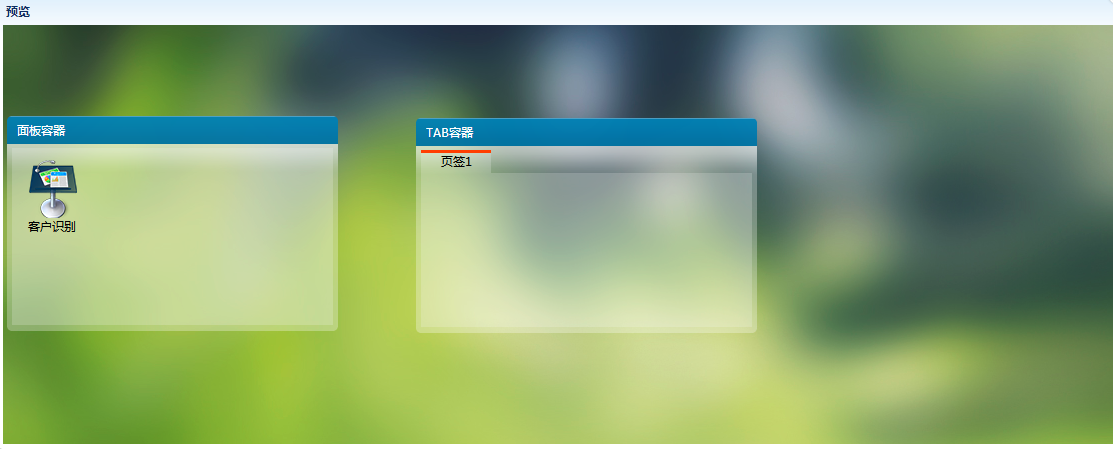


第四步：添加容器

从左边的组件列表中选择组件拖动到右边的容器中，也可以选择不放在容器中：



第五步：预览或保存。保存成功后即可在门户平台中看到对应菜单．

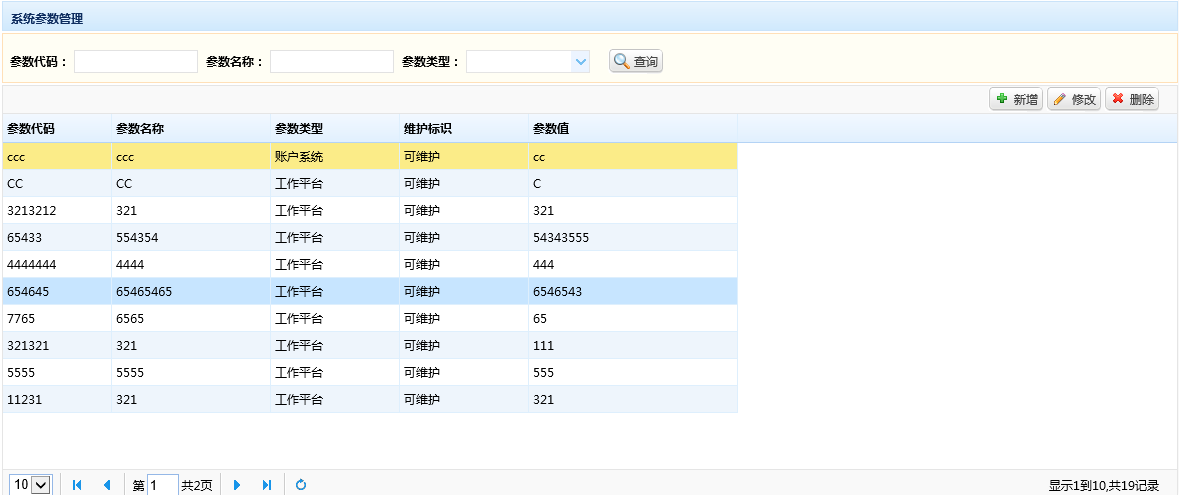


# 业务系统通用模块

## 参数管理

### 系统参数管理

系统参数管理模块用于系统参数的增删改查管理，页面预览图如下：



系统参数管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPM\_sysParamManage。界面上的功能有新增系统参数信息、修改系统参数信息、删除系统参数信息和查询系统参数信息。

* 新增系统参数信息

点击新增按钮，用户在弹出表单中完成系统参数信息的填写，调用配置好的service: sysParamInsert保存填写的数据，并弹出添加成功或者添加失败提示信息。

* 修改系统参数信息

选中一条记录后，根据维护标识判断该记录是否可以修改，如果可以修改则点击修改按钮，用户在弹出表单中完成系统参数信息的修改，调用配置好的service:sysParamUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息。

* 删除系统参数信息

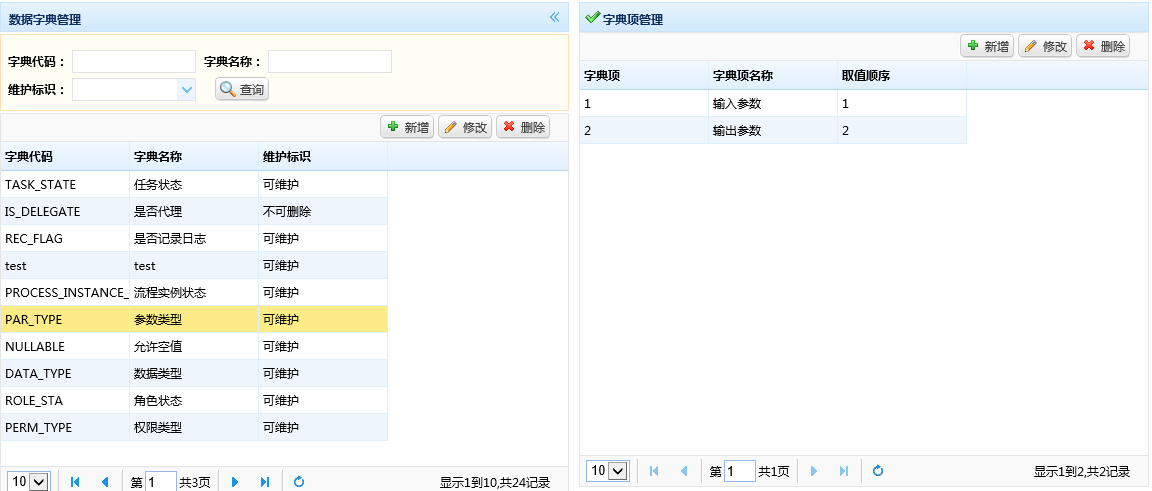
选中一条记录后，根据维护标识判断该记录是否可以删除，如果可以删除则点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service:sysParamDelete删除选中的记录数据，并弹出删除成功或者删除失败提示信息。

* 查询系统参数信息

在页面填写好相应的查询条件后，点击查询按钮，调用配置好的service:sysParamQuery查询显示出符合条件的记录数据。

### 数据字典管理

数据字典管理模块用于数据字典及字典项的增删改查管理，页面预览图如下：



数据字典管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPM\_dictionaryManage, UPM\_dicItemsManage。界面上的功能有新增数据字典信息、新增字典项信息、修改数据字典信息、修改字典项信息、删除数据字典信息、删除字典项信息和查询数据字典信息。

* 新增数据字典信息

点击新增按钮，用户在弹出表单中完成数据字典信息的填写，调用配置好的service: dictionaryInsert保存填写的数据，并弹出添加成功或者添加失败提示信息。

* 修改数据字典信息

选中一条记录后，根据维护标识判断该记录是否可以修改，如果可以修改则点击修改按钮，用户在弹出表单中完成数据字典信息的修改，调用配置好的service:dictionaryUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息，否则不能对此记录进行修改。

* 删除数据字典信息

选中一条记录后，根据维护标识判断该记录是否可以删除，如果可以删除则点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service:dictionaryDelete删除选中的记录数据，并弹出删除成功或者删除失败提示信息，否则不能对此记录进行删除。

* 查询数据字典信息

在页面填写好相应的查询条件后，点击查询按钮，调用配置好的service:dictionaryQuery查询显示出符合条件的记录数据。

* 新增字典项信息

选中左边的一条数据字典记录，点击右边字典项管理的新增按钮，用户在弹出表单中完成字典项信息的填写，调用配置好的service:dicItemsInsert保存填写的数据，并弹出添加成功或者添加失败提示信息。

* 修改字典项信息

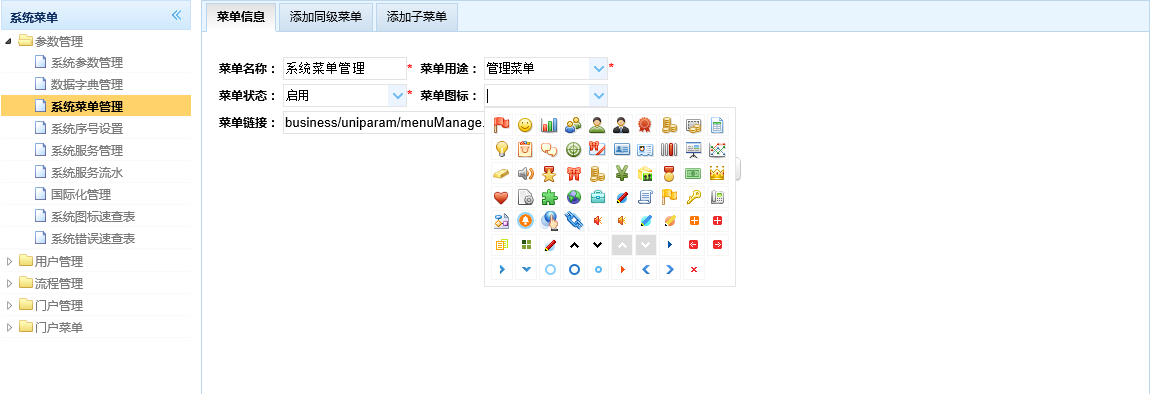
选中左边的一条数据字典记录，再选中字典项的一条记录，点击右边字典项管理的修改按钮，用户在弹出表单中完成字典项信息的修改，调用配置好的service:dicItemsUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息。

* 删除字典项信息

选中左边的一条数据字典记录后，再选中字典项一条记录，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service:dicItemsDelete删除选中的记录数据，并弹出删除成功或者删除失败提示信息。

### 系统菜单管理

系统菜单管理模块用于系统菜单增删改管理，页面预览图如下：



系统菜单管理模块界面上的功能有修改菜单信息、添加同级菜单信息、添加子菜单信息、删除菜单信息。

* 修改菜单信息

在左边的菜单树中选中需要修改的菜单节点，点击右边的菜单信息tab，在进行信息的修改后点击修改按钮保存修改后的数据信息，并弹出修改成功或者修改失败提示信息。

* 添加同级菜单信息

在左边的菜单树中选择跟新增菜单同级的菜单节点，点击右边的添加同级菜单tab，在填写好新增菜单的信息后点击确定按钮，保存填写好的菜单信息，并弹出添加成功或者添加失败提示信息。

* 添加子菜单信息

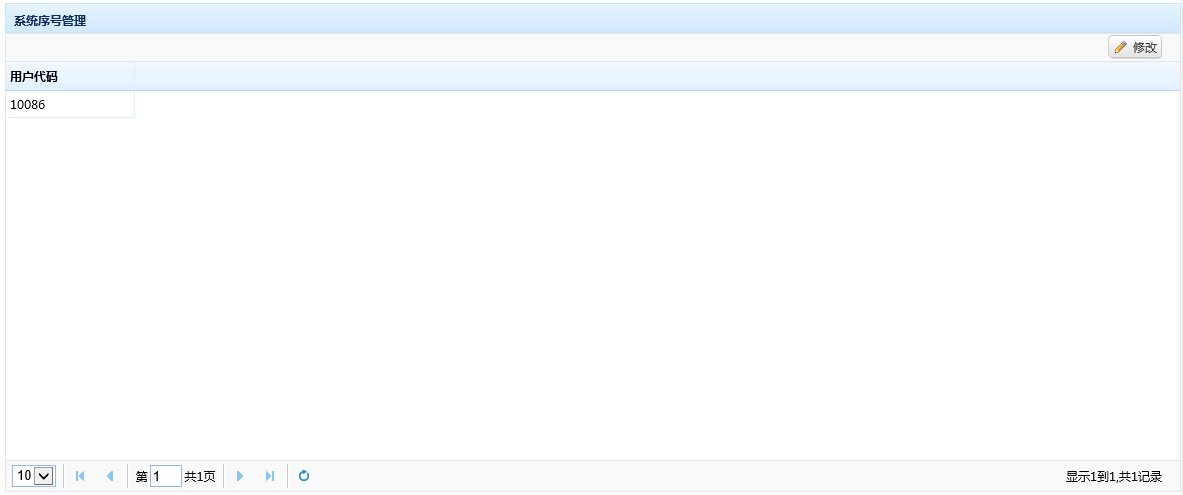
在左边的菜单树中选择新增菜单的父级的菜单节点点，点击右边的添加子菜单tab，在填写好新增菜单的信息后点击确定按钮，保存填写好的子菜单信息，并弹出添加成功或者添加失败提示信息。

* 删除菜单信息

在左边的菜单树中选中需要删除的菜单节点，点击右边的菜单信息tab，点击删除按钮删除对应的菜单信息，并弹出删除成功或者删除失败提示信息。

### 系统序号设置

系统序号设置模块用于系统序号的查看和修改功能，页面预览图如下：



系统序号设置模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPM\_sysSeqManage。界面上的功能有修改系统序号信息功能。

* 修改系统序号信息

在页面选中其中的一条记录后点击修改按钮按钮，用户在弹出表单中完成系统序号信息的修改，调用配置好的service:sysSeqUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息。

### 国际化管理

国际化管理模块用于国际化和国际化语言的增删改管理，页面预览图如下：



国际化管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID= UPM\_sysLocaleManage,UPM\_sysLocLangManage。界面上的功能有新增国际化信息、修改国际化信息、删除国际化信息和查询国际化信息，新增国际化语言信息、修改国际化语言信息、删除国际化语言信息。

* 新增国际化信息

点击新增按钮，用户在弹出表单中完成国际化信息的填写，调用配置好的service:localeInsert保存填写的数据，并弹出添加成功或者添加失败提示信息。

* 修改国际化信息

选中一条记录后，点击修改按钮，用户在弹出表单中完成国际化信息的修改，调用配置好的service:localeUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息。

* 删除国际化信息

选中一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service:localeDelete删除选中的记录数据，并弹出删除成功或者删除失败提示信息。

* 查询国际化信息

在页面填写好相应的查询条件后，点击查询按钮，调用配置好的service:localeQuery查询显示出符合条件的记录数据。

* 新增国际化语言信息

选择左边一条国际化记录后，在右边点击新增按钮，用户在弹出表单中完成国际化语言信息的填写，调用配置好的service:sysLocLangInsert保存填写的数据，并弹出添加成功或者添加失败提示信息。

* 修改国际化语言信息

选择左边一条国际化记录后，在右边选中需要修改的一条记录后点击修改按钮，用户在弹出表单中完成国际化语言信息的修改，调用配置好的service:sysLocLangUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息。

* 删除国际化语言信息

选择左边一条国际化记录后，在右边选中需要删除的一条记录后点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service:sysLocLangDelete删除选中的记录数据，并弹出删除成功或者删除失败提示信息。

### 系统服务流水管理

系统服务流水模块用于系统服务流水的查询功能和查看详细系统服务流水信息，页面预览图如下：



系统服务流水管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPM\_sysSerRecQuery。界面上的功能有查看系统服务流水的详细信息和查询系统服务流水信息。

* 查看系统服务流水的详细信息

选中一条记录后，点击查看详情按钮，调用upm\_sysSerRecDetail方法显示出该记录的详细信息。

* 查询系统服务流水信息

在页面填写好相应的查询条件后，点击查询按钮，调用配置好的service:sysSerRecQuery查询显示出符合条件的记录数据。

### 系统服务管理

系统服务管理模块用于系统业务的增删改及服务参数的管理，页面预览图如下：



系统业务模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPM\_sysServiceManage。界面上的功能有新增系统服务管理信息、修改系统服务管理信息、删除系统服务管理信息、查询系统服务管理信息及服务参数的管理。

* 新增系统服务管理信息

点击新增按钮，用户在弹出表单中完成系统服务管理信息的填写，调用配置好的service:sysServiceInsert保存填写的数据，并弹出添加成功或者添加失败提示信息。

* 修改系统服务管理信息

选中一条记录后，点击修改按钮，用户在弹出表单中完成系统服务管理信息的修改，调用配置好的service:sysServiceUpdate保存修改的数据，并弹出修改成功或者修改失败提示信息。

* 删除系统服务管理信息

选中一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service:sysServiceDelete删除选中的记录数据，并弹出删除成功或者删除失败提示信息。

* 查询系统服务管理信息

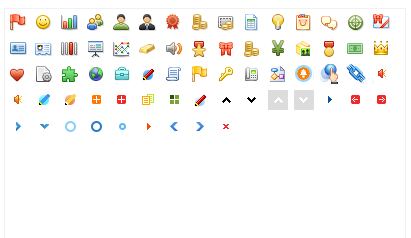
在页面填写好相应的查询条件后，点击查询按钮，调用配置好的service:sysServiceQuery查询显示出符合条件的记录数据。

* 服务参数管理信息

选中一条记录后，点击服务参数按钮，用户在弹出提示信息的对话框可以对该系统服务信息进行参数的设置，包括参数的新增、修改、删除操作。

### 系统图标速查表

系统图标速查表模块用于系统图标查看功能，页面预览图如下：



系统图标速查表是为了方便查看出系统图标对应的名字而设置的。

### 系统错误速查表

系统错误速查表模块用于系统错误的查看系统错误的详细信息及查询系统错误信息功能，页面预览图如下：



系统错误速查表模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPM\_sysErrorQuery。界面上的功能有查看系统错误信息详情和查询系统错误信息功能。

* 查看错误信息详情

在页面选中其中的一条记录后点击查看详情按钮按钮，调用upm\_sysErrorDetail方法显示出该记录的详细信息。

* 查询系统错误信息

在页面填写好相应的查询条件后，点击查询按钮，调用配置好的service:sysErrorQuery查询显示出符合条件的记录数据。

## 操作员管理

### 操作员信息管理

操作员信息管理模块用于操作员信息的查询、新增操作员、修改操作员信息、重置密码、变更操作员状态、删除操作员等，页面预览图如下：



操作员信息管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UUM\_userInfoManage。界面上的功能有新增操作员信息、修改操作员信息、删除操作员信息和查询操作员信息。

* 新增操作员信息

点击新增按钮，操作员在弹出表单中完成操作员信息的填写，调用配置好的service="userInfoInsert"保存填写的数据，并弹出添加成功提示信息。

* 修改操作员信息

在选中一条记录后，点击修改按钮，操作员在弹出表单中完成操作员信息的修改，调用配置好的service="userInfoUpdate"保存修改的数据，并弹出修改成功提示信息。

* 删除操作员信息

在选中一条记录后，点击删除按钮，操作员在弹出提示信息的对话框选择确定删除后，调用配置好的service="userInfoDelete"删除选中的记录数据，并弹出删除成功提示信息。

* 查询操作员信息

在页面填写好相应的查询调后后，点击查询按钮，调用配置好的service="userInfoQuery"查询显示出符合条件的记录数据。

* 重置密码

点击重置密码按钮，调用配置好的service="userInfoUpdate"，函数userPasswordReset，可对操作员密码进行重置

* 状态变更

点击重置密码按钮，调用配置好的service="userInfoUpdate"，函数userStatusModify，可对操作员状态进行修改

### 岗位管理

岗位管理模块用于岗位信息的查询，增删改管理，页面预览图如下：



岗位管理信息模块的界面是通过html生成的，调用的url为f business/userManage/roleManage.html。界面上的功能有新增岗位信息、修改岗位信息、删除岗位信息和查询岗位信息。

* 新增岗位信息

点击新增按钮，操作员在弹出表单中完成岗位信息的填写，调用配置好的service="roleInsert"保存填写的数据，并弹出添加成功提示信息。

* 修改岗位信息

在选中一条记录后，点击修改按钮，操作员在弹出表单中完成岗位信息的修改，调用配置好的service="roleUpdate"保存修改的数据，并弹出修改成功提示信息。

* 删除岗位信息

在选中一条记录后，点击删除按钮，操作员在弹出提示信息的对话框选择确定删除后，调用配置好的service="roleDelete"删除选中的记录数据，并弹出删除成功提示信息。

* 查询岗位信息

在页面填写好相应的查询调后后，点击查询按钮，调用配置好的service="roleQuery"查询显示出符合条件的记录数据。

### 组织机构管理

组织机构管理模块是对组织机构的管理和维护。

左边展示树状的组织机构信息；

右边展示为tabs标签，有机构信息维护(包括修改和删除)、新增同级机构、新增下级机构

* 机构信息

选中一个机构，显示该机构信息，可以进行修改或者删除该机构。

* 新增同级机构

选中一个机构，新增一个与该机构同级的机构

* 新增下级机构

选中一个机构，新增该机构下的营业部

### 用户授权管理

用户授权管理模块用于授予用户某个菜单的权限，页面预览图如下：



用户授权模块的界面是通过html生成的，调用的url为business/userManage/userAuthoManage.html。该模块具有的功能有查询用户权限、查询用户信息、设置用户权限。

* 查询用户权限和信息

选择某个用户，可以查看该用户以后的权限和信息

* 设置用户权限

在权限设置下面勾选权限，点击提交按钮，可以进行提交，不选择则将该用户的全部权限取消。

## 权限管理

### 菜单授权

菜单授权主要是给操作员、岗位、分子公司进行菜单的授权，授权方法灵活，有以下多种方式：

a、可以选定具体的菜单给具体的操作员，岗位，分子公司；

b、可以选定具体的操作员，或者岗位，或者分子公司，分配某些菜单

d、a/b可以进行批量设置

### 临时授权

临时授权菜单主要是用于专门用于离岗、备岗的管理，主要功能有设置权限有效期(即设置起始时间和终止时间)、临时授权某些菜单给指定操作员、临时授权A操作员菜单给B操作员。

左边显示所有权限，右边显示操作员信息。点击左边某个权限，右边即显示该权限下面的操作员。

* 设置权限有效期(即设置起始时间和终止时间)

选择某个权限，点击“设置有效期”，在弹出框里设置某个权限的有效期。

* 临时授权某些菜单给指定操作员

选择某个操作员，点击“临时授权”，在弹出框里选择菜单，点击确定，即可授予该操作员临时菜单权限

* 临时授权A操作员菜单给B操作员

选择A操作员，点击“临时授予其他操作员”，在弹出框里选择B操作员，点击确定，即可将A操作员权限授给B

### 岗位分配

岗位分配管理模块主要是为操作员分配岗位。主要功能有岗位信息的查询、操作员分配岗位、操作员删除岗位。

* 岗位信息的查询

输入岗位名称或岗位状态，点击查询按钮，即可查出某个岗位下的操作员信息

* 操作员分配岗位
* 新增操作员

选择一个岗位，点击岗位操作员管理，弹出该岗位下面拥有的操作员信息：



点击“新增”按钮，可以新增不在该岗位下的操作员，界面如下：



* 删除操作员

选择一个或多个操作员，点击“删除按钮”，即可删除该岗位下的操作员

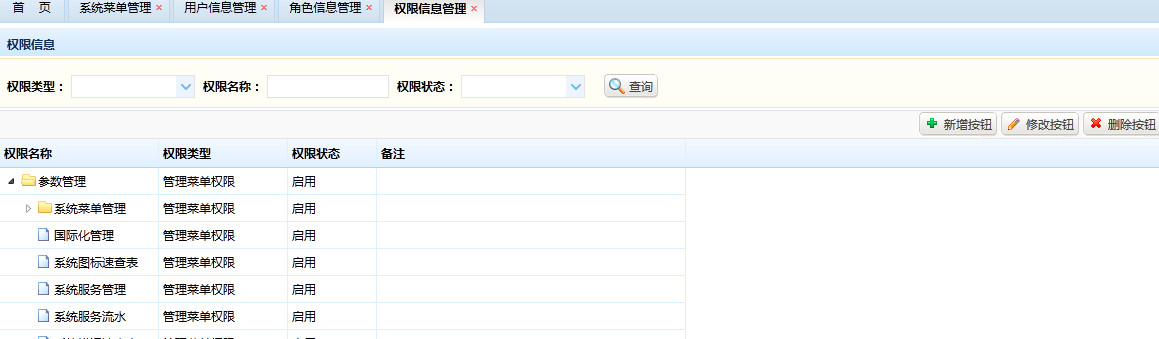


* 岗位授权权限设置

选中一个岗位，在弹出框里显示菜单权限，先勾选上该岗位已有的权限，也可以重新设置该岗位的新的权限。

### 权限信息管理

权限信息管理模块用于权限信息进行管理，页面预览图如下：



权限信息模块的界面是通过html生成的，调用的url为business/userManage/permInfoManage.html。界面上的功能有新增按钮、修改按钮、删除按钮信息和查询权限信息。其中，一级权限菜单和二级权限菜单不可删除或修改，按钮只能在第二级的权限菜单下进行新增。

* 新增按钮

选择二级权限菜单，点击新增按钮，用户在弹出表单中完成按钮信息的填写，通过调用函数addButtonInfo进行新增按钮信息。

* 修改按钮信息

在选中一条按钮信息后，点击修改按钮，用户在弹出表单中完成按钮信息的修改，调用函数updateButtonInfo保存修改的数据，并弹出修改成功提示信息。

* 删除按钮信息

在选中一条按钮信息后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用函数delPermInfo删除选中的记录数据，并弹出删除成功提示信息。

* 查询权限信息

在页面填写好相应的查询调后后，点击查询按钮，调用配置好的service="permissionQuery"查询显示出符合条件的记录数据。

## 流程管理

### 流程发布管理

流程发布管理模块用于流程的部署、查看流程列表、查看流程图和流程的删除管理，页面预览图如下：



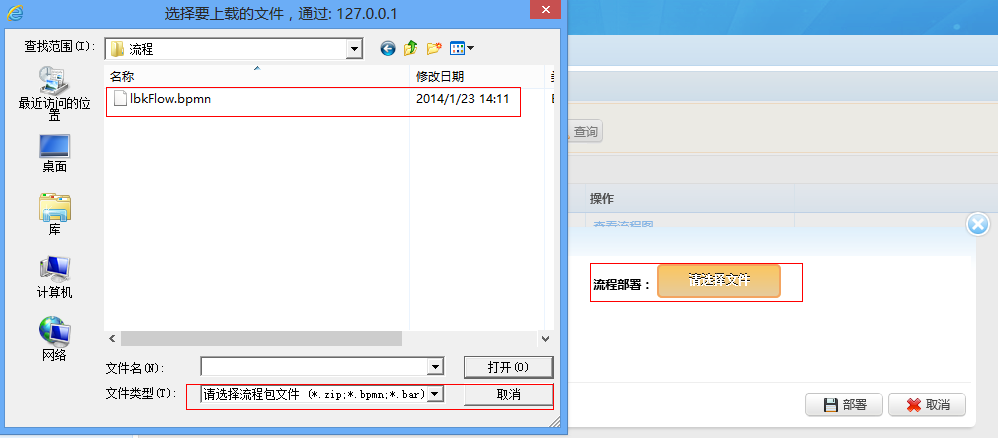
**图6-1-17 流程发布管理列表页面**

流程发布管理模块的界面上的功能有流程部署、查询流程发布记录、查看流程图和删除流程。

* 流程部署

点击流程部署按钮，用户在弹出的页面中选择要上传的流程包，此处限制只能上传一个文件，当选择了一个文件上传后，按钮显示不可操作状态，可以选择删除掉已上传的文件再进行重新上传，上传的格式此处限制为xml、rar、zip格式文件，如果强制选择其他文件格式上传会提示错误，如果没有选择上传的文件直接点击部署按钮会给出提示，点击部署按钮部署流程并弹出部署成功或者部署失败的提示信息。

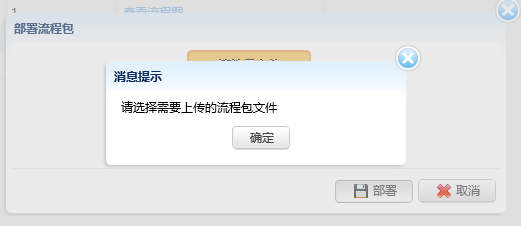
页面效果图如下：



**图6-1-18 流程部署页面1**



**图6-1-19 流程部署页面2**



**图6-1-20 流程部署页面3**

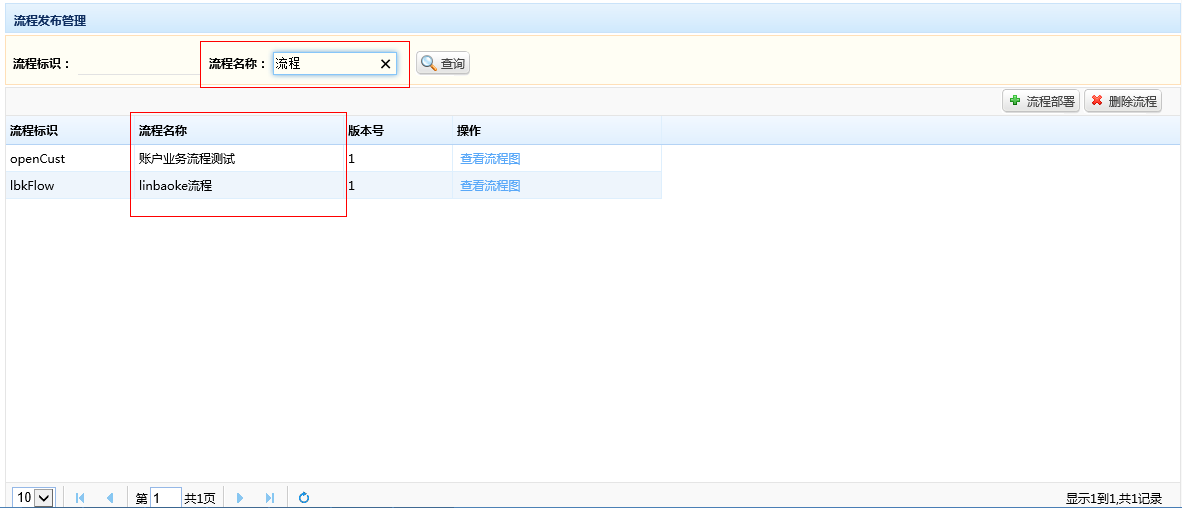


**图6-1-21 流程部署页面4**

* 查询流程发布记录

填写相应的查询条件后点击查询按钮，便可列出对应符合条件的记录，支持模糊查询。

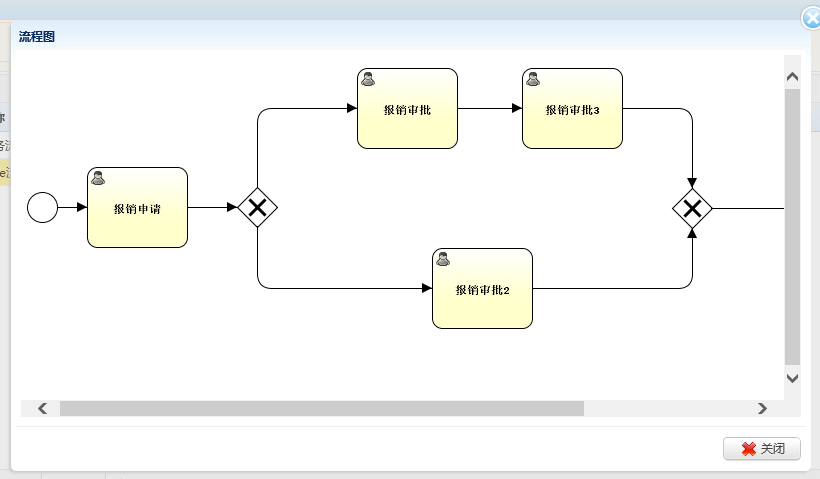
页面效果图如下：



**图6-1-22 查询流程发布信息页面**

* 查看流程图

点击每条记录的查看流程图链接，用户可以查看到该流程的详细流程图。

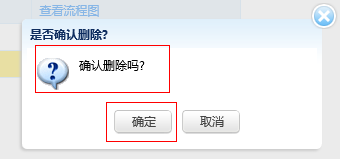
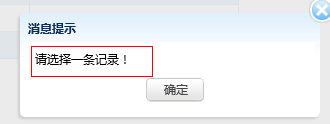


**图6-1-23 查看流程发布流程图页面**

* 删除流程

选中一条记录后，点击删除按钮，并弹出是否确认删除信息，选择确定后会提示删除成功或者删除失败提示信息，如果没有选择一条记录直接点击删除按钮，会出现提示信息。

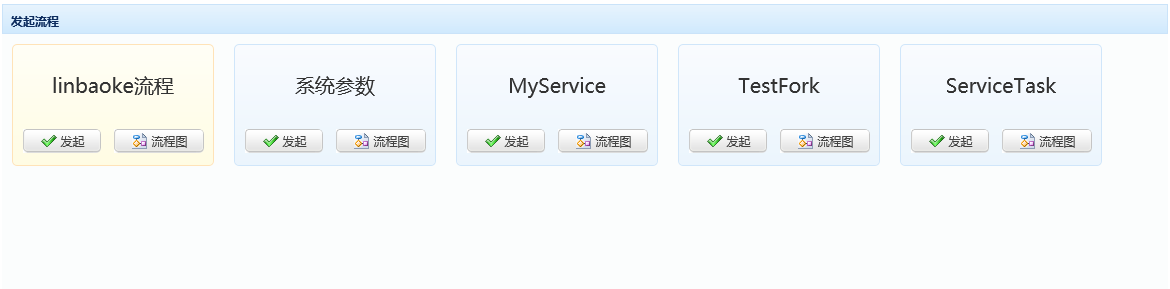
页面效果图如下：



**图6-1-24 删除流程发布信息页面**

### 发起流程管理

发起流程模块用于流程的发起和查看对应的流程图管理，页面预览图如下：



**图6-1-25 发起流程页面**

发起流程模块的界面上的功能有发起流程和查看流程图。

* 发起流程

点击发起按钮，如果该节点需要填写表单则用户在弹出表单中完成表单的填写，点击提交按钮并弹出发起成功或者发起失败提示信息，否则直接弹出提示信息。

页面效果图如下：



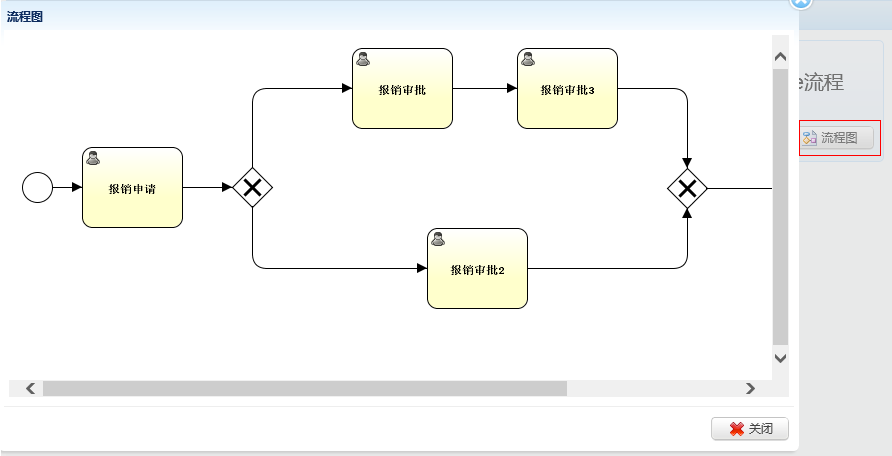
**图6-1-26 发起流程填写表单页面**



**图6-1-27 发起流程发起成功页面**

* 查看流程图

点击对应流程的流程图按钮可以查看对应流程的流程图。



**图6-1-28 查看发起流程流程图页面**

### 待办任务管理

待办任务模块用于待办任务信息的查询、认领、处理、代理、回退和查看监控流程图管理，页面预览图如下：



**图6-1-29 待办任务列表页面**

待办任务模块的界面上的功能有查询待办任务信息、认领待办任务、处理待办任务、回退待办任务和查看待办任务的监控流程图。

* 查询待办任务信息

填写需要查询的条件后点击查询按钮，便可查询出对应符合条件的待办任务记录。

页面效果图如下：



**图6-1-30 查询待办任务信息列表页面**

* 认领待办任务信息

选中一条未认领的待办任务记录后，认领按钮显示为可操作状态，处理按钮、代理按钮、和回退按钮显示为不可操作状态，点击认领按钮，当前用户即可认领该待办任务，并弹出认领成功或者认领失败的提示信息。

页面效果图如下：



**图6-1-31 认领待办任务页面**

* 处理待办任务信息

选中一条已认领的待办任务记录后，处理按钮、认领按钮和回退按钮显示为可操作状态，认领按钮显示为不可操作状态，点击处理按钮，即可在弹出的对话框中填写相应表单，点击提交后便完成当前任务的处理工作，关闭对话框并弹出提示信息。

页面效果图如下：



**图6-1-32 处理待办任务页面**

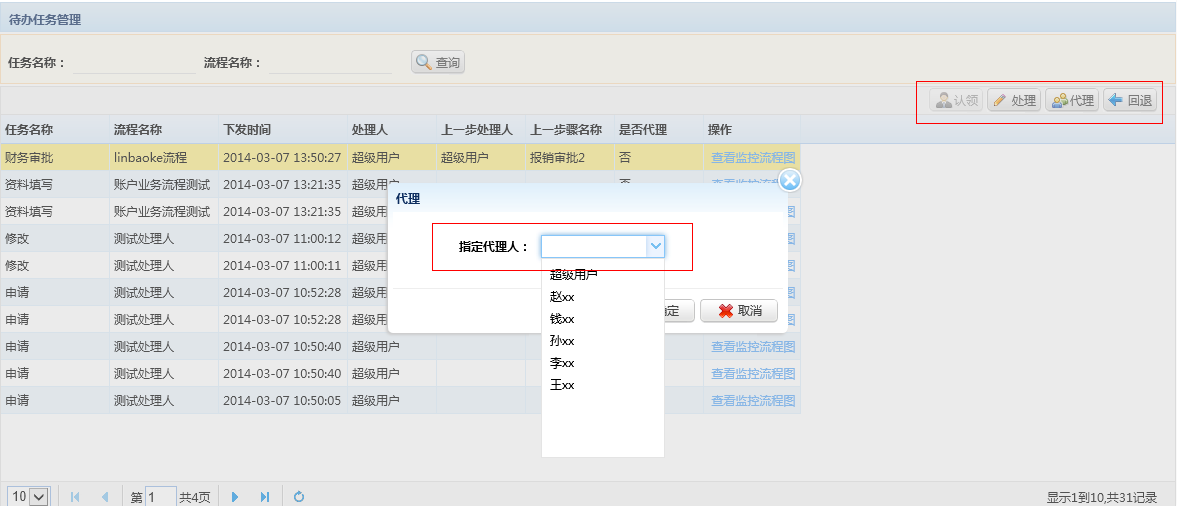


**图6-1-33 处理待办任务成功页面**

* 代理待办任务信息

选中一条已认领的待办任务记录后，处理按钮、认领按钮和回退按钮显示为可操作状态，认领按钮显示为不可操作状态，点击代理按钮，即可在弹出的对话框中选择接收代理的人，点击确定后便完成当前任务的代理工作，等待代理人去处理，关闭对话框并弹出提示信息。

页面效果图如下：

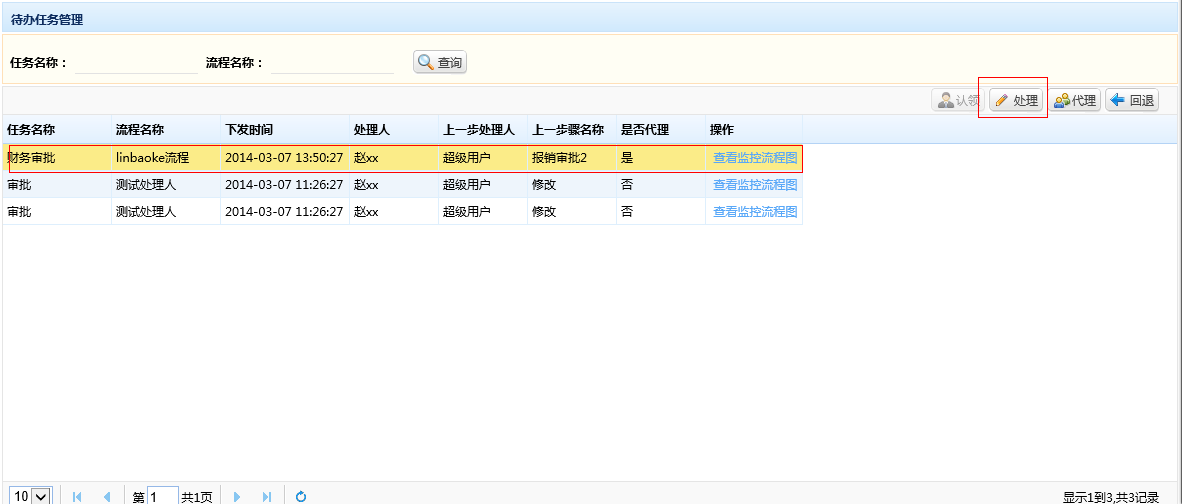


**图6-1-34 代理待办任务页面**



**图6-1-35 代理待办任务成功页面**

接收代理人登录后查看待办任务列表，并处理需要处理的任务，如下图：



**图6-1-36 处理代理的待办任务页面**

* 回退待办任务信息

选中一条已认领的待办任务记录后，点击回退按钮，选中的待办任务将会进行回退操作，并弹出提示信息。

页面效果图如下：

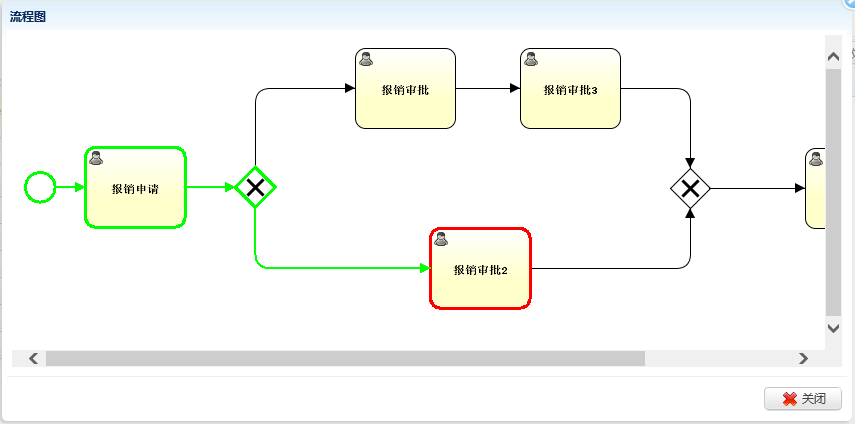


**图6-1-37 回退待办任务成功页面**

* 查看待办任务的监控流程图

点击每条记录的查看监控流程图链接，用户可以查看到该待办任务节点的监控流程图。

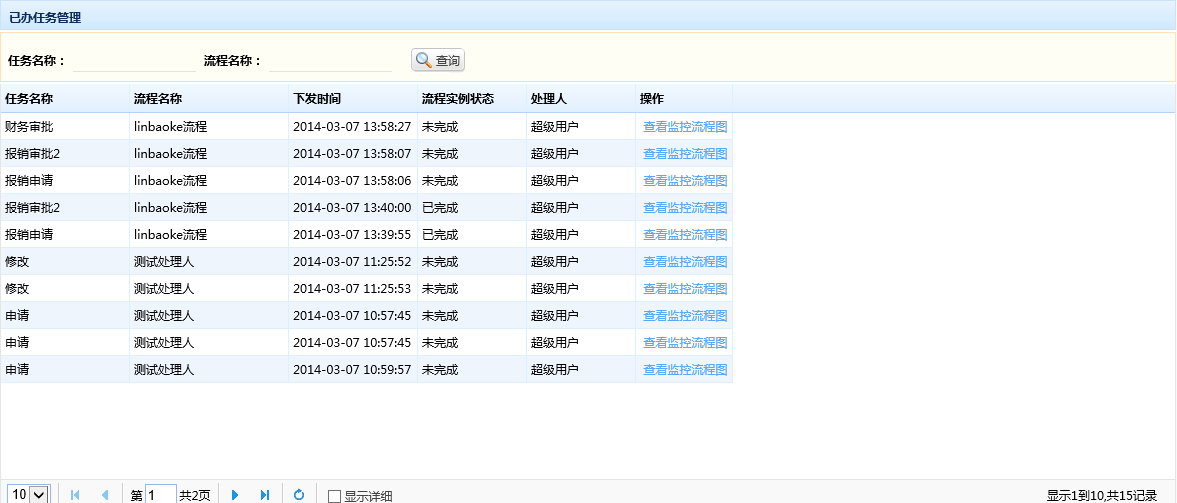
页面效果图如下：



**图6-1-38 查看待办任务监控流程图页面**

### 已办任务管理

已办任务模块用于已办任务信息的查询和查看已办任务的监控流程图信息管理，页面预览图如下：



**图6-1-39 查看已办任务列表页面**

已办任务模块的界面上的功能有查询已办任务信息、查看已办任务的监控流程图。

* 查询已办任务信息

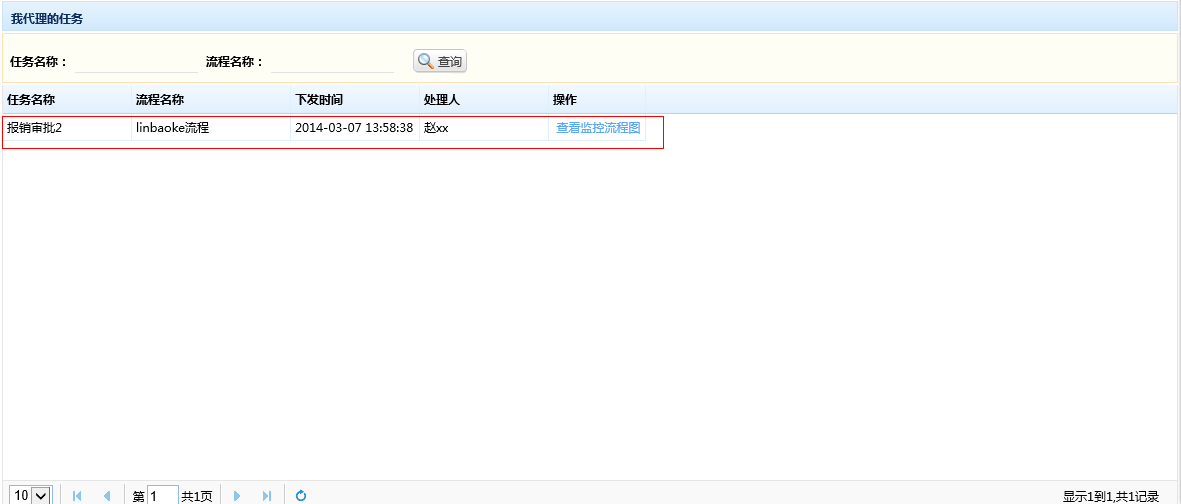
填写对应的查询条件后点击查询按钮，即可列出对应符合条件的已办任务信息。

* 查看已办任务的监控流程图

点击每条记录的查看监控流程图链接，用户可以查看到该已办任务节点的监控流程图。

### 代理任务管理

代理任务模块用于显示当前用户代理出去的任务的管理，页面预览图如下：

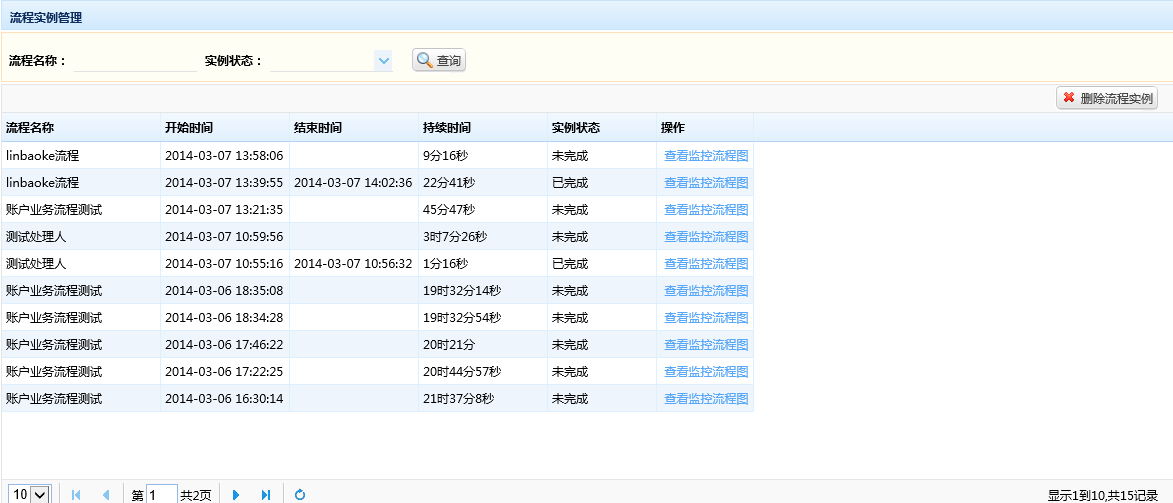


**图6-1-40 查看代理任务列表页面**

代理任务模块是为了显示当前用户已经代理出去的任务的信息，可以通过查询条件得到对应的记录，如果该代理任务已经被处理了则不显示，在已办任务列表或者待办列表可以查到。

### 流程实例管理

流程实例管理模块用于流程实例的查询、查看流程实例的监控流程图和删除流程实例管理，页面预览图如下：



**图6-1-41 流程实例管理列表页面**

流程实例管理模块的界面上的功能有查询流程实例信息、查看流程实例的监控流程图和删除流程实例。

* 查看流程实例信息

填写对应的查询条件后点击查询按钮，用户便可查询出符合条件的流程实例记录。

* 查看监控流程图

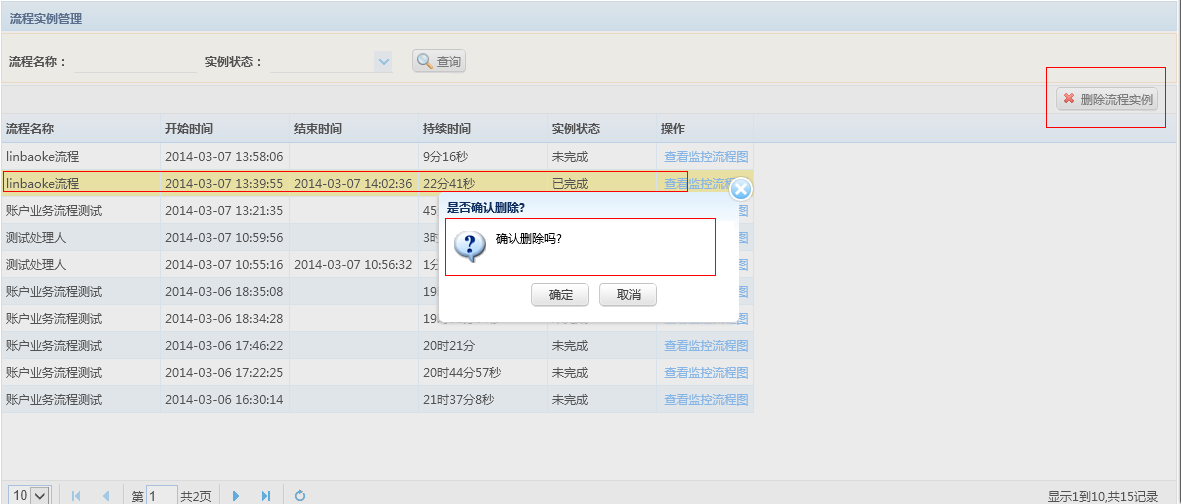
点击每条记录的查看监控流程图链接，用户可以查看到该流程实例的监控流程图。

* 删除流程实例

在选中一条记录后，点击删除按钮，并弹出确认框，选择确定后弹出删除成功或者删除失败提示信息，如果没有选择一条记录，直接点击删除流程实例按钮会弹出提示对话框。



**图6-1-42 删除流程实例列表页面1**



**图6-1-43 删除流程实例列表页面2**



**图6-1-44 删除流程实例列表页面4**

### 流程模型管理

流程模型管理模块用于流程模型的创建、流程部署、流程模型编辑、流程导入、流程导出、流程模型的查询和删除流程模型功能管理，页面预览图如下：

* 流程模型创建功能

进入页面后点击创建按钮会弹出一个对话框，填写模型的相关信息，其中模型标识和模型名称为必填项，点击保存按钮后会进入到一个模型设计器里，在模型设计器里可以进行流程的设计，页面效果图下：



**图6-1-45 创建流程模型页面1**

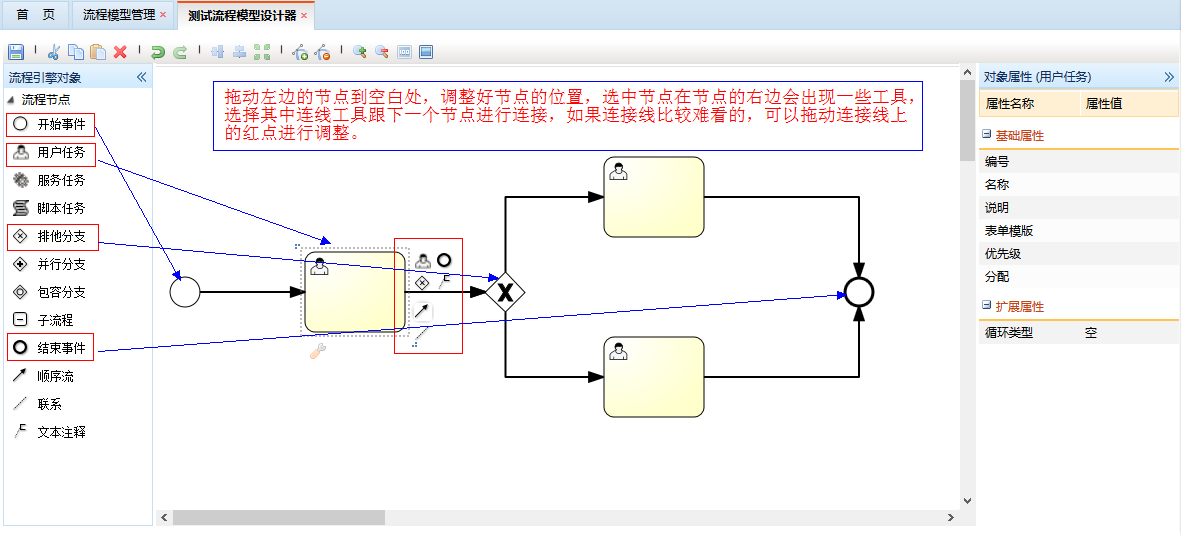


**图6-1-46 创建流程模型页面2**



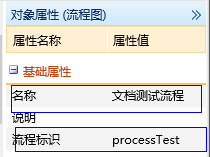
**图6-1-47 创建流程模型页面3**

进入模型设计器页面可以进行流程的设计：



**图6-1-48 流程模型设计器页面**

接下来对流程和流程个节点进行参数的配置，点击节点意外的空白处在右边会出现整个流程的属性设置，填写流程名称和流程标识，说明可以不填，流程标识默认为process，如下图所示。

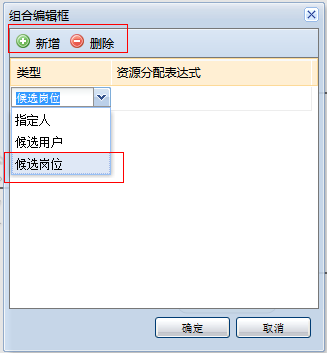


**图6-1-49 流程模型属性页面1**

对流程节点的属性进行配置：

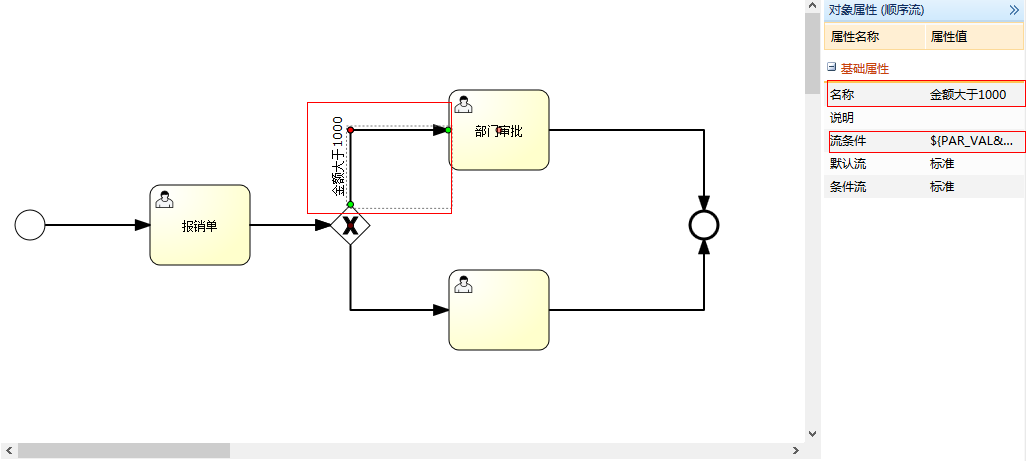
1. 开始节点：属性有名称和说明，填写名称属性，说明可以不填。
2. 用户任务节点：基础属性有编号、名称、说明、表单模板、优先级、分配。这种节点在我们流程中规定：
   * 1. 编号可以不填，流程会自动生成，名称为必填。
     2. 说明应该填配置操作表单的service的service\_code。
     3. 表单模板为配置表单的view ID。
     4. 优先级为数字数据格式。
     5. 分配为处理该任务节点的对象：分配的对象有三种类型，a 指定人：分配给指定人，表示该任务直接有该指定人处理，在待办任务中不需要认领操作，可以直接进行处理；b 候选用户：分配给候选用户，多个用户id用逗号隔开，表示该任务可以有候选用户去认领处理（此对象还没进行测试）；c 候选岗位：分配给候选岗位，填写岗位的id，表示在候选岗位下的用户都可以认领该任务进行处理。

配置属性举例效果图如下：

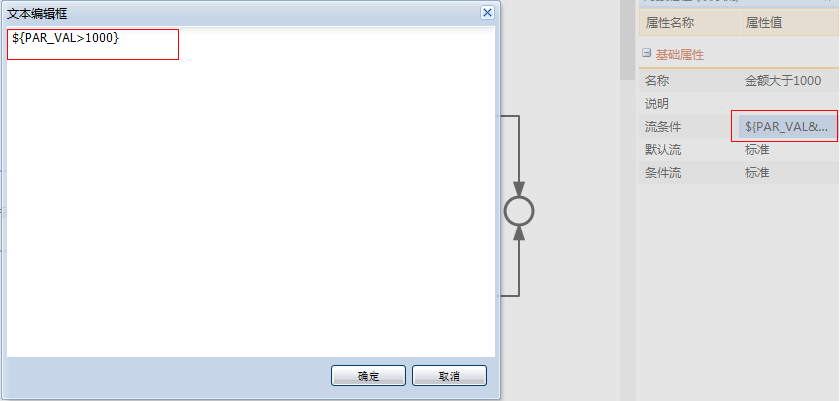
 

**图6-1-50 流程模型属性页面2**

* + 1. 排他分支节点：属性有名称和说明，都可以不填。
    2. 连接线：属性有名称、说明、流条件、默认流、条件流，没有做判断的连接线的属性可以不填；有分支的连接线的流条件需要填写；默认流和条件流选择标准，流条件要写你填写表单数据用来判断流程走向的字段名称，例子中用的是${PAR\_VAL>10000}，格式要求${}在大括号里填判断表达式，例子效果图如下：

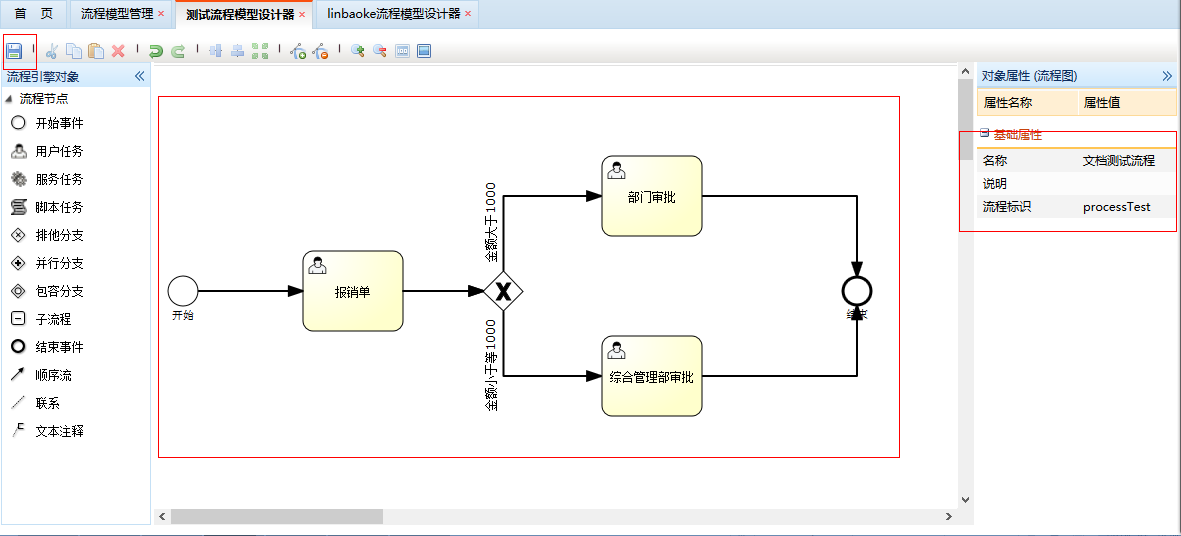


**图6-1-51 流程模型属性页面3**



**图6-1-52 流程模型属性页面4**

配置完所有需要的属性后，保存流程，这样就设计好一个流程了，效果图如下：



**图6-1-53 流程模型属性页面5**

* 流程模型部署功能

选择创建好的流程模型进行部署，部署成功后会弹出部署成功信息或者部署失败时会提示部署失败信息。

* 流程模型编辑功能

选择创建好的流程模型点击编辑按钮进行编辑，此时会跳到流程模型设计器的编辑页面，即可对流程模型进行编辑，编辑完成保存退出，如果没选中其中一条记录点击编辑则会弹出提示信息。

* 流程模型导入功能

点击导入按钮，会出现导入流程的对话框，选择好符合要求的流程后，即可导入，此处跟流程发布管理的部署功能类似。

* 流程模型导出功能

选择一条已创建好的流程模型记录后，点击导出按钮即可导出此流程模型，如何没有选择记录直接点击导出会弹出提示信息。

* 流程模型查询功能

填写对应的查询条件后点击查询按钮，即可列出对应符合条件的流程模型信息。

* 删除流程模型功能

选择一条需要删除的记录，点击删除按钮，在弹出的确认框选择确定后便可删除该记录，并弹出删除成功或者删除失败信息，如果没选任何记录点击删除按钮会弹出提示信息

## 门户设计

### 布局管理

布局管理模块用于查询布局信息、查看布局对应的板块信息、新增布局、修改布局、删除布局、新增布局对应的板块、修改布局对应的板块、删除布局对应的板块等，页面预览图如下：



布局管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPT\_layoutMrg,UPT\_boardMrg。

* 新增布局信息

点击新增按钮，用户在弹出表单中完成布局信息的填写，调用配置好的service="addLayout"保存填写的数据，并弹出添加成功提示信息。

* 修改数据字典信息

在选中一条记录后，点击修改按钮，用户在弹出表单中完成布局信息的修改，调用配置好的service="updateLayout"保存修改的数据，并弹出修改成功提示信息。

* 删除数据字典信息

在选中一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service="deleteLayout"删除选中的记录数据，并弹出删除成功提示信息。

* 查询布局信息

在页面填写好相应的查询调后后，点击查询按钮，调用配置好的service="getLayout"查询显示出符合条件的记录数据。

* 新增板块信息

在右边选中一条布局记录后点击右边板块管理的新增按钮，用户在弹出表单中完成板块项信息的填写，调用配置好的service="addBoard"保存填写的数据，并弹出添加成功提示信息。

* 修改板块信息

在右边选中一条板块记录后点击右边板块管理的修改按钮，用户在弹出表单中完成板块项信息的修改，调用配置好的service="updateBoard"保存修改的数据，并弹出修改成功提示信息。

* 删除板块信息

在右边选中一条数据字典记录后，选中板块一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service="deleteBoard"删除选中的记录数据，并弹出删除成功提示信息。

### 容器管理

容器管理模块用于查询容器信息、查看容器对应的容器页签信息、新增容器、修改容器、删除容器、新增容器应的容器页签、修改容器对应的容器页签、删除容器对应的容器页签等，页面预览图如下：



容器管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPT\_contMrg,UPT\_tabMrg。

* 新增容器信息

点击新增按钮，用户在弹出表单中完成容器信息的填写，调用配置好的service="addContainer"保存填写的数据，并弹出添加成功提示信息。

* 修改容器信息

在选中一条记录后，点击修改按钮，用户在弹出表单中完成容器信息的修改，调用配置好的service="updateContainer"保存修改的数据，并弹出修改成功提示信息。

* 删除容器信息

在选中一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service="deleteContainer"删除选中的记录数据，并弹出删除成功提示信息。

* 查询容器信息

在页面填写好相应的查询调后后，点击查询按钮，调用配置好的service="getContainer"查询显示出符合条件的记录数据。

* 新增容器页签信息

在右边选中一条容器记录后点击右边容器页签管理的新增按钮，用户在弹出表单中完成容器页签项信息的填写，调用配置好的service="addTab"保存填写的数据，并弹出添加成功提示信息。

* 修改容器页签信息

在右边选中一条板块记录后点击右边容器页签管理的修改按钮，用户在弹出表单中完成容器页签项信息的修改，调用配置好的service="updateTab"保存修改的数据，并弹出修改成功提示信息。

* 删除容器页签信息

在右边选中一条容器页签记录后，选中板块一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service="deleteTab"删除选中的记录数据，并弹出删除成功提示信息。

### 组件管理

组件管理模块用于查询组件信息、查看组件对应的组件参数信息、新增组件、修改组件、删除组件、新增组件应的组件参数、修改组件对应的组件参数、删除组件对应的组件参数等，页面预览图如下：



组件管理模块的界面是通过动态的VIEW配置自动生成的，调用的url为frame/page/genericParam.html?VIEW\_ID=UPT\_compMrg,UPT\_compParamMrg。

* 新增组件信息

点击新增按钮，用户在弹出表单中完成组件信息的填写，调用配置好的service="addComponent"保存填写的数据，并弹出添加成功提示信息。

* 修改组件器信息

在选中一条记录后，点击修改按钮，用户在弹出表单中完成组件信息的修改，调用配置好的service="updateComponent"保存修改的数据，并弹出修改成功提示信息。

* 删除组件信息

在选中一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service="deleteComponent"除选中的记录数据，并弹出删除成功提示信息。

* 查询组件信息

在页面填写好相应的查询调后后，点击查询按钮，调用配置好的service="getComponents"查询显示出符合条件的记录数据。

* 新增组件参数信息

在右边选中一条组件记录后点击右边组件参数管理的新增按钮，用户在弹出表单中完成组件参数项信息的填写，调用配置好的service="addCompParam保存填写的数据，并弹出添加成功提示信息。

* 修改组件参数信息

在右边选中一条板块记录后点击右边组件参数管理的修改按钮，用户在弹出表单中完成组件参数项信息的修改，调用配置好的service="updateCompParam"保存修改的数据，并弹出修改成功提示信息。

* 删除组件参数信息

在右边选中一条组件参数记录后，选中板块一条记录后，点击删除按钮，用户在弹出提示信息的对话框选择确定删除后，调用配置好的service="deleteCompParam"删除选中的记录数据，并弹出删除成功提示信息。

### 运行平台管理

运行平台管理功能用于设置用户或用户角色在运行平台菜单中的使用的布局和组件。



* 初始化

选择角色或输入用户编号，调用getFrontMenu获取运行平台的菜单配置加载菜单选择界面，调用getLayout获取布局配置加载布局选择界面，调用getComonent和getContainer获取组件和容器的配置加载容器组件选择的界面。

* 设置容器或组件

选择某一个菜单时，调用getMenuCompCfgs和getMenuCompParam分别获取菜单组件配置和组件参数配置，如果已经设置了菜单中的布局和组件则调用解析器把界面解析并展示在主工作区中，否则由用户选择布局，自行拖拽容器或组件到工作区中进行界面定制。在需要时用户可点击预览功能，此时解析器根据用户在界面拖拽的配置解析和展示预览界面。

* 保存

点击保存按钮调用saveMenuCompCfgs保存菜单组件配置和组件参数。

# 开发环境

|  |  |  |
| --- | --- | --- |
| 工具类型 | 软件名称 | 备注 |
| 服务器端开发工具 | eclipse、Myeclipse、idea、STS |  |
| 页面开发工具 | idea、editplus、plus++、ue |  |
| 运行环境工具 | JDK1.6及以上 |  |
| 数据库开发工具 | plsql |  |

# KJDP开发工程配置

## 安装JDK及配置

这里使用的是JDK1.6，可以到oracle网站去下载：<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

注意版本，1.6以上或以下的版本都未经过测试，可能会有兼容性问题。

下载完成后，即可安装到系统目录下，这里以E:\JDK1.6为例。

注意路径，路径选择最好不要有中文，容易出莫名其妙的问题。

安装完成后需要配置环境变量。

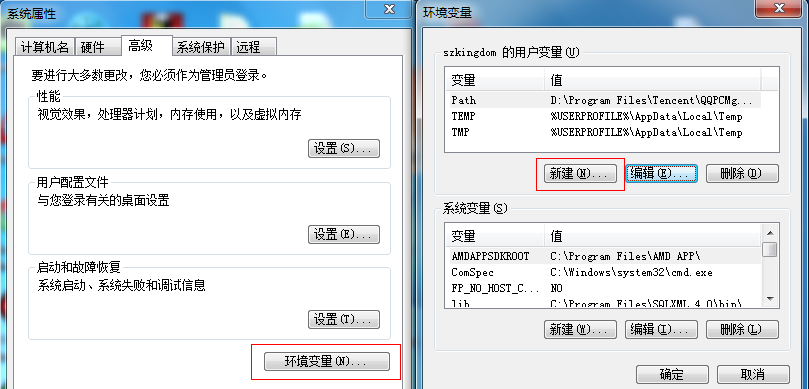
配置环境变量如下：

JAVA\_HOME = E:\JDK1.6

PATH = %JAVA\_HOME%\bin;%JAVA\_HOME%\lib;%JAVA\_HOME%\jre\lib;

CLASSPATH = %JAVA\_HOME%\lib;%JAVA\_HOME%\jre\lib;

步骤如下：右键我的电脑 -> 高级系统设置 -> 高级 -> 环境变量 -> 新建



## 安装Tomcat及配置

这里使用的是Tomcat6.0.35，可以到apache网站去下载：<http://tomcat.apache.org/download-60.cgi>

这里可以选择绿色版和安装版，两者均可。

注意版本，6.0以上或以下的版本都未经过测试，可能会有兼容性问题。

注意路径，路径选择不要有中文或空格字符，否则容易出莫名其妙的问题。

在tomcat中进行工程部署：

1、打开apache-tomcat-6.0.32\conf\server.xml

2、找到最后一个Host标签

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true"

xmlValidation="false" xmlNamespaceAware="false">

</Host>

1. 在中间加入Context标签

<Context path="/" docBase="平台工程webapp路径" debug="0"

crosscontext="true" reloadable="true" >

</Context>

安装完成后需要配置环境变量。

配置环境变量如下：

TOMCAT\_HOME = 你的安装路径

## 安装浏览器firefox或Chrome

建议安装浏览器firefox或chrome。

## 安装开发工具

开发工具可选Myeclipse、IDEA、UE、NotePLUS等

## 安装数据库

使用PL/SQL做为客户端工具，在TNS配置数据库连接地址，例如：

UAS =

(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST =192.168.50.210)(PORT = 1521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SERVICE\_NAME = UAS)

)

)

## 安装FireFox和FireBug及配置

Firefox下载地址：<http://www.firefox.com.cn/download/>。

下载完成后直接安装即可。

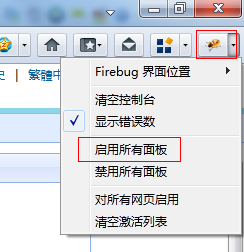
Firebug安装：进入火狐应用中心，搜索firebug然后点击安装即可。



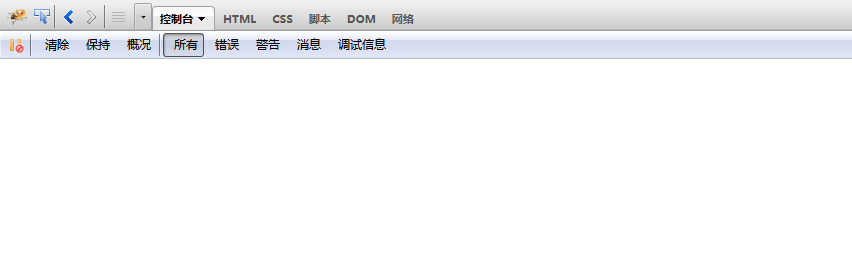
## 前端功能调试

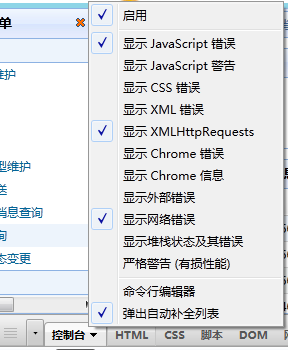
Firefox调试:

在浏览器右上角显示有Firebug的Logo，点击启用所有面板开启Firebug。

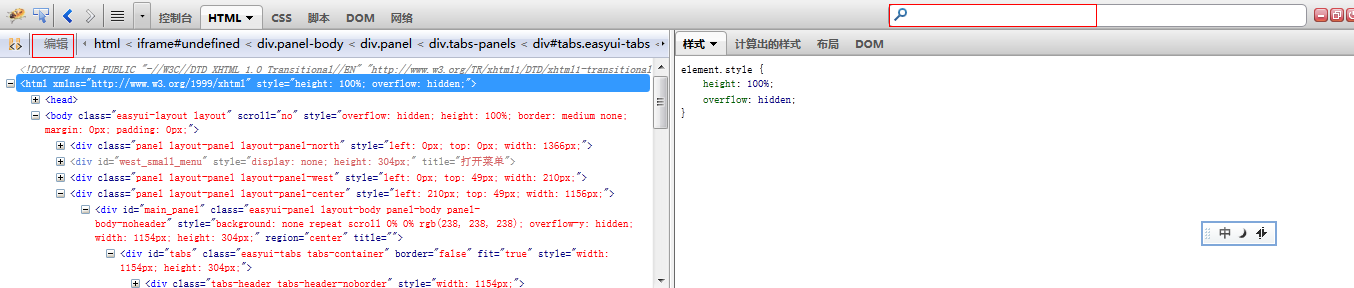


按F12可以进入Firebug的调试界面。

其中控制台用于各种信息的输出，输出消息类型可定制。

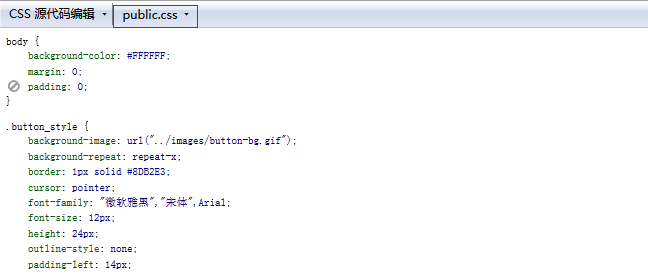


HTML标签用于显示和编辑所有的HTML元素。

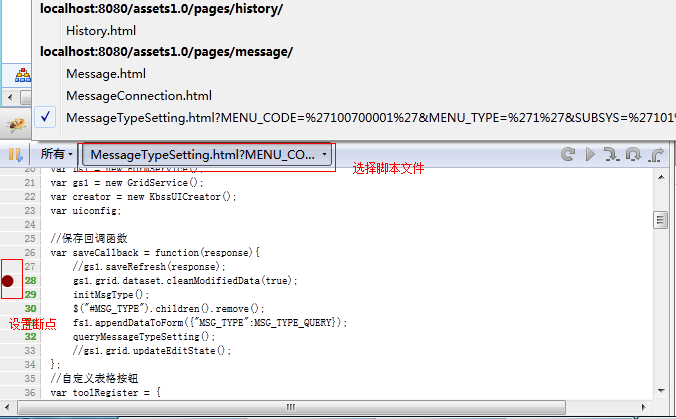


搜索框可用于定位页面关键字，选定某一元素后，可点击编辑按钮进行实时编辑。标签属性可以直接双击进入编辑。

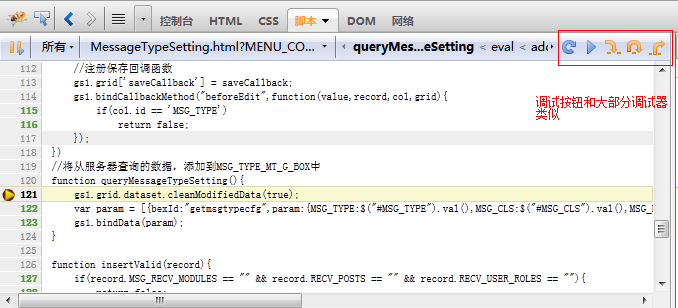
CSS标签用于显示和编辑所有的CSS元素。



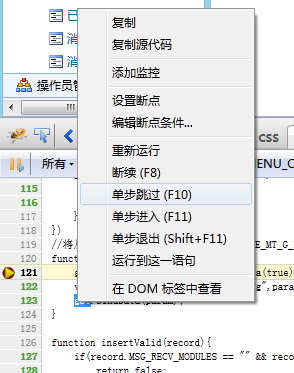
同样可以用搜索框进行关键字定位，点击属性名和属性值可以直接进行编辑。脚本标签用于调试所有页面Javascript代码。



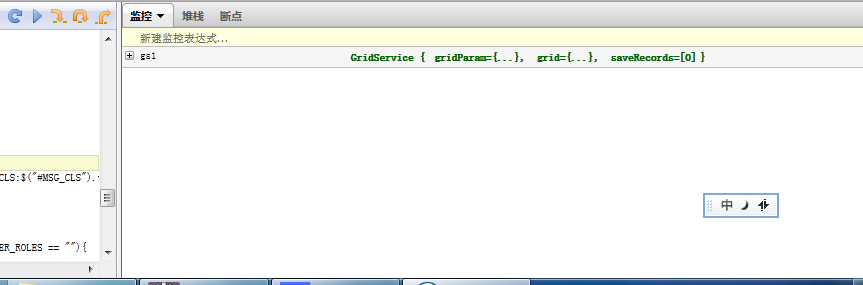
进行相关操作激活断点



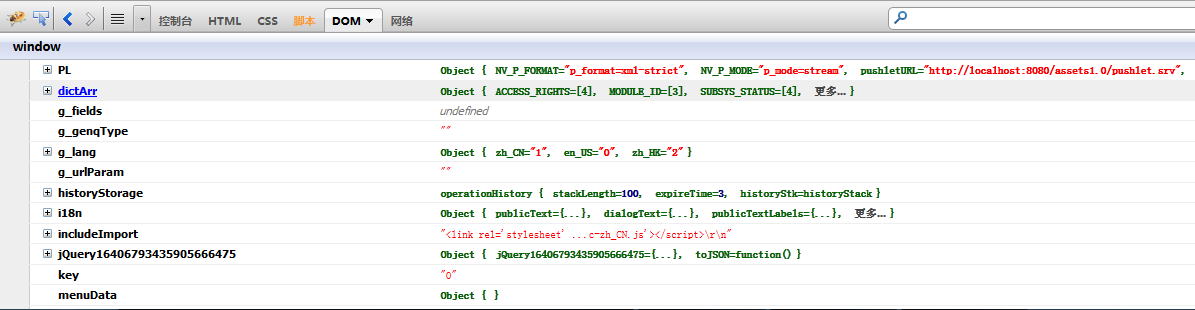
选定一变量或语句后可以右键操作，如可以添加到监控区等。



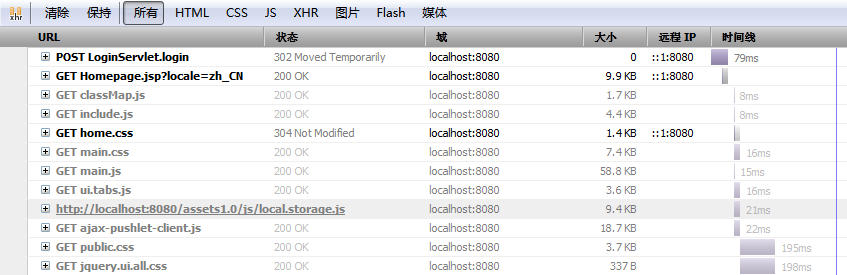
右边有监控、堆栈和断点三个标签，其中比较常用的监控区用于实时查看变量值，断点区用于查看和定位所有设置的断点。



DOM标签用于查看Window下的所有DOM元素。



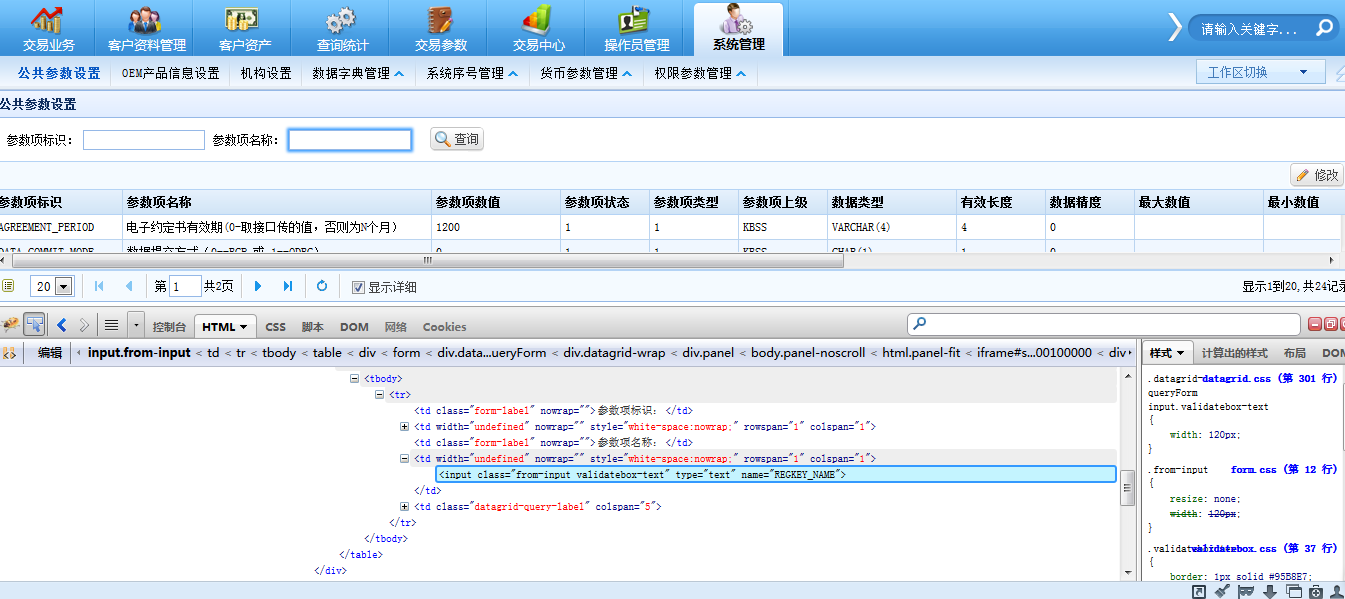
网络标签用于查看所有需要和服务器交互的请求信息。



其中头部的标签栏可以对和服务器端交互的信息进行分类，大小和时间线可以用于性能测试及调优。

另外，Firebug还可以通过以下两种方法定位元素。

1、通过Firebug工具栏定位标签



2、直接通过右击页面某元素进入



以上概括了Firebug常用功能，更多的用法请参照Firebug官方文档：

[http://getfirebug.com/wiki/index.php/Main\_Page](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

IE调试,按F12呼出调试界面。下载地址：

<http://www.microsoft.com/en-us/download/details.aspx?id=22185>

使用方法类似firebug，获取更详细信息请参照微软官方文档。

# 编程规范

## 版面书写规则（JAVA）

* 空格与制表符

源程序应使用空格替代制表符，有利于在不同的编辑环境中保持源程序的版面统一。

* 缩进

**缩进单位**：两个空格为一个缩进单位。

**代码域块**：具有独立局部变量作用域空间的代码块，以大括号为域块的开始和结束，如：函数体，循环体等。

* 域块的内容缩进一个单位，域块嵌套也递归缩进一个单位数。
* 语句换行，要相对行首缩进一个单位。
* 大括号
* 左右大括号独立行，且与上一语句的行首对齐。
* 程序流程控制语句（如：if、while、for等）其后的代码域块应使用大括号镶套，即使只有一条语句，也应使用大括号镶套。

示例：

|  |
| --- |
| private int iCondition = 0; |
| ...... |
| if (iCondition == 3) |
| { |
| for (int iIndex = 0; iIndex < 8; iIndex++) |
| { |
| ...... |
| } |
| } |

* 空格间隔
* 所有二元运算符（加减乘除与或、比较运算、赋值运算等）的前后都用空格间隔。
* 分隔符（分号、逗号等）前面不用空格间隔，后面用空格间隔。
* 流程控制关键字（如：if、while、for等）与其后面的括号之间用空格间隔。
* 函数名称后紧跟括号，之间不使用空格间隔。

示例：

|  |
| --- |
| ...... |
| // ...... |
| for (int iIndex = 0; iIndex < strlen(szData); iIndex++) |
| { |
| ...... |
| } |

* 注释
* 注释采用简体中文。
* 对于文件注释、函数注释的起始位置应顶行书写。
* 语句注释，如果注释内容较长，则可以把注释内容放予语句的前一行，并与需注释的语句对齐；如果注释内容较短，则可以放在语句后，起始位置为：16 \* n + 1 = {17, 33, 49, 65, 81, ......}。

## 命名规则（java）

|  |  |  |
| --- | --- | --- |
| **标识符类型** | **命名规则** | **例子** |
| 包 (Packages) | 一个唯一包名的前缀总是全部小写的ASCII字母并且是一个顶级域名，通常是com，edu，gov，mil，net，org，或1981年ISO 3166标准所指定的标识国家的英文双字符代码。包名的后续部分根据不同机构各自内部的命名规范而不尽相同。这类命名规范可能以特定目录名的组成来区分部门(department)，项目(project)，机器(machine)，或注册名(login names)。 | com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese |
| 类 (Classes) | 命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像URL，HTML) | class Raster; class ImageSprite; |
| 接口(Interfaces) | 命名规则：大小写规则与类名相似 | interface RasterDelegate; interface Storing; |
| 方法 (Methods) | 方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。 | run(); runFast(); getBackground(); |
| 变量(Variables) | 除了变量名外，所有实例，包括类，类常量，均采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。 变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为i，j，k，m和n，它们一般用于整型；c，d，e，它们一般用于字符型。 | char c; int i; float myWidth; |
| 实例变量(Instance Variables) | 大小写规则和变量名相似，除了前面需要一个下划线 | int \_employeeId; String \_name; Customer \_customer; |
| 常量(Constants) | 类常量和ANSI常量的声明，应该全部大写，单词间用下划线隔开。(尽量避免ANSI常量，容易引起错误) | static final int MIN\_WIDTH = 4; static final int MAX\_WIDTH = 999; static final int GET\_THE\_CPU = 1; |

## Mybaties sql书写规则

* MYBATIS内所有的EXEC SQL语句全部采用大写字母书写，宿主变量除外。
* SELECT语句书写规则

SELECT 表字段列表

INTO 宿主变量列表

FROM 数据表

WHERE 条件表达式

GROUP BY 统计表达式

ORDER BY 排序表达式

备注：

1.表字段列表之间以逗号相隔，逗号前不用空格间隔，逗号后用空格间隔，换行与上一行的第一个字段对齐。

2．宿主变量列表之间以逗号相隔，逗号前不用空格间隔，逗号后用空格间隔，宿主变量列表的换行与表字段列表的换行相一致，换行与上一行的第一个宿主变量对齐。

3.数据表之间以逗号相隔，逗号前不用空格间隔，逗号后用空格间隔，换行与上一行的第一个表名对齐。如果存在多表，则每一表都必须定义别名，且语句内所有的表字段都必须灌于表别名。

4.条件表达式以“与(AND)或(OR)”关键字为一节，建议每一节为一行，每节相互对齐。每一节通常将主表字段放在左边。

5.统计表达式、排序表达式的书写规则与表字段列表的书写规则相同

示例：

SELECT A.CUST\_CODE, A.CUST\_NAME, A.ID,

A.TEL, A.EMAIL

INTO lCustCode, szCustName, szID,szTel, szEmail

FROM CUSTOMERS A

WHERE A.CUST\_CODE = lCustCode OR A.ID = szID;

* INSERT语句书写规则

INSERT INTO 数据表

(表字段列表)

VALUES

(宿主变量列表);

INSERT INTO数据表

(表字段列表)

SELECT ......;

备注：

1.表字段列表之间以逗号相隔，逗号前不用空格间隔，逗号后用空格间隔，换行与上一行的第一个字段对齐。

2.宿主变量列表之间以逗号相隔，逗号前不用空格间隔，逗号后用空格间隔，宿主变量列表的换行与表字段列表的换行相一致，换行与上一行的第一个宿主变量对齐。

3.对于子查询式的INSERT语句，子查询的书写规则参照SELECT语句书写规则，子查询里表字段列表的换行与INSERT表字段列表的换行相一致。

示例：

INSERT INTO CUSTOMERS

(CUST\_CODE, CUST\_NAME, ID, TEL, EMAIL)

VALUES

(lCustCode, szCustName, szID, szTel, szEmail);

* UPDATE语句书写规则

UPDATE 数据表

SET 字段赋值列表

WHERE 条件表达式;

备注：

1.字段赋值列表以一个字段的赋值为一节，每节一行，换行与上一节对齐。

2.条件表达式以“与(AND)或(OR)”关键字为一节，建议每一节为一行，每节相互对齐。每一节通常将主表字段放在左边。

示例：

UPDATE CUSTOMERS

SET TEL = szTel,EMAIL = szEmail

WHERE CUST\_CODE = lCustCode;

* DELETE语句书写规则

DELETE FROM 数据表

WHERE 条件表达式;

备注：

1.条件表达式以“与(AND)或(OR)”关键字为一节，建议每一节为一行，每节相互对齐。每一节通常将主表字段放在左边。如果条件表达式只有一节，则允许将WHERE字句直接跟在数据表后面

示例：

DELETE FROM CUSTOMERS WHERE CUST\_CODE = lCustCode;

## 注释描述规则

**js文件注释描述**：应该在每个js文件开头对文件进行描述说明。

/\*\*

\* 文件名称：系统工具类

\* 文件描述：提供显示tip、弹出对话框、表格对齐等功能

\* 开发作者：中文姓名

\* 创建日期：2012-02-06

\* 文件版本：1.0

\*/

**函数注释描述**：应于对每个js函数进行描述说明。

/\*\*

\* 功能描述：提供显示tip、弹出对话框、表格对齐等功能

\* 基本用法：$.openWindow(jsonMessage);

\* 入参说明：JSON格式的文本

完整格式为：{type:"",title:"",content:"",width:"",height:"",callback:""};

type：支持5种对话框info/warn/error/confirm/iframe/window

title：对话框标题

content：对话框内容

width/height：对话框的宽度和高度

callback：点击对话框的确定按钮时的回调函数

\* 返回说明：无

\*/

**功能域块注释**：建议对重要的功能域块进行注释，注释内容应位于域块的首语句前，并对齐。

**语句注释**：建议对程序重要的语句进行注释。

## 程序开发约定

* **html文件**

在客户端开发过程中，禁止使用JSP服务器端脚本，一些特殊功能需要用到JSP有框架开发组进行统一开发，业务模块的开发一律使用html文件。

需要将业务开发文件统一存放在\webapp\business目录

business目录规范介绍：

-|css 所有业务页面使用的样式表文件存放目录

-|images 所有业务页面图片存放目录

-|js 所有业务页面使用的javascript脚本存放目录

-|业务模块 如custmanager

注:业务模块目录(如：custmanager)下不可建立子目录并存放业务页面。

-|业务模块.html

客户端程序开发框架封装已经比较完善，在模块开发过程中，html文件的<head/>元素内链接其他文件时遵循如下几点：

所有模块的页面，必须链接kui-base/js/core/jquery-1.8.3.min.js、kui-base/js/core/kui.loader.js文件，

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html >

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>

<title>业务页面</title>

<link rel="stylesheet" type="text/css" href="../../kui-base/themes/default/easyui.css">

<script type="text/javascript" src="../../kui-base/js/core/jquery-1.8.3.min.js"></script>

<script type="text/javascript" src="../../kui-base/js/core/kui.loader.js"></script>

</head>

* **JavaScript代码**

**语法**：由于客户端框架引入了jQuery，所以在编写JS代码时，要求优先使用jQuery的写法而少用甚至避免使用原生的JavaScript写法，特别是在访问和控制DOM对象、事件绑定、遍历元素等方面。但在封装一个独立的插件时除外（减少依赖）。示例如下：

var userName = document.getElementById("USER\_NAME").value; //避免

document.getElementById("DIV\_TIP").style.display = "block"; //避免

document.getElementById("DIV\_TIP").innerHTML = "test"; //避免

var userName = $("#USER\_NAME").val(); //正确

$("#DIV\_TIP").css("display", "block").html("test"); //正确

**JS程序入口：**在html文件中编写JS程序时，统一将执行的入口程序放在JS代码段落的末尾，做法类似main()函数。

<script type="text/javascript">

var ajax = new AjaxEngine();

function renderTree() {

//TODO

}

function loadChkObjectDDL() {

//TODO

}

/\*\*

\* 程序入口，初始化页面

\*/

window.onload = function() {

renderTree();

loadChkObjectDDL();

//Others

}

</script>

**函数还是对象**：在实现一个模块的过程中，通常都需要写一些JS函数用来处理不同的任务，那么到底是通篇都写成一个一个的函数还是按模块封装成对象，这需要权衡模块的复杂度和功能多少。推荐的做法是在开始编码前先弄清楚有哪些函数，当函数较多让你需要提取成JS文件时，那么你需要按功能进行划分，并封装成若干个对象。此外，当你写的JS函数有可能被其他模块重用时，你需要按类型进行划分并封装成对象以便复用。

**对象的写法：**JavaScript定义对象典型的有构造函数和原型等方式，在本框架下，基本都采用构造函数方式，示例如下：

function GridService(gridParam) {

this.gridService = null;

this.gridParam = gridParam || null;

this.setGridParam = function(gridParam) {

this.gridParam = gridParam;

};

this.loadProcess = function() {

};

this.saveProcess = function() {

var index = 0;

};

}

**代码保存位置规范**：通常为了简单方便和提高开发效率及运行效率，JS代码可以直接写在网页的<script/>标记里，但是为了避免在一个网页内写入过多的内容而降低可读性，所以当JS代码达到5屏（大约150行）时建议提取成一个JS文件，再通过链接的方式放在网页的<head/>内。

**优化选择器的执行速度：**在使用代码访问元素时，如果有id号，建议使用id号直接访问元素，这样速度是最快的；如果没有id号，可使用元素标记访问，其次就是使用类别（class）进行访问。

**减少对DOM元素的直接操作：**在网页中反复对DOM进行操作，尤其是循环或递归操作，其性能是很低的，会造成在网页上显示结果需要等待数秒甚至更长时间。因此为了减少对DOM元素直接操作的次数，有必要在操作前完善大部分的DOM操作，最后通过一次直接操作，更新其DOM结构。常用的做法是先在循环或递归中拼字符串，最后一次性完成DOM元素的增加。

* **JSON数据格式**

在本框架下，比较广泛地运用到这种数据格式，下面通过示例对其定义和使用做简要说明：

var json1 = {param1: "参数一", param2: "参数二"};

var json2 = [{attr1: "属性一"},{attr2: "属性二"}];

var json3 = {parent1: [{leaf1: "叶子一"}, {leaf2: "叶子二"}]};

alert(json1.param1); //返回“参数一”

alert(json2[0].attr1); //返回“属性一”

alert(json3.parent1[0].leaf1); //返回“叶子一”

if ($.isEmptyObject(json1)) {} //判断是否为空

$.getJsonKey(json1); //返回所有的key数组，即：["param1", "param2"]

$.toJSON(json1); //把json1转换成标准的JSON数据格式

for (key in json1) { //遍历json1

alert(key + " = " + json1[key]);

}

# 附注

## KJDP平台发展过程

