计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(1):80−85 [doi: 10.15888/j.cnki.csa.007251]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Activiti+MongoDB 的后勤自由流程信息系统①

刘　聪, 李世川, 吕雪峰

(中央军委后勤保障部 信息中心, 北京 100842)
通讯作者: 刘　聪, E-mail: congliu2005@163.com

摘　要: 较大型单位的后勤保障时刻离不开流程管理信息系统, 传统方法采用固定流程图和固定表单的方式, 无法适应新形势下后勤灵活自主管理的要求. 同时, 传统信息系统采用的关系型数据库方式在适应当前后勤保障大数据时也捉襟见肘. 本文设计并实现了使用 Activiti 和 MongoDB 的后勤自由流程信息系统, 以自由灵活流程管理为目标, 以关系型数据库辅助非关系型数据库为设计理念, 给出了自由流程框架、关键接口和重要代码逻辑, 以及两类数据库协同的设计方法. 实现的系统有效地解决了上述问题, 较大地减少了开发人员的工作量, 为当前大型企业或单位后勤保障流程管理信息化提供了一个较高效的系统设计参考.

关键词: Activiti; MongoDB; 后勤; 自由流程

引用格式: 刘聪,李世川,吕雪峰.基于 Activiti+MongoDB 的后勤自由流程信息系统.计算机系统应用,2020,29(1):80–85. http://www.c-s-a.org.cn/1003-3254/7251.html

Logistics Free Process Information System Based on Activiti+MongoDB

LIU Cong, LI Shi-Chuan, LYU Xue-Feng

(Information Center, Logistics Support Department, CMC, Beijing 100842, China)

Abstract: The logistics support of large units can not be separated from the process management information system at any time. The traditional methods adopt the fixed flow diagrams and the fixed forms, which can not meet the requirements of flexible and independent logistics management under the new situation. At the same time, the relational database mode adopted by traditional information systems is also difficult to adapt to the current big data in logistics support. This study designs and implements a logistics free process information system using Activiti and MongoDB. With the goal of free and flexible process management and the idea of relational database assisting non-relational database as design concept, the free process framework, key interfaces and important code logic are given, as well as the design method of two kinds of database collaboration. The implemented system effectively solves the above problems, greatly reduces the workload of developers, and provides an effective system design reference for logistics support process management informatization of large enterprises or units.

Key words: Activiti; MongoDB; logistics; free process

Activiti 是当前流行并重要的业务流程管理框架. 获得 Apache 许可, 它具有轻量级、可嵌入的优点, 同时还被设计适用于可扩展的云架构. Activiti 具有以下特点: 一是开源特性, 它是一个开源项目[1], 适用者可以阅读源码理解工作原理, 也可以修改源代码实现自定义功能. 二是使用 MyBatis 框架, MyBatis 具有轻量化的特点, 利于上手和后期优化, 使用该框架可加快数据的持久化. 三是可融合 Spring, 它提供了便捷的 Spring

① 收稿时间: 2019-06-17; 修改时间: 2019-07-12; 采用时间: 2019-07-26; csa 在线出版时间: 2019-12-27

接入机制, 可充分利用 Spring 优势, 快速实现项目开发[2,3]. 四是支持 BPMN2.0 规范, BPMN2.0 用标准定义了描述业务的图元和规范的执行语义, 保证在不同的流程引擎得到的结果一致[4]. 五是支持广泛的数据库, 它支持当前主流数据库如: Oracle、MySQL、PostgreSQL、DB2 等.

MongoDB 是一个目前流行使用的非结构化数据库, 广泛应用于大数据应用的后台, 是非关系型数据库中最像关系数据库和功能最丰富的一款. 它支持松散的数据结构, 类似于 JSON 的 BSON 格式, 用来存储较为复杂的数据类型. 具有以下显著特点: 一是面向集合存储, 集合类似于关系数据库中的表, 一个集合中可包括任意多的文档; 二是模式自由, 集合中存储的数据是无模式的, 这是区别于关系数据库的重要特征; 三是在支持索引、查询等传统关系数据库的功能之外, 具有强大聚合工具, 如 count, group 等, 支持采用 MapReduce 完成的聚合任务; 四是文件存储采用 BSON 格式, 它是二进制格式 JSON 的缩写, 支持文档与数组的嵌套[5,6].

后勤业务涵盖面比较广泛, 例如物资请领与采购、财务管理、营房管理、公车使用、人员管理、办公办文等等方面. 一个较大单位的后勤保障必然涉及到上述的诸多方面, 这些方面的信息化管理最常用的需求就是流程审批与管理, 不夸张地说, 后勤业务系统一刻也离不开流程使用[7,8]. 现有系统或使用标记位定义的流程, 随意性较大, 修改复杂; 或使用 BPMN1.0 的内核, 较为陈旧; 或使用已完全封装好的流程软件, 不利于修改流程. Activiti 具有开源的优势, 且可以自定义流程, 通过开发可实现自由流程的源代码, 利于后勤业务系统.

同时, 后勤业务涵盖面广, 产生的数据量也大, 是最容易产生大数据的领域[9]. 现有的关系型数据库在非结构性数据的存储处理上捉襟见肘, 限制了快速高效地操作后勤数据. 适应新需求新前沿, 引入主流非关系型数据库 MongoDB, 可以更好满足大数据的要求. 将 Activiti 实现的自由流程和 MongoDB 结合起来, 可以实现一种全新的后勤自由流程框架, 带来独特的实现效果. 文献[2]中使用在线审批和 Activiti 相绑定, 形成了实用在线审批流程算法, 本文则侧重于利用 Activiti 实现自由流程流转. 文献[5]中利用 Node.js 和 MongoDB 搭建的重点在实现高性能、维护成本低 Web 框架, 本文则重点利用 MongoDB 服务于流程的高效性.

1 自由流程框架

传统带流程的系统, 常常事先画好复杂工作流程图, 写死逻辑代码, 一旦人员机构发生变化带来轻微的修改, 都给程序员带来大量的工作. 不指定流程流转的下一个人、不绑死流程流转的业务表, 在系统中实现一个基本的自由流程框架很有意义.

图 1 显示了利用 Activiti 实现自由流程的示意图. 从左至右, 流程启动后, 用户填写相应业务申请表单, 并手动指定呈批对象, 流程会自动流转到该审批对象. 审批对象完成相应批示后, 既可以继续呈批到更高级的审批人, 也可以返回申请人修改申请. 如果是继续向上呈批, 将循环前述审批步骤; 如果是返回申请人修改, 修改后可以自行决定是否重新呈批. 最终审批人完成审批意见后可直接返回申请人确认, 如果申请人认为工作完成则选择结束流程, 如果认为流程还需其他人员协助完成, 还可继续流转到下一审批人, 直至任务完成后再由本人结束.

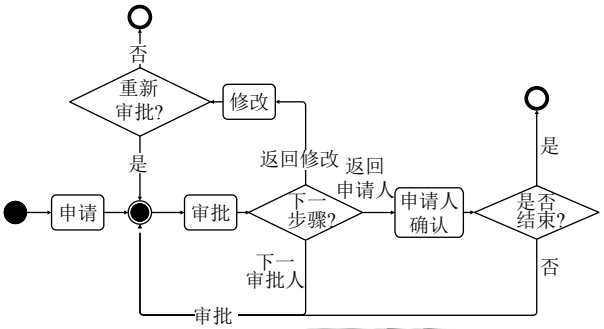

图 1 Activiti 实现自由流程图

该自由流程框架有如下 3 个特点: (1) 不自动指定流程流转的下一人, 由用户根据单位线下实际工作的流程自行指定下一人, 此举利于适应单位组织人员的变化; (2) 不绑定流程流转的业务表, 在流程启动前, 由用户选择业务表, 这样利于流程与业务表独立出来, 高效实现各自功能; (3) 给予申请人充分自主权, 流程形成闭环. 流程由申请人发起, 最后由同一人结束, 符合信息流程闭环的要求, 申请当事人可及时充分地知悉办事过程, 能起到一定的监督作用.

2 关键接口与代码逻辑

表 1 给出了该流程框架实现的代码接口和相应意义. 其中接口 1、2 和 4 的意义最为典型, 接口 1 的意义为: 在起始申请和中间节点时, 用户填写完相应表单后, 提交给流程处理. 接口 2 为用户作为中间节点的处

理人, 处理待办事件时的逻辑功能. 接口 4 是某个流程的详细历史记录. 接口 3 为撤回发出的任务. 而其他接口则是接口 4 功能的延伸, 为实现较完备功能而准备, 实现原理类似.

表 1 流程重要接口实现

序号	接口	接口意义
1	/platform/save	提交和保存任务节点, 向下一节点流转
2	/doTask/{taskId}	开始中间任务节点流程
3	/withdraw/{parentId}	撤回任务
4	/historyList/{keyId}	查看某个流程的详细记录
5	/unfinished/{userId}	某人发起的未完成的流程列表
6	/finished/{userId}	某人发起的已经完成的流程列表
7	/todo/{userId}	个人待办列表
8	/participated/{userId}	某人参与的, 包括发起的和中间处理的, 已完成和未完成的

算法 1. 开启新流程逻辑伪代码

输入: 当前用户 id—userId

```
01 startUserId = userId;//初始用户 id 预留好.
02 map.put("leader", startUserId); //编辑节点都是由申请者完成.
03 String processDefinitionKey = "generalduty";//流程名
04 ProcessDefinition processDefinition = repositoryService
.createProcessDefinitionQuery()
.processDefinitionKey(processDefinitionKey)
.latestVersion().singleResult();
05 String processDefinitionId = processDefinition.getId();
06 String uuidBussness = this.get32UUID();
07 String bussness = uuidBussness+","+tableId;
08 identityService.setAuthenticatedUserId(userId);
//以 businessKey(包含多种信息--bId 与 tableId) 方式启动流程
09 ProcessInstance processInstance = runtimeService
.startProcessInstanceById (processDefinitionId, bussness, map);
10 String pid = processInstance.getProcessInstanceId();
//存储流程实例信息, 放入表 universe
11 String uuidUniverse = this.get32UUID();
12 PageData pdUniverse = new PageData();
13 pdUniverse.put("ID", uuidUniverse);
14 pdUniverse.put("PID",pid);
15 pdUniverse.put("CREATETIME",createTime);
16 try{ universeService.save(pdUniverse);
17    } catch (Exception e4) e4.printStackTrace();
//得到初始编辑节点的 taskId
18 List<Task> tasks = new ArrayList<Task>();
19 tasks = taskService.createTaskQuery()
.processDefinitionKey(processDefinitionKey)
.taskCandidateOrAssigned(userId)
.active().orderByTaskCreateTime()
.desc().listPage(0, 10);
20 Task firstTask = tasks.get(0);
21 taskId = firstTask.getId()
```

算法 2. 中间节点流转逻辑伪代码

输入: 流转用户名—nextUserName, 当前任务 Id—taskId, 流转标记—approval, 当前任务—task

```
01 Task task=taskService.createTaskQuery().taskId(taskId)
.singleResult();
02 String pId =task.getProcessInstanceId();
//对各个任务节点需要分别进行处理
03 switch(task.getTaskDefinitionKey()) {
04 case "usertask1":
//0-在流程图里转向下一个审批人/转办 (14)
05 if (approval.equals("0") || approval.equals("14")){
06    if(nextUserName.equals(username)){
07          msg="请提交给正确的业务审核人员!";
08          errInfo="false";
09          map1.put("msg", msg);
10          map1.put("result", errInfo);
11          return AppUtil.returnObject(new PageData(), map1,true);
12    } else {
13          pdUserId.put("USERNAME", nextUserName);
14          nextUserId = userService.findByUId(pdUserId)
.getString("USER_ID");
15          map.put("leader", nextUserId);
16          message = "请尽快完成审批! ";
17            bussMessageService.sendMessage(sendSmsUser,
nextUserName, processDefinitionKey,message); }}
//1-转给申请人修改
18 else if(approval.equals("1")){
19       startUserId =mapData.getString("userId");
20       map.put("leader",startUserId);
21       PageData apllypd = new PageData();
22       applypd.put("USER_ID", startUserId);
23       apllypd= userService.findByUiId(apllypd);
24       String recieveUser = apllypd.getString("USERNAME");
25       String message =null;
26       nextUserName=recieveUser;
27       message="请申请人修改申请表! ";
28       bussMessageService.sendMessage(sendSmsUser, recieveUser,
processDefinitionKey,massege);
}
//11-转向申请人确认
29 else if(approval.equals("11") || approval.equals("17")){
...
//以请假流程为例, 进行表的更新.
30          if(tableId2.equals("leave")){
31             mapData.put("isPass", "1");
32             freeModel.setMap(mapData);
33             freeModel.setId(bId2); //update 时, 需要 Id 关键字
34             freeDao.update(freeModel,tableId2);
          }
35          Message = "请申请人确认! ";
36            bussMessageService.sendMessage(sendSmsUser,
recieveUser, processDefinitionKey,message);
```

```
37      }
38      break;
39 case "usertask2":…
40      break;
41 default:…
}
//根据流程分支条件, 往下一节点流转
42 taskService.complete(taskId, map)
```

梳理这些接口的核心代码, 其中两部分关键代码逻辑值得特别展示. 算法 1 和算法 2 给出了两段关键原理代码逻辑, 它们分别实现: 开始一个新流程和流程的中间节点流转. 此两部分代码构成了自由流程正常流转的核心部分. 需要说明的是: 一个流程 (process), 在每一个节点处都有一个任务 (task), 即一个 process 包含多个 task. 此外, 此两表只给出核心代码的原理逻辑, 其中用不少伪代码示意.

3 数据表设计关系

图 2 给出了本文自由流程框架的数据库表设计关系图, 表中的箭头方向为一对多的关系. 其中浅灰色表为关系型数据库表, 深灰色表为非关系型数据库表[10]. 表 act_hi_procinst 和 act_ru_task 是 Activiti 工作自带的关键表, 前者存储了已完成流程的详细记录, 后者存储了已启动、尚未结束流程的记录. 表 log 则不是 Activiti 的保留表, 它用来存储流程中各节点的处理意见, 需要说明的是, 它并不是业务表, 而且配合流程使用的流程日志表.

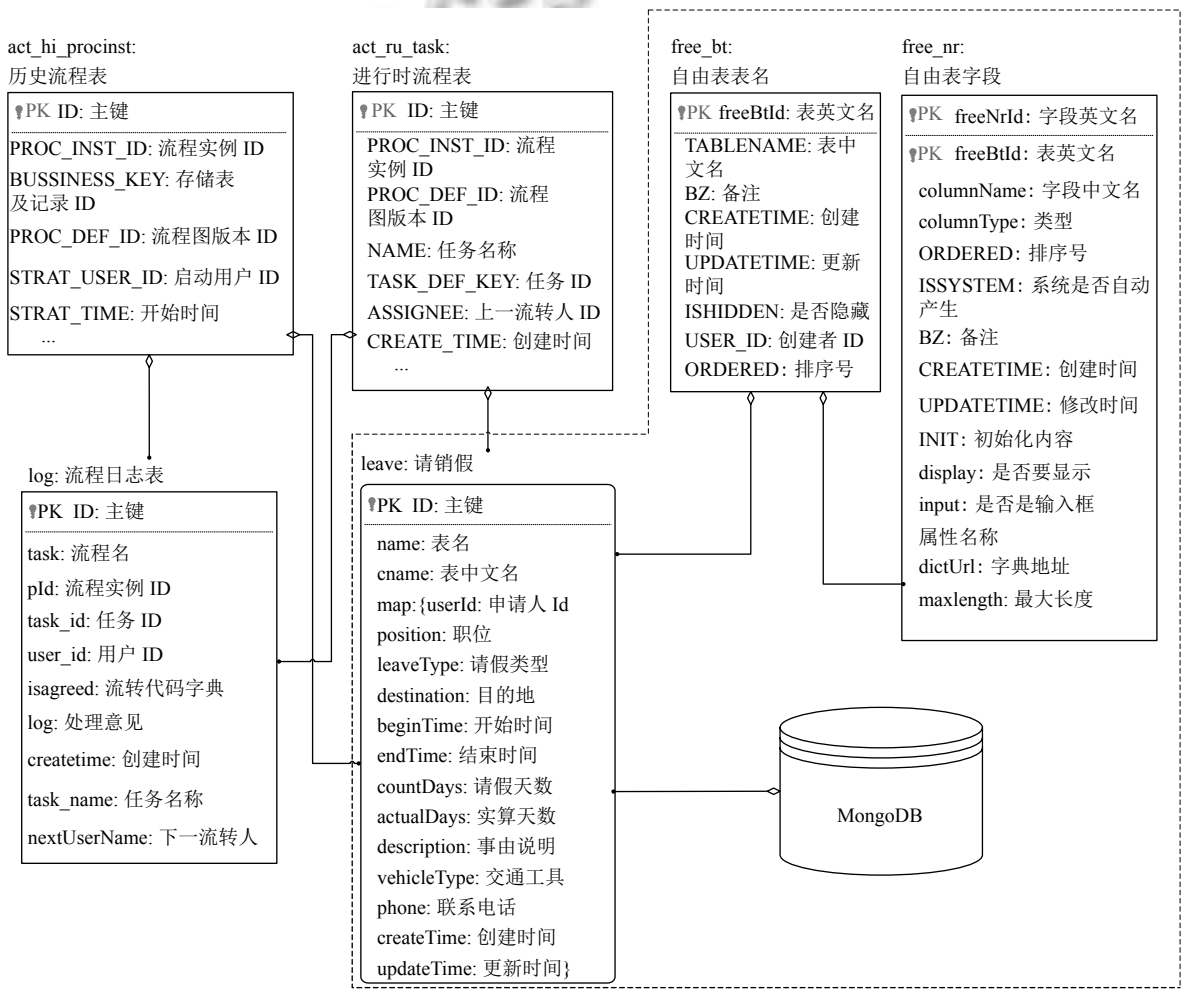

图 2 数据结构设计关系图

图 2 中虚线框部分给出了非结构化自由业务表原理. 深灰色标注表 leave 为举例业务表——请销假表, 它存储在 MongoDB 中, 表中列出了请销假需要的主要字段. 这种业务表可以根据用户的需要自由添加, 并可

修改对应字段, 当用户添加了一个新业务表时, 关系型表 free_bt 会记录下该表的相关信息. 当用户对新表字段进行添加或修改时, 关系型表 free_nr 又会对相应字段信息进行登记. 在前端配合的情况下, 后端可以自如地添加修改表及表的字段结构, 具体实现效果在第 4 部分中展示.

除了单一流程, 还会存在如下情形: 在某个流程尚未结束时, 用户需要启动其他流程并在其中流转. 图 3 给出了延伸为多流程的数据表设计关系图, 该示意图对应需求为: 在用户请假的同时需要申请派车送站. 其中灰色部分仍为关系型表, 深灰色部分为非关系型表. 深灰色部分表 leave 和 applyCar 对应请假和派车的业务表, 表 act_hi_procinst 和 act_ru_task 同图 2 的意义, 表 multiProcess 用来管理多个流程. 在设计中, 后台视之为实质是同一个流程的扩展, 流程实例和 id 无需变化, 只需变换相应业务表的对应关系, 让携带新业务表的流程继续在自由流程框架中流转, 直至结束, 后再切换回前一流程对应的业务表继续流转. 当然, 也可以做到两个业务表不切换, 让它们同时流转, 以利于后一流程处理人更加直观方便知悉相关情况.

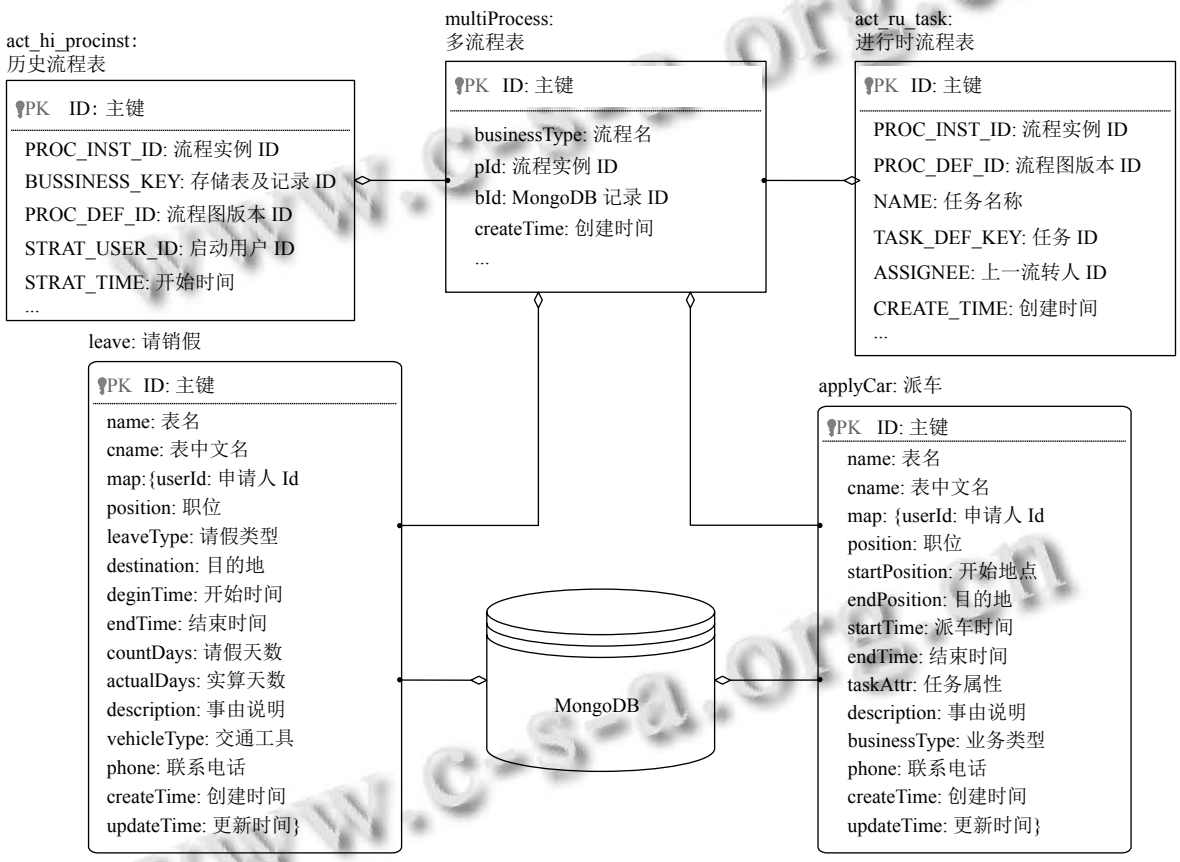

图 3 多流程的数据结构设计关系图

4 实现效果

(1) 自由流程的应用场景. 自由流程可以应用到一个单位很多的后勤事务工作中去, 详见表 2.

(2) 以请销假流程为例说明自由流程使用达到的效果.

图 4 至图 6 用实例给出了请销假自由流程的过程, 图 4 中的承接人可以让用户选择任意人完成对请假的审批工作. 对于承接人, 在完成了相应审批 (图 5) 时, 他的上级审批人同样由他选择, 从而实现申请-审批工作的自由灵活, 最后结果可在图 6 的历史详情页中获悉.

表 2 自由流程应用场景

序号	应用场景
1	单位人员请销假
2	内部人员派遣
3	单位公车派车使用
4	办公物资请领
5	其他事务处理: 包括办文流转、值班登记、维修保障、营房维修、文件传阅、电话咨询等

图 4 请假申请

图 5 审批过程

图 6 请假历史页

另外, 用户通过该系统能获取参与但尚未结束的流程的各步骤详情; 还能在每次进入个人主页时, 看到需要待办的事务. 由于篇幅所限, 并不在此一一展示. 对于通用事务处理, 系统内设了一些业务处理单, 它们对应着不同的流程业务类型, 可以为不同的后勤管理进行自由流转, 从而满足相应的工作需要.

(3) 灵活添加业务表单. 如图 7 所示, 点击业务类型右边的星型按钮可以添加新表, 点击字段后的向下箭头按钮可以让用户添加新字段. 从而实现了表和字段的自由添加, 在申请之初为用户提供方便. 由于业务表和自由流程之间的保持相对独立, 这种设计非常有利于开展新业务, 并使之进行业务流转.

5 结语

本文的主要贡献如下:

(1) 本文充分利用了流程框架 Activiti 和非关系型数据库 MongoDB 的优点, 提出了适用于大型单位后勤管理的自由流程信息系统设计流程框架、系统重要接口、关键逻辑代码和两类数据库结合的数据结构协同设计方法.

(2) 本文设计的系统支持多个流程同时并行进行, 或者多个流程串行运行, 支持针对不同的业务类型根据需求自由添加表和字段.

实现效果显示本文设计的自由流程管理系统实用、灵活和高效, 最大限度地支持业务工作和减少研发人员的工作量, 对于单位后勤管理信息化具有较好的设计参考价值.

图 7 表和字段任意添加

参考文献

1 Activiti 项目. https://www.activiti.org/. [2019-06-01].

2 李修云. 基于 Activiti 框架的在线审批流程应用研究. 计算机科学, 2016, 43(6A): 555–557. [doi: 10.11896/j.issn.1002-137X.2016.6A.132]

3 梁继刚, 郭凌, 刘凌. 基于 SOA 的数字后勤日常业务信息系统设计. 四川兵工学报, 2015, 36(4): 92–95.

4 张毅, 孙加光, 喻波, 等. 一种基于 Activiti 的流程创建部署方法及系统: 201810430062.4. [2018-10-19].

5 朱爱华, 付曹政, 曹钟, 等. 基于 Node.js 框架和 MongoDB 数据库的物流信息服务系统设计. 北京建筑大学学报, 2018, 34(4): 41–46.

6 Holemans A, Kasprzyk JP, Donnay JP. Coupling an unstructured NoSQL database with a geographic information system. Proceedings of the 10th International Conference on Advanced Geographic Information Systems, Applications, and Services GEOProcessing 2018. Rome, Italy. 2018. 23–28.

7 于晓芸. 民远学院后勤管理系统设计与实现[硕士学位论文]. 成都: 电子科技大学, 2015.

8 Casanova H, Pandey S, Oeth J, *et al*. WRENCH: A framework for simulating workflow management systems. Proceedings of 2018 IEEE/ACM Workflows in Support of Large-Scale Science. Dallas, TX, USA. 2018. 74–85.

9 牟杨. 大数据在后勤库存物资智能化管理系统中的应用. 自动化与仪器仪表, 2019, (4): 233–236.

10 潘明明, 李丁丁, 汤庸, 等. 一种基于中间件的异构数据库融合访问方法及系统. 计算机科学, 2018, 45(5): 163–167.