石化行业工艺文件审批流程管理系统

源代码

北京杰迅鸿翔信息技术有限公司

package com.jiexunnet.workflow.base.model;

import java.io.Serializable;

import java.util.Date;

import org.apache.commons.lang3.builder.ToStringBuilder;

public abstract class BaseWorkflowModel implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = -668713726822987006L;

public abstract String getUuid();

public abstract void setUuid(String uuid);

public abstract String getProccessInstUuid();

public abstract void setProccessInstUuid(String proccessInstUuid);

public abstract String getActivityInstUuid();

public abstract String getDj();

public abstract String getMj();

public abstract String getBt();

public abstract String getYzh();

public abstract void setActivityInstUuid(String activityInstUuid);

public abstract BaseWorkflowModel copyModel();

@Override

public String toString() {

return ToStringBuilder.reflectionToString(this);

}

}

package com.jiexunnet.workflow.base.model;

public class RouteLog {

private String activityInstUuid;

private String processsInstUuid;

private String personId;

private String personName;

private String beginTime;

private String endTime;

private String status;

private String arrivedTime;

private String activityName;

private String personDeptName;

private String backReson;

public RouteLog(){

}

public String getActivityInstUuid() {

return activityInstUuid;

}

public void setActivityInstUuid(String activityInstUuid) {

this.activityInstUuid = activityInstUuid;

}

public String getProcesssInstUuid() {

return processsInstUuid;

}

public void setProcesssInstUuid(String processsInstUuid) {

this.processsInstUuid = processsInstUuid;

}

public String getPersonId() {

return personId;

}

public void setPersonId( String personId) {

this.personId = personId;

}

public String getPersonName() {

return personName;

}

public void setPersonName(String personName) {

this.personName = personName;

}

public String getBeginTime() {

return beginTime;

}

public void setBeginTime(String beginTime) {

this.beginTime = beginTime;

}

public String getEndTime() {

return endTime;

}

public void setEndTime(String endTime) {

this.endTime = endTime;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public void setArrivedTime(String arrivedTime) {

this.arrivedTime = arrivedTime;

}

public String getArrivedTime() {

return arrivedTime;

}

public void setActivityName(String activityName) {

this.activityName = activityName;

}

public String getActivityName() {

return activityName;

}

public String getPersonDeptName() {

return personDeptName;

}

public void setPersonDeptName(String personDeptName) {

this.personDeptName = personDeptName;

}

public String getBackReson() {

return backReson;

}

public void setBackReson(String backReson) {

this.backReson = backReson;

}

}

package com.jiexunnet.workflow.controller;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.BPMActivityDef;

import com.jiexunnet.workflow.service.IActivityDefService;

@Controller

public class ActivityDefController {

@Autowired

private IActivityDefService activityDefService;

@RequestMapping(value="/workflowdefine/activitydef/list.do")

public @ResponseBody Map<String, Object> list(String versionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

List<BPMActivityDef> lists = this.activityDefService.getAllByVersionUuid(versionUuid);

retMap.put("success", true);

retMap.put("data", lists);

return retMap;

}

@RequestMapping(value="/workflowdefine/activitydef/add.do")

public @ResponseBody Map<String, Object> add(@RequestBody BPMActivityDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

this.activityDefService.save(def);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/activitydef/del.do")

public @ResponseBody Map<String, Object> del(@RequestBody BPMActivityDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

this.activityDefService.deleteActivity(def.getUuid());

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/activitydef/update.do")

public @ResponseBody Map<String, Object> update(@RequestBody BPMActivityDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

this.activityDefService.updateActivity(def);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/activitydef/get.do")

public @ResponseBody BPMActivityDef get(String uuid){

return this.activityDefService.get(uuid);

}

public IActivityDefService getActivityDefService() {

return activityDefService;

}

public void setActivityDefService(IActivityDefService activityDefService) {

this.activityDefService = activityDefService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.WebDataBinder;

import org.springframework.web.bind.annotation.InitBinder;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.common.web.support.editor.DateTimeEditor;

import com.jiexunnet.workflow.model.BpmActivityEvent;

import com.jiexunnet.workflow.service.IBPMActivityEventService;

@Controller("BpmActivityEventController")

public class BpmActivityEventController {

@Autowired

@Qualifier("IBPMActivityEventService")

private IBPMActivityEventService activityEventService;

@RequestMapping(value="/workflow/bpmActivityEvent/findByActivitydefUuid.do")

public @ResponseBody Map<String, Object> findByActivitydefUuid(String activitydefUuid,int start,int limit){

Map<String, Object> retMap = new HashMap<String, Object>();

List<BpmActivityEvent> bpmActivityListenerlist = this.activityEventService.findByActivitydefUuid(activitydefUuid);

retMap.put("success", true);

retMap.put("data", bpmActivityListenerlist);

return retMap;

}

@SuppressWarnings("unused")

private Map<String, Object> beanTomap(BpmActivityEvent event){

Map<String, Object> map = new HashMap<String, Object>();

map.put("uuid", event.getUuid());

map.put("activitydefUuid", event.getActivitydefUuid());

map.put("eventName", event.getEventName());

map.put("eventClasses", event.getEventClasses());

map.put("eventFlag", event.getEventFlag());

map.put("eventFormula", event.getEventFormula());

map.put("eventOrder", event.getEventOrder());

map.put("proccessdefUuid()", event.getProccessdefUuid());

return map;

}

@RequestMapping(value="/workflow/bpmActivityEvent/add.do")

public @ResponseBody Map<String, Object> add(@RequestBody BpmActivityEvent event){

Map<String, Object> retMap = new HashMap<String, Object>();

this.activityEventService.save(event);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmActivityEvent/update.do")

public @ResponseBody Map<String,Object> update(@RequestBody BpmActivityEvent event){

Map<String, Object> map = new HashMap<String, Object>();

BpmActivityEvent bevent = new BpmActivityEvent();

bevent.setActivitydefUuid(event.getActivitydefUuid());

bevent.setEventClasses(event.getEventClasses());

bevent.setEventFlag(event.getEventFlag());

bevent.setEventFormula(event.getEventFormula());

bevent.setEventName(event.getEventName());

bevent.setEventOrder(event.getEventOrder());

bevent.setProccessdefUuid(event.getProccessdefUuid());

bevent.setUuid(event.getUuid());

this.activityEventService.update(bevent);

map.put("success", true);

return map;

}

@RequestMapping(value="/workflow/bpmActivityEvent/delete.do")

public @ResponseBody Map<String,Object> delete(String activityEventIds){

Map<String, Object> map = new HashMap<String, Object>();

try {

this.activityEventService.delActivityListeners(activityEventIds);

} catch (Exception e) {

e.printStackTrace();

}

map.put("success", true);

return map;

}

@InitBinder

public void initBinder(WebDataBinder binder) {

binder.registerCustomEditor(Date.class, new DateTimeEditor());

}

public IBPMActivityEventService getActivityEventService() {

return activityEventService;

}

public void setActivityEventService(

IBPMActivityEventService activityEventService) {

this.activityEventService = activityEventService;

}

}

package com.jiexunnet.workflow.controller;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.WebDataBinder;

import org.springframework.web.bind.annotation.InitBinder;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.common.web.support.editor.DateTimeEditor;

import com.jiexunnet.workflow.model.BpmActivityListener;

import com.jiexunnet.workflow.service.IBpmActivityListenerService;

@Controller

public class BpmActivityListenerController {

@Autowired

private IBpmActivityListenerService activityListenerService;

@RequestMapping(value="/workflow/bpmActivityListener/findByActivitydefUuid.do")

public @ResponseBody Map<String, Object> findByActivitydefUuid(String activitydefUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

@SuppressWarnings("unused")

List<Map<String, Object>> maplist = new ArrayList<Map<String, Object>>();

List<BpmActivityListener> bpmActivityListenerlist = this.activityListenerService.findByActivitydefUuid(activitydefUuid);

retMap.put("success", true);

retMap.put("data", bpmActivityListenerlist);

return retMap;

}

@SuppressWarnings("unused")

private Map<String, Object> beanTomap(BpmActivityListener listener){

Map<String, Object> map = new HashMap<String, Object>();

map.put("uuid", listener.getUuid());

map.put("activitydefUuid", listener.getActivitydefUuid());

map.put("listenerName", listener.getListenerName());

map.put("listenerClass", listener.getListenerClass());

map.put("listenerType", listener.getListenerType());

map.put("listenerformula", listener.getListenerformula());

map.put("listenerOrder", listener.getListenerOrder());

if(listener.getCreateDate()!=null){

map.put("createDate", new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(listener.getCreateDate()));

}else{

map.put("createDate","");

}

return map;

}

@RequestMapping(value="/workflow/bpmActivityListener/add.do")

public @ResponseBody Map<String, Object> add(@RequestBody BpmActivityListener listener){

Map<String, Object> retMap = new HashMap<String, Object>();

listener.setCreateDate(new Date());

this.activityListenerService.save(listener);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmActivityListener/update.do")

public @ResponseBody Map<String,Object> update(@RequestBody BpmActivityListener listener){

Map<String, Object> map = new HashMap<String, Object>();

BpmActivityListener bal = new BpmActivityListener();

bal.setActivitydefUuid(listener.getActivitydefUuid());

bal.setCreateDate(listener.getCreateDate());

bal.setListenerClass(listener.getListenerClass());

bal.setListenerformula(listener.getListenerformula());

bal.setListenerName(listener.getListenerName());

bal.setListenerOrder(listener.getListenerOrder());

bal.setListenerType(listener.getListenerType());

bal.setUuid(listener.getUuid());

/\*Date createDate = this.activityListenerService.get(listener.getUuid()).getCreateDate();

listener.setCreateDate(createDate);\*/

this.activityListenerService.update(bal);

map.put("success", true);

return map;

}

@RequestMapping(value="/workflow/bpmActivityListener/delete.do")

public @ResponseBody Map<String,Object> delete(String activityListenerIds){

Map<String, Object> map = new HashMap<String, Object>();

try {

this.activityListenerService.delActivityListeners(activityListenerIds);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

map.put("success", true);

return map;

}

@InitBinder

public void initBinder(WebDataBinder binder) {

binder.registerCustomEditor(Date.class, new DateTimeEditor());

}

public IBpmActivityListenerService getActivityListenerService() {

return activityListenerService;

}

public void setActivityListenerService(

IBpmActivityListenerService activityListenerService) {

this.activityListenerService = activityListenerService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.HashMap;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.BpmActivityPrivilege;

import com.jiexunnet.workflow.service.IBpmActivityPrivilegeService;

@Controller

public class BpmActivityPrivilegeController {

@Autowired

private IBpmActivityPrivilegeService activityPrivilegeService;

@RequestMapping(value="/workflow/bpmActivityPrivilege/findByActivitydefUuid.do")

public @ResponseBody Map<String, Object> findByActivitydefUuid(String activitydefUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

BpmActivityPrivilege bpmActivityPrivilege = this.activityPrivilegeService.findByActivitydefUuid(activitydefUuid);

retMap.put("success", true);

retMap.put("data", bpmActivityPrivilege);

return retMap;

}

public IBpmActivityPrivilegeService getActivityPrivilegeService() {

return activityPrivilegeService;

}

public void setActivityPrivilegeService(

IBpmActivityPrivilegeService activityPrivilegeService) {

this.activityPrivilegeService = activityPrivilegeService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.BpmActivityPerformer;

import com.jiexunnet.workflow.model.BpmActivityReader;

import com.jiexunnet.workflow.model.BpmParticipant;

import com.jiexunnet.workflow.service.IBpmActivityPerformerService;

import com.jiexunnet.workflow.service.IBpmActivityReaderService;

import com.jiexunnet.workflow.service.IBpmParticipantService;

@Controller

public class BpmParticipantController {

@Autowired

private IBpmParticipantService bpmParticipantService;

@Autowired

private IBpmActivityPerformerService activityPerformerService;

@Autowired

private IBpmActivityReaderService activityReaderService;

@RequestMapping(value="/workflow/bpmParticipant/updateReaderQX.do")

public @ResponseBody Map<String, Object> updateReaderQX(String uuid, String ids, String names,

String participantSelectUuid, String activitydefUuid, String proccessDefUuid, String proccessDefVersionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

BpmActivityReader reader = new BpmActivityReader();

reader.setUuid(uuid);

reader.setActivitydefUuid(activitydefUuid);

reader.setParticipantSelectParam(ids);

reader.setParticipantSelectParamName(names);

reader.setParticipantSelectUuid(participantSelectUuid);

reader.setProccessdefUuid(proccessDefUuid);

reader.setProccessdefVersionUuid(proccessDefVersionUuid);

this.activityReaderService.update(reader);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmParticipant/updatePerformerQX.do")

public @ResponseBody Map<String, Object> updatePerformerQX(String uuid, String ids, String names,

String participantSelectUuid, String activitydefUuid, String proccessDefUuid, String proccessDefVersionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

BpmActivityPerformer performer = new BpmActivityPerformer();

performer.setUuid(uuid);

performer.setParticipantSelectParam(ids);

performer.setParticipantSelectParamName(names);

performer.setParticipantSelectUuid(participantSelectUuid);

performer.setActivitydefUuid(activitydefUuid);

performer.setProccessdefUuid(proccessDefUuid);

performer.setProccessdefVersionUuid(proccessDefVersionUuid);

this.activityPerformerService.update(performer);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmParticipant/addReaderQX.do")

public @ResponseBody Map<String, Object> addReaderUsers(String ids, String names,

String participantSelectUuid, String activitydefUuid,

String proccessDefUuid, String proccessDefVersionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

BpmActivityReader reader = new BpmActivityReader();

reader.setActivitydefUuid(activitydefUuid);

reader.setParticipantSelectParam(ids);

reader.setParticipantSelectParamName(names);

reader.setParticipantSelectUuid(participantSelectUuid);

reader.setProccessdefUuid(proccessDefUuid);

reader.setProccessdefVersionUuid(proccessDefVersionUuid);

this.activityReaderService.save(reader);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmParticipant/addQX.do")

public @ResponseBody Map<String, Object> addUsers(String ids, String names,

String participantSelectUuid, String activitydefUuid,

String proccessDefUuid, String proccessDefVersionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

BpmActivityPerformer performer = new BpmActivityPerformer();

performer.setActivitydefUuid(activitydefUuid);

performer.setParticipantSelectParam(ids);

performer.setParticipantSelectParamName(names);

performer.setParticipantSelectUuid(participantSelectUuid);

performer.setProccessdefUuid(proccessDefUuid);

performer.setProccessdefVersionUuid(proccessDefVersionUuid);

this.activityPerformerService.save(performer);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmParticipant/listAll.do")

public @ResponseBody Map<String, Object> findByActivitydefUuid(){

Map<String, Object> retMap = new HashMap<String, Object>();

List<BpmParticipant> list = this.bpmParticipantService.listAll();

retMap.put("success", true);

retMap.put("data", list);

return retMap;

}

public IBpmParticipantService getBpmParticipantService() {

return bpmParticipantService;

}

public void setBpmParticipantService(

IBpmParticipantService bpmParticipantService) {

this.bpmParticipantService = bpmParticipantService;

}

public IBpmActivityPerformerService getActivityPerformerService() {

return activityPerformerService;

}

public void setActivityPerformerService(

IBpmActivityPerformerService activityPerformerService) {

this.activityPerformerService = activityPerformerService;

}

public IBpmActivityReaderService getActivityReaderService() {

return activityReaderService;

}

public void setActivityReaderService(

IBpmActivityReaderService activityReaderService) {

this.activityReaderService = activityReaderService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.WebDataBinder;

import org.springframework.web.bind.annotation.InitBinder;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.common.web.support.editor.DateTimeEditor;

import com.jiexunnet.workflow.model.BpmActivityEvent;

import com.jiexunnet.workflow.model.BpmProccessEvent;

import com.jiexunnet.workflow.service.IBpmProcessEventService;

@Controller("BpmProcessEventController")

public class BpmProcessEventController {

@Autowired

@Qualifier("IBpmProcessEventService")

private IBpmProcessEventService eventService;

@RequestMapping(value="/workflow/bpmProcessEvent/findByProcessdefUuid.do")

public @ResponseBody Map<String, Object> findByProcessdefUuid(String ProcessdefUuid,int start,int limit){

Map<String, Object> retMap = new HashMap<String, Object>();

List<BpmProccessEvent> bpmProcessEventlist = this.eventService.findAllEventByProccessdefUuid(ProcessdefUuid);

retMap.put("success", true);

retMap.put("data", bpmProcessEventlist);

return retMap;

}

@SuppressWarnings("unused")

private Map<String, Object> beanTomap(BpmProccessEvent event){

Map<String, Object> map = new HashMap<String, Object>();

map.put("uuid", event.getUuid());

map.put("proccessdefUuid", event.getProccessdefUuid());

map.put("eventName", event.getEventName());

map.put("eventClasses", event.getEventClasses());

map.put("eventFlag", event.getEventFlag());

map.put("eventFormula", event.getEventFormula());

map.put("eventOrder", event.getEventOrder());

return map;

}

@RequestMapping(value="/workflow/bpmProcessEvent/add.do")

public @ResponseBody Map<String, Object> add(@RequestBody BpmProccessEvent event){

Map<String, Object> retMap = new HashMap<String, Object>();

this.eventService.save(event);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmProcessEvent/update.do")

public @ResponseBody Map<String,Object> update(@RequestBody BpmProccessEvent event){

Map<String, Object> map = new HashMap<String, Object>();

BpmActivityEvent bevent = new BpmActivityEvent();

bevent.setEventClasses(event.getEventClasses());

bevent.setEventFlag(event.getEventFlag());

bevent.setEventFormula(event.getEventFormula());

bevent.setEventName(event.getEventName());

bevent.setEventOrder(event.getEventOrder());

bevent.setProccessdefUuid(event.getProccessdefUuid());

bevent.setUuid(event.getUuid());

this.eventService.update(event);

map.put("success", true);

return map;

}

@RequestMapping(value="/workflow/bpmProcessEvent/delete.do")

public @ResponseBody Map<String,Object> delete(String eventIds){

Map<String, Object> map = new HashMap<String, Object>();

try {

this.eventService.deleteByProccessdefUuid(eventIds);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

map.put("success", true);

return map;

}

@InitBinder

public void initBinder(WebDataBinder binder) {

binder.registerCustomEditor(Date.class, new DateTimeEditor());

}

public IBpmProcessEventService getEventService() {

return eventService;

}

public void setEventService(IBpmProcessEventService eventService) {

this.eventService = eventService;

}

}

package com.jiexunnet.workflow.controller;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.ProcessListener;

import com.jiexunnet.workflow.service.IProcessListenerService;

@Controller

public class BpmProcessListenerController {

@Autowired

private IProcessListenerService processListenerService;

@RequestMapping(value="/workflow/bpmProcessListener/findByVersionUuid.do")

public @ResponseBody Map<String, Object> findByVersionUuid(String proccessDefVersionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

List<Map<String, Object>> maplist = new ArrayList<Map<String, Object>>();

List<ProcessListener> bpmProcessListenerlist = this.processListenerService.findByProcessVersion(proccessDefVersionUuid);

for(int i=0;i<bpmProcessListenerlist.size();i++){

maplist.add(beanTomap(bpmProcessListenerlist.get(i)));

}

retMap.put("success", true);

retMap.put("data", bpmProcessListenerlist);

return retMap;

}

private Map<String, Object> beanTomap(ProcessListener listener){

Map<String, Object> map = new HashMap<String, Object>();

map.put("uuid", listener.getUuid());

map.put("activitydefUuid", listener.getProcessVersion());

map.put("listenerName", listener.getListenerName());

map.put("listenerClass", listener.getListenerClass());

if(listener.getListenerType()==0){

map.put("listenerType", "实体类class");

}else{

map.put("listenerType", "表达式");

}

map.put("listenerFormula", listener.getListenerFormula());

map.put("listenerOrder", listener.getListenerOrder());

if(listener.getCreateDate()!=null){

map.put("createDate", new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(listener.getCreateDate()));

}else{

map.put("createDate","");

}

return map;

}

@RequestMapping(value="/workflow/bpmProcessListener/add.do")

public @ResponseBody Map<String, Object> add(@RequestBody ProcessListener listener){

Map<String, Object> retMap = new HashMap<String, Object>();

listener.setCreateDate(new Date());

this.processListenerService.save(listener);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflow/bpmProcessListener/update.do")

public @ResponseBody Map<String,Object> update(@RequestBody ProcessListener listener){

Map<String, Object> map = new HashMap<String, Object>();

ProcessListener bal = new ProcessListener();

bal.setProcessVersion(listener.getProcessVersion());

bal.setCreateDate(listener.getCreateDate());

bal.setListenerClass(listener.getListenerClass());

bal.setListenerFormula(listener.getListenerFormula());

bal.setListenerName(listener.getListenerName());

bal.setListenerOrder(listener.getListenerOrder());

bal.setListenerType(listener.getListenerType());

bal.setUuid(listener.getUuid());

this.processListenerService.update(bal);

map.put("success", true);

return map;

}

@RequestMapping(value="/workflow/bpmProcessListener/delete.do")

public @ResponseBody Map<String,Object> delete(String ListenerIds){

Map<String, Object> map = new HashMap<String, Object>();

try {

this.processListenerService.delProcessListeners(ListenerIds);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

map.put("success", true);

return map;

}

public IProcessListenerService getProcessListenerService() {

return processListenerService;

}

public void setProcessListenerService(

IProcessListenerService processListenerService) {

this.processListenerService = processListenerService;

}

}

package com.jiexunnet.workflow.controller;

import java.net.URL;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

import net.sf.ehcache.Cache;

import net.sf.ehcache.CacheManager;

import net.sf.ehcache.Element;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

import com.jiexunnet.common.service.ISmsService;

import com.jiexunnet.common.web.filter.SessionFilter;

import com.jiexunnet.usm.model.TDept;

import com.jiexunnet.usm.model.TUser;

import com.jiexunnet.usm.service.IUsmService;

@Controller

public class LoginWorkflowController {

@Autowired

@Qualifier("IUsmService")

private IUsmService usmService;

@Autowired

@Qualifier("ISmsService")

private ISmsService smsService;

@RequestMapping(value = "/workflow/login")

public String index(HttpServletRequest request) {

return "/workflow/login";

}

@RequestMapping(value="/workflow/checkLogin.do")

public ModelAndView login(String loginName, String password, HttpSession session, HttpServletRequest request){

TUser user = this.usmService.login(loginName, password);

if(user == null){

ModelAndView mav = new ModelAndView("/oa/error");

return mav;

}

URL url = getClass().getResource("/ehcache\_login.xml");

CacheManager cacheManager = CacheManager.create(url);

Cache cache = cacheManager.getCache("login");

Element element1 = cache.get(loginName);

if(element1 != null)

cache.remove(loginName);

Element element = new Element(loginName, session.getId());

cache.put(element);

List<TDept> depts = this.usmService.getUserDepts(user);

session.setAttribute("user", user);

session.setAttribute("depts", depts);

session.setAttribute("userName", loginName);

//验证单点登陆

SessionFilter sessionFilter = new SessionFilter();

sessionFilter.putSessionMap(session, user);

ModelAndView mav = new ModelAndView("/workflow/index");

return mav;

}

//将时间转换成前台要的格式

public String timeZhuanHuan(Date createDate){

SimpleDateFormat formater = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return formater.format(createDate);

}

}

package com.jiexunnet.workflow.controller;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.BpmActivityPerformer;

import com.jiexunnet.workflow.model.BpmParticipant;

import com.jiexunnet.workflow.service.IBpmActivityPerformerService;

import com.jiexunnet.workflow.service.IBpmParticipantService;

@Controller

public class PerformerSelectController {

@Autowired

private IBpmActivityPerformerService performerService ;

@Autowired

private IBpmParticipantService PerticipantService ;

@RequestMapping(value="/workflowdefine/performer/list.do")

public @ResponseBody Map<String, Object> findByActivitydefUuid(String activitydefId){

Map<String, Object> Map = new HashMap<String, Object>();

List<Map<String, Object>> acdefs = new ArrayList<Map<String, Object>>();

@SuppressWarnings("unchecked")

List<Object[]> performerslist = this.performerService.getBpmActivityPerformers(activitydefId);

if(performerslist.size()>0){

for(int i=0;i<performerslist.size();i++){

Object[] objects = performerslist.get(i);

acdefs.add(bpmpantToMap(objects)) ;

}

}

//for(int i=0;i<performerslist.size();i++){

// BpmParticipant bpmppant= PerticipantService.getBpmParticipant(performerslist.get(i).getParticipantSelectUuid());

// }

Map.put("success", true);

Map.put("data", acdefs);

return Map;

}

@RequestMapping(value="/workflowdefine/performer/del.do")

public @ResponseBody Map<String, Object> del(String performerIds){

Map<String, Object> retMap = new HashMap<String, Object>();

this.performerService.delPerformers(performerIds);

retMap.put("success", true);

return retMap;

}

private Map<String, Object> bpmpantToMap(Object[] object){

BpmParticipant bp = (BpmParticipant)object[0];

BpmActivityPerformer bp1 = (BpmActivityPerformer)object[1];

Map<String, Object> map = new HashMap<String, Object>();

map.put("uuid", bp1.getUuid());

map.put("participantName", bp.getParticipantName());

map.put("participantDesc", bp.getParticipantDesc());

map.put("participantSelectUuid", bp1.getParticipantSelectUuid());

map.put("participantSelectParamName", bp1.getParticipantSelectParamName());

map.put("participantSelectParam", bp1.getParticipantSelectParam());

return map;

}

public IBpmActivityPerformerService getPerformerService() {

return performerService;

}

public void setPerformerService(IBpmActivityPerformerService performerService) {

this.performerService = performerService;

}

public IBpmParticipantService getPerticipantService() {

return PerticipantService;

}

public void setPerticipantService(IBpmParticipantService perticipantService) {

PerticipantService = perticipantService;

}

}

package com.jiexunnet.workflow.controller;

import java.beans.PropertyEditorSupport;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.xml.bind.JAXBException;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.WebDataBinder;

import org.springframework.web.bind.annotation.InitBinder;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.controller.vo.WorkflowDefVo;

import com.jiexunnet.workflow.model.ActivityListener;

import com.jiexunnet.workflow.model.BPMActivityDef;

import com.jiexunnet.workflow.model.BPMActivityDefXMLObject;

import com.jiexunnet.workflow.model.BPMProccessDef;

import com.jiexunnet.workflow.model.BPMProccessDefVersion;

import com.jiexunnet.workflow.model.BPMProcessXMLObject;

import com.jiexunnet.workflow.model.BPMProcessXMLObjectImport;

import com.jiexunnet.workflow.model.BPMRouteDef;

import com.jiexunnet.workflow.model.BpmActivityPerformer;

import com.jiexunnet.workflow.model.BpmActivityPrivilege;

import com.jiexunnet.workflow.model.BpmActivityReader;

import com.jiexunnet.workflow.model.ProcessListener;

import com.jiexunnet.workflow.service.IActivityDefService;

import com.jiexunnet.workflow.service.IActivityListenerService;

import com.jiexunnet.workflow.service.IBpmActivityPerformerService;

import com.jiexunnet.workflow.service.IBpmActivityPrivilegeService;

import com.jiexunnet.workflow.service.IBpmActivityReaderService;

import com.jiexunnet.workflow.service.IProccessDefService;

import com.jiexunnet.workflow.service.IProccessDefVersionService;

import com.jiexunnet.workflow.service.IProcessListenerService;

import com.jiexunnet.workflow.service.IRouteDefService;

@Controller

public class ProccessDefController {

@Autowired

private IProccessDefService proccessDefService;

@Autowired

private IProccessDefVersionService proccessDefVersionService;

@Autowired

private IActivityDefService activityDefService;

@Autowired

private IRouteDefService routeDefService;

@Autowired

private IBpmActivityPrivilegeService activityPrivilegeService;

@Autowired

private IBpmActivityPerformerService activityPerformerService;

@Autowired

private IBpmActivityReaderService activityReaderService;

@Autowired

private IProcessListenerService processListenerService;

@Autowired

private IActivityListenerService activityListenerService;

public IActivityDefService getActivityDefService() {

return activityDefService;

}

public void setActivityDefService(IActivityDefService activityDefService) {

this.activityDefService = activityDefService;

}

public IRouteDefService getRouteDefService() {

return routeDefService;

}

public void setRouteDefService(IRouteDefService routeDefService) {

this.routeDefService = routeDefService;

}

public IBpmActivityPrivilegeService getActivityPrivilegeService() {

return activityPrivilegeService;

}

public void setActivityPrivilegeService(

IBpmActivityPrivilegeService activityPrivilegeService) {

this.activityPrivilegeService = activityPrivilegeService;

}

public IBpmActivityPerformerService getActivityPerformerService() {

return activityPerformerService;

}

public void setActivityPerformerService(

IBpmActivityPerformerService activityPerformerService) {

this.activityPerformerService = activityPerformerService;

}

public IBpmActivityReaderService getActivityReaderService() {

return activityReaderService;

}

public void setActivityReaderService(

IBpmActivityReaderService activityReaderService) {

this.activityReaderService = activityReaderService;

}

public IProcessListenerService getProcessListenerService() {

return processListenerService;

}

public void setProcessListenerService(

IProcessListenerService processListenerService) {

this.processListenerService = processListenerService;

}

public IActivityListenerService getActivityListenerService() {

return activityListenerService;

}

public void setActivityListenerService(

IActivityListenerService activityListenerService) {

this.activityListenerService = activityListenerService;

}

@RequestMapping(value="/workflowdefine/proccessdef/tree.do")

public @ResponseBody Map<String, Object> tree(){

Map<String, Object> retMap = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

List<BPMProccessDef> proccessDefs = this.proccessDefService.listAll();

for(int i=0; i<proccessDefs.size(); i++){

lists.add(proccessChildren(proccessDefs.get(i)));

}

retMap.put("success", true);

retMap.put("children", lists);

return retMap;

}

private Map<String, Object> proccessChildren(BPMProccessDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

retMap.put("id", def.getUuid());

retMap.put("text", def.getProccessDefName());

retMap.put("loaded", true);

retMap.put("leaf", true);

retMap.put("expanded", true);

retMap.put("version", "");

retMap.put("status", "");

retMap.put("versiontype", "proccessdef");

List<BPMProccessDefVersion> versions = this.proccessDefVersionService.getAllByProccessDefUuid(def.getUuid());

if(versions!=null && versions.size()>0){

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

for(int i=0; i<versions.size(); i++){

BPMProccessDefVersion version = versions.get(i);

Map<String, Object> map = new HashMap<String, Object>();

map.put("id", version.getUuid());

String status = "";

if(version.getPublicStatus().equals(BPMProccessDefVersion.PUBLIC\_STATUS\_RELEASED))

status = "(激活)";

else if(BPMProccessDefVersion.PUBLIC\_STATUS\_TEST.equals(version.getPublicStatus()))

status ="(测试)";

else if(BPMProccessDefVersion.PUBLIC\_STATUS\_FREEZ.equals(version.getPublicStatus()))

status = "(冻结)";

else

status = "(修订中)";

map.put("text", "版本" + new Integer(version.getProccessDefVersion()+1).toString() + status);

map.put("loaded", true);

map.put("leaf", true);

map.put("expanded", true);

map.put("version", version.getProccessDefVersion());

map.put("status", version.getPublicStatus());

map.put("versiontype", "proccessdefversion");

lists.add(map);

}

retMap.put("leaf", false);

retMap.put("children", lists);

}

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/create.do")

public @ResponseBody Map<String, Object> newProccess(@RequestBody WorkflowDefVo vo){

Map<String, Object> retMap = new HashMap<String, Object>();

BPMProccessDef def = vo.toBPMProccessDef();

BPMProccessDefVersion ver = vo.toBPMProccessDefVersion();

ver.setActiveTime(new Date());

this.proccessDefService.newProccessDef(def, ver);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/delete.do")

public @ResponseBody Map<String, Object> delProccess(@RequestBody WorkflowDefVo vo){

Map<String, Object> retMap = new HashMap<String, Object>();

this.proccessDefService.delProccessDef(vo.getProccessuuid());

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/update.do")

public @ResponseBody Map<String, Object> updateProccess(@RequestBody WorkflowDefVo vo){

Map<String, Object> retMap = new HashMap<String, Object>();

BPMProccessDef def = vo.toBPMProccessDef();

this.proccessDefService.update(def);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/get.do")

public @ResponseBody WorkflowDefVo getProccess(String uuid, String versionuuid){

WorkflowDefVo vo = new WorkflowDefVo();

BPMProccessDef def = this.proccessDefService.get(uuid);

BPMProccessDefVersion version = null;

if(versionuuid==null || versionuuid.length()==0){

version = this.proccessDefVersionService.getActiveProccessDefVersionByDefUuid(uuid);

}

else{

version = this.proccessDefVersionService.get(versionuuid);

}

if(def != null && version != null)

vo.copyData(def, version);

return vo;

}

@RequestMapping(value="/workflowdefine/proccessdef/createversion.do")

public @ResponseBody Map<String, Object> createProccessVersion(@RequestBody WorkflowDefVo vo){

Map<String, Object> retMap = new HashMap<String, Object>();

@SuppressWarnings("unused")

BPMProccessDef def = vo.toBPMProccessDef();

BPMProccessDefVersion ver = vo.toBPMProccessDefVersion();

ver.setUuid(null);

this.proccessDefVersionService.addVersion(ver);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/updateversion.do")

public @ResponseBody Map<String, Object> upateProccessVersion(@RequestBody WorkflowDefVo vo){

Map<String, Object> retMap = new HashMap<String, Object>();

@SuppressWarnings("unused")

BPMProccessDef def = vo.toBPMProccessDef();

BPMProccessDefVersion ver = vo.toBPMProccessDefVersion();

this.proccessDefVersionService.update(ver);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/delversion.do")

public @ResponseBody Map<String, Object> delProccessVersion(@RequestBody WorkflowDefVo vo){

Map<String, Object> retMap = new HashMap<String, Object>();

this.proccessDefVersionService.delVersion(vo.getId());

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/freez.do")

public @ResponseBody Map<String, Object> delProccessVersion(String uuid){

Map<String, Object> retMap = new HashMap<String, Object>();

this.proccessDefVersionService.freeez(uuid);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/active.do")

public @ResponseBody Map<String, Object> activeProccessVersion(String uuid){

Map<String, Object> retMap = new HashMap<String, Object>();

this.proccessDefVersionService.active(uuid);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/importproccess")

public @ResponseBody Map<String, Object> importProccess(String xmlstr){

Map<String, Object> retMap = new HashMap<String, Object>();

try {

BPMProcessXMLObjectImport object = BPMProcessXMLObjectImport.decodeXml(xmlstr);

this.proccessDefService.importProccess(object);

} catch (JAXBException e) {

e.printStackTrace();

retMap.put("success", false);

return retMap;

}

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/proccessdef/exportproccess")

public @ResponseBody Map<String, Object> exprotProccess(String uuid){

Map<String, Object> retMap = new HashMap<String, Object>();

BPMProcessXMLObject object = new BPMProcessXMLObject();

BPMProccessDef def = this.proccessDefService.get(uuid);

object.setProcessDef(def);

List<BPMProccessDefVersion> processDefVersions = this.proccessDefVersionService.getAllByProccessDefUuid(uuid);

object.getProcessDefVersions().addAll(processDefVersions);

for(int i=0; i<processDefVersions.size(); i++){

BPMProccessDefVersion version = processDefVersions.get(i);

List<BPMActivityDef> activityDefs = this.activityDefService.getAllByVersionUuid(version.getUuid());

List<BPMRouteDef> routeDefs = this.routeDefService.listchildren(version.getUuid());

object.getRouteDefs().addAll(routeDefs);

List<ProcessListener> processListeners = this.processListenerService.findByProcessVersion(version.getUuid());

object.getProcessListeners().addAll(processListeners);

for(int j=0; j<activityDefs.size(); j++){

BPMActivityDef activityDef = activityDefs.get(j);

BpmActivityPrivilege privilege = this.activityPrivilegeService.findByActivitydefUuid(activityDef.getUuid());

BPMActivityDefXMLObject activityObj = new BPMActivityDefXMLObject();

activityObj.setActivityDef(activityDef);

activityObj.setActivityPrivilege(privilege);

List<BpmActivityPerformer> performers = this.activityPerformerService.findByActivityDefUuid(activityDef.getUuid());

activityObj.getActivityPerformers().addAll(performers);

List<BpmActivityReader> readers = this.activityReaderService.findByActivityDefUuid(activityDef.getUuid());

activityObj.getActivityReaders().addAll(readers);

List<ActivityListener> activityListeners = this.activityListenerService.findByActivityDefUuid(activityDef.getUuid());

activityObj.getActivityListners().addAll(activityListeners);

object.getBpmActivityDefXMLObjects().add(activityObj);

}

}

String xmlStr;

try {

xmlStr = object.toXml();

} catch (JAXBException e) {

e.printStackTrace();

retMap.put("success", false);

return retMap;

}

retMap.put("success", true);

retMap.put("xmlstr", xmlStr);

return retMap;

}

public void setProccessDefService(IProccessDefService proccessDefService) {

this.proccessDefService = proccessDefService;

}

public IProccessDefService getProccessDefService() {

return proccessDefService;

}

@InitBinder

public void initBinder(WebDataBinder binder) {

binder.registerCustomEditor(Date.class, new PropertyEditorSupport() {

public void setAsText(String value) {

try {

setValue(new SimpleDateFormat("yyyy-MM-dd HH:ss:mm").parse(value));

} catch (ParseException e) {

setValue(null);

}

}

public String getAsText() {

return new SimpleDateFormat("yyyy-MM-dd HH:ss:mm")

.format((Date) getValue());

}

});

}

public IProccessDefVersionService getProccessDefVersionService() {

return proccessDefVersionService;

}

public void setProccessDefVersionService(

IProccessDefVersionService proccessDefVersionService) {

this.proccessDefVersionService = proccessDefVersionService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.BpmActivityReader;

import com.jiexunnet.workflow.model.BpmParticipant;

import com.jiexunnet.workflow.service.IBpmActivityReaderService;

import com.jiexunnet.workflow.service.IBpmParticipantService;

@Controller

public class ReaderSelectController {

@Autowired

private IBpmActivityReaderService readerService ;

@Autowired

private IBpmParticipantService PerticipantService ;

@RequestMapping(value="/workflowdefine/reader/list.do")

public @ResponseBody Map<String, Object> findByActivitydefUuid(String activitydefId){

Map<String, Object> Map = new HashMap<String, Object>();

List<Map<String, Object>> acdefs = new ArrayList<Map<String, Object>>();

@SuppressWarnings("unchecked")

List<Object[]> readerslist = this.readerService.getBpmActivityReaders(activitydefId);

if(readerslist.size()>0){

for(int i=0;i<readerslist.size();i++){

Object[] objects = readerslist.get(i);

acdefs.add(bpmpantToMap(objects)) ;

}

}

Map.put("success", true);

Map.put("data", acdefs);

return Map;

}

@RequestMapping(value="/workflowdefine/reader/del.do")

public @ResponseBody Map<String, Object> del(String readerIds){

Map<String, Object> retMap = new HashMap<String, Object>();

if(readerIds!=null)

this.readerService.delBpmActivityReaders(readerIds);

retMap.put("success", true);

return retMap;

}

private Map<String, Object> bpmpantToMap(Object[] object){

Map<String, Object> map = new HashMap<String, Object>();

BpmParticipant bp = (BpmParticipant)object[0];

BpmActivityReader bp1 = (BpmActivityReader)object[1];

map.put("uuid", bp1.getUuid());

map.put("participantName", bp.getParticipantName());

map.put("participantDesc", bp.getParticipantDesc());

map.put("participantSelectUuid", bp1.getParticipantSelectUuid());

map.put("participantSelectParam", bp1.getParticipantSelectParam());

map.put("participantSelectParamName", bp1.getParticipantSelectParamName());

return map;

}

public IBpmParticipantService getPerticipantService() {

return PerticipantService;

}

public void setPerticipantService(IBpmParticipantService perticipantService) {

PerticipantService = perticipantService;

}

public IBpmActivityReaderService getReaderService() {

return readerService;

}

public void setReaderService(IBpmActivityReaderService readerService) {

this.readerService = readerService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.workflow.model.BPMActivityDef;

import com.jiexunnet.workflow.model.BPMRouteDef;

import com.jiexunnet.workflow.service.IRouteDefService;

@Controller

public class RouteDefController {

@Autowired

private IRouteDefService routeDefService;

@RequestMapping(value="/workflowdefine/fromroute/list.do")

public @ResponseBody Map<String, Object> fromlist(String proccessversionid){

Map<String, Object> retMap = new HashMap<String, Object>();

List<Map<String, Object>> acdefs = new ArrayList<Map<String, Object>>();

List<BPMActivityDef> lists = this.routeDefService.getActivityDef(1, proccessversionid);

for(int i=0;i<lists.size();i++){

Map<String, Object> map = this.ActivityToMap(lists.get(i));

if(map !=null)

acdefs.add(map);

}

retMap.put("success", true);

retMap.put("data", acdefs);

return retMap;

}

@RequestMapping(value="/workflowdefine/endroute/list.do")

public @ResponseBody Map<String, Object> tolist(Integer activityType, String proccessversionid){

Map<String, Object> retMap = new HashMap<String, Object>();

List<Map<String, Object>> acdefs = new ArrayList<Map<String, Object>>();

List<BPMActivityDef> lists = this.routeDefService.getActivityDef(0, proccessversionid);

for(int i=0;i<lists.size();i++){

Map<String, Object> map = this.ActivityToMap1(lists.get(i));

if(map !=null)

acdefs.add(map);

}

retMap.put("success", true);

retMap.put("data", acdefs);

return retMap;

}

@RequestMapping(value="/workflowdefine/routedef/list.do")

public @ResponseBody Map<String, Object> listchildren(String versionUuid){

Map<String, Object> retMap = new HashMap<String, Object>();

List<BPMRouteDef> lists = this.routeDefService.listchildren(versionUuid);

retMap.put("success", true);

retMap.put("data", lists);

return retMap;

}

@RequestMapping(value="/workflowdefine/routedef/add.do")

public @ResponseBody Map<String, Object> add(@RequestBody BPMRouteDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

this.routeDefService.save(def);

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/routedef/del.do")

public @ResponseBody Map<String, Object> del(@RequestBody BPMRouteDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

this.routeDefService.delete(def.getUuid());

retMap.put("success", true);

return retMap;

}

@RequestMapping(value="/workflowdefine/routedef/update.do")

public @ResponseBody Map<String, Object> update(@RequestBody BPMRouteDef def){

Map<String, Object> retMap = new HashMap<String, Object>();

this.routeDefService.update(def);

retMap.put("success", true);

return retMap;

}

private Map<String, Object> ActivityToMap(BPMActivityDef activitydef){

Map<String, Object> map = new HashMap<String, Object>();

map.put("id", activitydef.getUuid());

map.put("fromRoute", activitydef.getActivityDefName());

return map;

}

private Map<String, Object> ActivityToMap1(BPMActivityDef activitydef){

Map<String, Object> map = new HashMap<String, Object>();

map.put("id", activitydef.getUuid());

map.put("endRoute", activitydef.getActivityDefName());

return map;

}

public IRouteDefService getRouteDefService() {

return routeDefService;

}

public void setRouteDefService(IRouteDefService routeDefService) {

this.routeDefService = routeDefService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.usm.model.TDept;

import com.jiexunnet.usm.model.TDuty;

import com.jiexunnet.usm.model.TGroup;

import com.jiexunnet.usm.model.TPosition;

import com.jiexunnet.usm.model.TUser;

import com.jiexunnet.usm.model.URole;

import com.jiexunnet.usm.service.IDeptService;

import com.jiexunnet.usm.service.IDutyService;

import com.jiexunnet.usm.service.IGroupService;

import com.jiexunnet.usm.service.IPositionService;

import com.jiexunnet.usm.service.IUserService;

import com.jiexunnet.usm.service.URoleService;

@Controller

public class UsmController {

@Autowired

@Qualifier("IGroupService")

private IGroupService groupService;

@Autowired

@Qualifier("IDutyService")

private IDutyService dutyService;

@Autowired

@Qualifier("URoleService")

private URoleService roleService;

@Autowired

@Qualifier("IDeptService")

private IDeptService deptService;

@Autowired

@Qualifier("IPositionService")

private IPositionService positionService;

@Autowired

@Qualifier("IUserService")

private IUserService userService;

//查询所有的人员（树--所有部门，然后部门下的所有人员）

@RequestMapping(value="/workflow/usertreeinit.do")

public String usertree(String checked, String disabled,HttpServletRequest request){

request.setAttribute("checked", checked);

request.setAttribute("disabled", disabled);

return "/workflow/usertree";

}

@RequestMapping(value="/workflow/usertree.do")

public @ResponseBody Map<String, Object> treegrid(){

Map<String, Object> map = new HashMap<String, Object>();

try{

List<TDept> roots = this.deptService.getRootDept();

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(0);

if(roots != null && roots.size()>0){

for(int i=0; i<roots.size(); i++){

list.add(processChild(roots.get(i)));

}

}

map.put("id", -1);

map.put("name", "组织人员树");

map.put("isParent", true);

map.put("children", list);

// map.put("success", true);

}catch(Exception e){

e.printStackTrace();

// map.put("success", false);

}

return map;

}

private Map<String, Object> processChild(TDept dept){

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("name", dept.getDeptName());

map.put("isParent", true);

List<TUser> users = this.userService.getAllUser(dept.getId());

Collections.sort(users, new Comparator<TUser>(){

public int compare(TUser arg0, TUser arg1) {

return (arg0.getUserOrder())-(arg1.getUserOrder());

}

});

if(users != null && users.size()>0){

for(int i=0; i<users.size(); i++){

Map<String, Object> user = new HashMap<String, Object>();

user.put("id", "user-" + users.get(i).getId());

user.put("name", users.get(i).getUserName());

user.put("isParent", false);

lists.add(user);

}

}

List<TDept> children = dept.getDepts();

Collections.sort(children, new Comparator<TDept>(){

public int compare(TDept arg0, TDept arg1) {

return (arg0.getDeptOrder())-(arg1.getDeptOrder());

}

});

if(children != null && children.size() > 0){

// List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(0);

for(int i=0; i<children.size(); i++){

lists.add(processChild(children.get(i)));

}

}

if(lists.size()>0)

map.put("children", lists);

return map;

}

//查询所有的部门（树--所有部门） //带有多选框的部门树

@RequestMapping(value="/workflow/depttreeinit.do")

public String orgtree(String checked, String disabled,HttpServletRequest request){

request.setAttribute("checked", checked);

request.setAttribute("disabled", disabled);

return "/workflow/depttree";

}

@RequestMapping(value="/workflow/depttree.do")

public @ResponseBody Map<String, Object> treedeptgrid(){

Map<String, Object> map = new HashMap<String, Object>();

try{

List<TDept> roots = this.deptService.getRootDept();

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(0);

if(roots != null && roots.size()>0){

for(int i=0; i<roots.size(); i++){

list.add(processDeptChild(roots.get(i)));

}

}

map.put("id", -1);

map.put("name", "组织机构树");

map.put("isParent", true);

map.put("children", list);

// map.put("success", true);

}catch(Exception e){

e.printStackTrace();

// map.put("success", false);

}

return map;

}

private Map<String, Object> processDeptChild(TDept dept){

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("name", dept.getDeptName());

map.put("isParent", true);

List<TDept> children = dept.getDepts();

Collections.sort(children, new Comparator<TDept>(){

public int compare(TDept arg0, TDept arg1) {

return (arg0.getDeptOrder())-(arg1.getDeptOrder());

}

});

if(children != null && children.size() > 0){

// List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(0);

for(int i=0; i<children.size(); i++){

lists.add(processDeptChild(children.get(i)));

}

}

if(lists.size()>0)

map.put("children", lists);

return map;

}

//查询所有的岗位（树--所有部门，然后部门下的所有岗位）

@RequestMapping(value="/workflow/positiontreeinit.do")

public String positiontree(String checked, String disabled,HttpServletRequest request){

request.setAttribute("checked", checked);

request.setAttribute("disabled", disabled);

return "/workflow/positiontree";

}

@RequestMapping(value="/workflow/positiontree.do")

public @ResponseBody Map<String, Object> positiontreegrid(){

Map<String, Object> map = new HashMap<String, Object>();

try{

List<TDept> roots = this.deptService.getRootDept();

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(0);

if(roots != null && roots.size()>0){

for(int i=0; i<roots.size(); i++){

list.add(processPositionChild(roots.get(i)));

}

}

map.put("id", -1);

map.put("name", "组织岗位树");

map.put("isParent", true);

map.put("children", list);

// map.put("success", true);

}catch(Exception e){

e.printStackTrace();

// map.put("success", false);

}

return map;

}

private Map<String, Object> processPositionChild(TDept dept){

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("name", dept.getDeptName());

map.put("isParent", true);

List<TPosition> positions = this.positionService.getPositions(dept.getId());

for(int i=0;i<positions.size();i++){

Map<String,Object> map1 = new HashMap<String,Object>();

map1.put("id","position-" + positions.get(i).getId());

map1.put("name", positions.get(i).getPositionName());

map1.put("isParent", false);

lists.add(map1);

}

List<TDept> children = dept.getDepts();

Collections.sort(children, new Comparator<TDept>(){

public int compare(TDept arg0, TDept arg1) {

return (arg0.getDeptOrder())-(arg1.getDeptOrder());

}

});

if(children != null && children.size() > 0){

// List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(0);

for(int i=0; i<children.size(); i++){

lists.add(processPositionChild(children.get(i)));

}

}

if(lists.size()>0)

map.put("children", lists);

return map;

}

//查询所有的角色

@RequestMapping(value="/workflow/listAllRole.do")

public @ResponseBody Map<String, Object> listAllRole(String ids){

Map<String, Object> retMap = new HashMap<String, Object>();

List<URole> list = this.roleService.listRoleYx(ids);

retMap.put("success", true);

retMap.put("data", list);

return retMap;

}

//查询所有的职务

public @ResponseBody Map<String, Object> listDuty(){

Map<String, Object> retMap = new HashMap<String, Object>();

List<TDuty> list = this.dutyService.listAll();

retMap.put("success", true);

retMap.put("data", list);

return retMap;

}

//查询所有的工作组

public @ResponseBody Map<String, Object> listGroup(){

Map<String, Object> retMap = new HashMap<String, Object>();

List<TGroup> list = this.groupService.listAll();

retMap.put("success", true);

retMap.put("data", list);

return retMap;

}

public IGroupService getGroupService() {

return groupService;

}

public void setGroupService(IGroupService groupService) {

this.groupService = groupService;

}

public IDutyService getDutyService() {

return dutyService;

}

public void setDutyService(IDutyService dutyService) {

this.dutyService = dutyService;

}

public URoleService getRoleService() {

return roleService;

}

public void setRoleService(URoleService roleService) {

this.roleService = roleService;

}

public IDeptService getDeptService() {

return deptService;

}

public void setDeptService(IDeptService deptService) {

this.deptService = deptService;

}

public IPositionService getPositionService() {

return positionService;

}

public void setPositionService(IPositionService positionService) {

this.positionService = positionService;

}

public IUserService getUserService() {

return userService;

}

public void setUserService(IUserService userService) {

this.userService = userService;

}

}

package com.jiexunnet.workflow.controller;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.SortedSet;

import java.util.concurrent.ConcurrentSkipListSet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

import com.jiexunnet.usm.model.URole;

import com.jiexunnet.usm.service.URoleService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.ApplicationContext;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.jiexunnet.common.util.SpringContextUtil;

import com.jiexunnet.usm.model.TDept;

import com.jiexunnet.usm.model.TUser;

import com.jiexunnet.usm.service.IDeptService;

import com.jiexunnet.usm.service.IUserService;

import com.jiexunnet.workflow.model.BpmActivityPerformer;

import com.jiexunnet.workflow.model.BpmActivityReader;

import com.jiexunnet.workflow.model.BpmParticipant;

import com.jiexunnet.workflow.service.IBpmActivityPerformerService;

import com.jiexunnet.workflow.service.IBpmActivityReaderService;

import com.jiexunnet.workflow.service.UserInterfaceService;

@Controller

public class Workflowtree {

private static final Logger LOGGER = LoggerFactory.getLogger(Workflowtree.class);

@Autowired

private IBpmActivityPerformerService performerService;

@Autowired

private IBpmActivityReaderService readerService;

@Autowired

private IDeptService deptService;

@Autowired

private IUserService userService;

@Autowired

private URoleService uRoleService;

private ApplicationContext context = SpringContextUtil

.getApplicationContext();

//@RequestMapping(value="/workflow/workflowtreelistHy.do")

public @ResponseBody List<Map<String, Object>> getPerformersHy(String activitydefId,String processInstId,String activityInstId,HttpSession session){

Map<String, Object> resultMap = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

List<Map<String, Object>> rlist = new ArrayList<Map<String, Object>>(0);

@SuppressWarnings("unchecked")

List<Object[]> list = performerService.getBpmActivityPerformers(activitydefId);

TUser user = (TUser)session.getAttribute("user");

Object [] objects = list.get(0);

BpmActivityPerformer bp1 = (BpmActivityPerformer) objects[1];

resultMap.put("id", -1);

resultMap.put("text", "局有关单位");

resultMap.put("hasChildren", true);

resultMap.put("expanded", true);

//UserInterfaceRoleService

UserInterfaceService uiService = (UserInterfaceService) context.getBean("UserInterfaceRoleService");

List<TDept> roots = this.deptService.getRootDept();

for (int j = 0; j < roots.size(); j++) {

Map<String, Object> c = processChildHy(roots.get(j), uiService, activitydefId,processInstId, activityInstId, bp1.getParticipantSelectParam(), user.getId().toString());

if(c!=null)

rlist.add(c);

}

Map<String, Object> attributesMap = new HashMap<String, Object>();

attributesMap.put("flag", "user");

resultMap.put("attributes", attributesMap);

resultMap.put("children", rlist);

lists.add(resultMap);

return lists;

}

private Map<String, Object> processChildHy(TDept dept, UserInterfaceService uiService,

String activityDefId,

String processInstId,

String activityInstId,

String params,

String personId){

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("text", dept.getDeptName());

map.put("expanded", true);

map.put("hasChildren", true);

map.put("flag", false);

List<TUser> user = uiService.getUsers(processInstId, activityInstId, activityDefId, context, params, dept.getId(), personId);

for (int i = 0; i < user.size(); i++) {

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", "user-" + user.get(i).getId());

map1.put("text", user.get(i).getUserName());

map1.put("expanded", true);

map1.put("hasChildren", false);

Map<String, Object> attributesMap1 = new HashMap<String, Object>();

attributesMap1.put("flag", true);

map1.put("attributes", attributesMap1);

lists.add(map1);

}

List<TDept> children = dept.getDepts();

if (children != null && children.size() > 0) {

for (int i = 0; i < children.size(); i++) {

Map<String, Object> c = processChildHy(children.get(i), uiService, activityDefId,processInstId, activityInstId, params, personId);

if (c != null)

lists.add(c);

}

}

if (lists.size() > 0) {

map.put("children", lists);

return map;

} else

return null;

}

@RequestMapping(value = "/workflow/workflowrolelist.do")

public @ResponseBody List<Map<String, Object>> getRole(){//获取所有角色

Map<String, Object> resultMap = new HashMap<String, Object>();

List<Map<String, Object>> rlist = new ArrayList<Map<String, Object>>(0);

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

resultMap.put("id", -1);

resultMap.put("text", "所有岗位");

resultMap.put("hasChildren", true);

resultMap.put("expanded", true);

resultMap.put("flag", "role");

List<URole> allRole = uRoleService.findAllRole();

for(URole role : allRole){

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", role.getId());

map1.put("text", role.getRoleName());

map1.put("expanded", true);

map1.put("hasChildren", false);

map1.put("flag", true);

rlist.add(map1);

}

resultMap.put("flag", "role");

resultMap.put("children", rlist);

lists.add(resultMap);

return lists;

}

@RequestMapping(value = "/workflow/workflowrole.do")

public @ResponseBody Map<String, Object> getPerformerRole(String activitydefId){//根据环节定义，从BpmActivityPerformer获取指定的角色id

List<BpmActivityPerformer> activityPerformerList = performerService.findByActivityDefUuid(activitydefId);

Map<String, Object> map = new HashMap<String, Object>();

map.put("excutorId", activityPerformerList.get(0).getParticipantSelectParam());

return map;

}

@SuppressWarnings("unused")

@RequestMapping(value = "/workflow/workflowtreelist.do")

public @ResponseBody

List<Map<String, Object>> getPerformers(String activitydefId,String processInstId,String activityInstId,HttpSession session) {

long time = System.currentTimeMillis();

LOGGER.warn("进入/workflow/workflowtreelist.do的时间="+time);

Map<String, Object> resultMap = new HashMap<String, Object>();

List<Map<String, Object>> rlist = new ArrayList<Map<String, Object>>(0);

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

@SuppressWarnings("unchecked")

List<Object[]> list = performerService.getBpmActivityPerformers(activitydefId);

TUser users = (TUser)session.getAttribute("user");

Integer userId = users.getId();

resultMap.put("id", -1);

resultMap.put("text", "组织机构树");

resultMap.put("hasChildren", true);

resultMap.put("expanded", true);

resultMap.put("flag", "user");

List<Map<String,Object>> bplist = new ArrayList<Map<String, Object>>(0);

if (list != null && list.size() > 0) {

for (int i = 0; i < list.size(); i++) {

Object[] objects = list.get(i);

BpmParticipant bp = (BpmParticipant) objects[0];

BpmActivityPerformer bp1 = (BpmActivityPerformer) objects[1];

Map<String, Object> bpmap = new HashMap<String, Object>();

String params = bp1.getParticipantSelectParam();

bpmap.put("params", params);

String executeclass = bp.getParticipantClass();

bpmap.put("executeclass", executeclass);

String participantName = bp.getParticipantName();

bpmap.put("participantName", participantName);

Integer deptFlag = bp.getDeptFlag();

bpmap.put("deptFlag", deptFlag);

bplist.add(bpmap);

}

}

List<TUser> userlist = new ArrayList<TUser>();

List<TUser> userlists = new ArrayList<TUser>();

if (bplist.size() > 0) {

for (int i = 0; i < bplist.size(); i++) {

UserInterfaceService uiService = (UserInterfaceService) context

.getBean((String)bplist.get(i).get("executeclass"));

if((Integer)bplist.get(i).get("deptFlag")==0){

List<TDept> roots = this.deptService.getRootDept();

for (int j = 0; j < roots.size(); j++) {

List<Map<String, Object>> c = processChildHyLevel1(roots.get(j), activitydefId,processInstId, activityInstId, userId.toString(),list);

if(c!=null)

rlist.addAll(c);

}

resultMap.put("flag", "user");

resultMap.put("children", rlist);

lists.add(resultMap);

long time2 = System.currentTimeMillis();

LOGGER.warn("出/workflow/workflowtreelist.do的时间="+time2);

return lists;

}else if((Integer)bplist.get(i).get("deptFlag")==1){

List<TDept> deptlist = new ArrayList<TDept>();

String params =(String)bplist.get(i).get("params");

if (params!= null) {

String[] param = params.split(",");

for (int j = 0; j < param.length; j++) {

int p = Integer.parseInt(param[j]);

List<TDept> deptss = uiService.getDepts(processInstId, activityInstId,

activitydefId, null, p, userId+"");

deptlist.addAll(deptss);

}

}

for (int j = 0; j < deptlist.size(); j++) {

Map<String, Object> c = processDept1(deptlist.get(j));

if(c!=null)

rlist.add(c);

}

resultMap.put("flag", "dept1");

resultMap.put("children", rlist);

lists.add(resultMap);

return lists;

}else{

List<TDept> deptlist = new ArrayList<TDept>();

String params =(String)bplist.get(i).get("params");

if (params != null) {

String[] param = params.split(",");

for (int j = 0; j < param.length; j++) {

int p = Integer.parseInt(param[i]);

List<TDept> deptss = uiService.getDepts(processInstId, activityInstId,

activitydefId, null, p, userId+"");

deptlist.addAll(deptss);

}

}

for (int j = 0; j < deptlist.size(); j++) {

Map<String, Object> c = processDept(deptlist.get(j));

if(c!=null)

rlist.add(c);

}

resultMap.put("flag", "dept2");

resultMap.put("children", rlist);

lists.add(resultMap);

return lists;

}

}

}

return lists;

}

@SuppressWarnings({ "unused", "unchecked" })

@RequestMapping(value = "/workflow/withoutchoice.do")

public @ResponseBody List<Map<String, Object>> withoutchoice(String activitydefId,String processInstId,String activityInstId,HttpSession session ) {

Map<String, Object> resultMap = new HashMap<String, Object>();

List<Map<String, Object>> rlist = new ArrayList<Map<String, Object>>(0);

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

List<Object[]> list = performerService.getBpmActivityPerformers(activitydefId);

TUser user = (TUser)session.getAttribute("user");

Integer userId = user.getId();

resultMap.put("id", -1);

resultMap.put("text", "市有关单位");

resultMap.put("hasChildren", true);

resultMap.put("expanded", true);

resultMap.put("flag", "user");

List<Map<String,Object>> bplist = new ArrayList<Map<String, Object>>(0);

if (list != null && list.size() > 0) {

for (int i = 0; i < list.size(); i++) {

Object[] objects = list.get(i);

BpmParticipant bp = (BpmParticipant) objects[0];

BpmActivityPerformer bp1 = (BpmActivityPerformer) objects[1];

Map<String, Object> bpmap = new HashMap<String, Object>();

String params = bp1.getParticipantSelectParam();

bpmap.put("params", params);

String executeclass = bp.getParticipantClass();

bpmap.put("executeclass", executeclass);

String participantName = bp.getParticipantName();

bpmap.put("participantName", participantName);

Integer deptFlag = bp.getDeptFlag();

bpmap.put("deptFlag", deptFlag);

bplist.add(bpmap);

}

}

List<TUser> userlist = new ArrayList<TUser>();

List<TUser> userlists = new ArrayList<TUser>();

if (bplist.size() > 0) {

for (int i = 0; i < bplist.size(); i++) {

UserInterfaceService uiService = (UserInterfaceService) context

.getBean((String)bplist.get(i).get("executeclass"));

if((Integer)bplist.get(i).get("deptFlag")==0){

userlist = uiService.getUsers(processInstId, activityInstId, activitydefId, null,

(String)bplist.get(i).get("params"), null, userId+"");

if (userlist.size() > 0) {

userlists.addAll(userlist);

}

}else if((Integer)bplist.get(i).get("deptFlag")==1){

String params =(String)bplist.get(i).get("params");

if (params!= null) {

String[] param = params.split(",");

for (int j = 0; j < param.length; j++) {

int p = Integer.parseInt(param[j]);

List<TDept> deptss = uiService.getDepts(processInstId, activityInstId,

activitydefId, null, p, userId+"");

if(deptss!=null && deptss.size()>0){

userlist = userService.getAllUser(deptss.get(0).getId());

if (userlist.size() > 0) {

userlists.addAll(userlist);

}

}

}

}

}else{

String params =(String)bplist.get(i).get("params");

if (params != null) {

String[] param = params.split(",");

for (int j = 0; j < param.length; j++) {

int p = Integer.parseInt(param[i]);

List<TDept> deptss = uiService.getDepts(processInstId, activityInstId,

activitydefId, null, p, userId+"");

if(deptss!=null && deptss.size()>0){

for(int m=0;m<deptss.size();m++){

userlist = userService.getAllUser(deptss.get(m).getId());

if (userlist.size() > 0) {

userlists.addAll(userlist);

}

}

}

}

}

}

}

}

if(userlists.size()>0){

for(int j=0;j<userlists.size();j++){

Map<String, Object> c = userToMap(userlists.get(j));

if(c!=null)

lists.add(c);

}

}

return lists;

}

public List<TUser> removeDuplicateWithOrder(List<TUser> userlists) {

Set<TUser> set = new HashSet<TUser>();

List<TUser> newList = new ArrayList<TUser>();

for (Iterator<TUser> iter = userlists.iterator(); iter.hasNext();) {

TUser element = iter.next();

if (set.add(element))

newList.add(element);

}

userlists.clear();

userlists.addAll(newList);

return userlists;

}

private Map<String, Object> userToMap(TUser user){

Map<String, Object> map = new HashMap<String, Object>();

List<TDept> deptlist = deptService.queryByUserId(user.getId());

Integer deptId=null;

if(deptlist!=null && deptlist.size()>0){

deptId = deptlist.get(0).getId();

}

map.put("id", user.getId());

map.put("deptId", deptId);

map.put("userName", user.getUserName());

map.put("loginName", user.getLoginName());

map.put("loginPassword", user.getLoginPassword());

map.put("userEmail", user.getUserEmail());

map.put("userMobile", user.getUserMobile());

map.put("userOfficePhone", user.getUserOfficePhone());

map.put("userHomePhone", user.getUserHomePhone());

map.put("isAdmin", user.getIsAdmin());

map.put("registerDate", user.getRegisterDate());

map.put("userOrder", user.getUserOrder());

return map;

}

@SuppressWarnings("unused")

@RequestMapping(value = "/workflow/workflowtreereaderlist.do")

public @ResponseBody

List<Map<String, Object>> getReader(String activitydefId,String processInstId,String activityInstId,HttpSession session) {

long time = System.currentTimeMillis();

LOGGER.warn("进入/workflow/workflowtreereaderlist.do的时间="+time);

Map<String, Object> resultMap = new HashMap<String, Object>();

List<Map<String, Object>> rlist = new ArrayList<Map<String, Object>>(0);

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

@SuppressWarnings("unchecked")

List<Object[]> list = readerService.getBpmActivityReaders(activitydefId);

TUser users = (TUser)session.getAttribute("user");

Integer userId = users.getId();

resultMap.put("id", -1);

resultMap.put("text", "组织机构树");

resultMap.put("hasChildren", true);

resultMap.put("expanded", true);

resultMap.put("flag", "user");

List<Map<String,Object>> bplist = new ArrayList<Map<String, Object>>(0);

if (list != null && list.size() > 0) {

for (int i = 0; i < list.size(); i++) {

Object[] objects = list.get(i);

BpmParticipant bp = (BpmParticipant) objects[0];

BpmActivityReader bp1 = (BpmActivityReader) objects[1];

Map<String, Object> bpmap = new HashMap<String, Object>();

String params = bp1.getParticipantSelectParam();

bpmap.put("params", params);

String executeclass = bp.getParticipantClass();

bpmap.put("executeclass", executeclass);

String participantName = bp.getParticipantName();

bpmap.put("participantName", participantName);

Integer deptFlag = bp.getDeptFlag();

bpmap.put("deptFlag", deptFlag);

bplist.add(bpmap);

}

}

List<TUser> userlist = new ArrayList<TUser>();

List<TUser> userlists = new ArrayList<TUser>();

if (bplist.size() > 0) {

for (int i = 0; i < bplist.size(); i++) {

UserInterfaceService uiService = (UserInterfaceService) context

.getBean((String)bplist.get(i).get("executeclass"));

if((Integer)bplist.get(i).get("deptFlag")==0){

List<TDept> roots = this.deptService.getRootDept();

for (int j = 0; j < roots.size(); j++) {

Map<String, Object> c = processChildReader(roots.get(j), activitydefId,processInstId, activityInstId, userId.toString(),list);

if(c!=null)

rlist.add(c);

}

resultMap.put("flag", "user");

resultMap.put("children", rlist);

lists.add(resultMap);

long time2 = System.currentTimeMillis();

LOGGER.warn("出/workflow/workflowtreereaderlist.do的时间="+time2);

return lists;

}else if((Integer)bplist.get(i).get("deptFlag")==1){

List<TDept> deptlist = new ArrayList<TDept>();

String params =(String)bplist.get(i).get("params");

if (params!= null) {

String[] param = params.split(",");

for (int j = 0; j < param.length; j++) {

int p = Integer.parseInt(param[j]);

List<TDept> deptss = uiService.getDepts(processInstId, activityInstId,

activitydefId, null, p, userId+"");

deptlist.addAll(deptss);

}

}

for (int j = 0; j < deptlist.size(); j++) {

Map<String, Object> c = processDept1(deptlist.get(j));

if(c!=null)

rlist.add(c);

}

resultMap.put("flag", "dept1");

resultMap.put("children", rlist);

lists.add(resultMap);

return lists;

}else{

List<TDept> deptlist = new ArrayList<TDept>();

String params =(String)bplist.get(i).get("params");

if (params != null) {

String[] param = params.split(",");

for (int j = 0; j < param.length; j++) {

int p = Integer.parseInt(param[i]);

List<TDept> deptss = uiService.getDepts(processInstId, activityInstId,

activitydefId, null, p, userId+"");

deptlist.addAll(deptss);

}

}

for (int j = 0; j < deptlist.size(); j++) {

Map<String, Object> c = processDept(deptlist.get(j));

if(c!=null)

rlist.add(c);

}

resultMap.put("flag", "dept2");

resultMap.put("children", rlist);

lists.add(resultMap);

return lists;

}

}

}

return lists;

}

private Map<String, Object> processChildHy(TDept dept,String activityDefId,String processInstId,String activityInstId, String personId,List<Object[]> list){

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("text", dept.getDeptName());

map.put("expanded", true);

map.put("hasChildren", true);

Map<String, Object> attributesMap = new HashMap<String, Object>();

attributesMap.put("flag", false);

map.put("attributes", attributesMap);

SortedSet<TUser> users = new ConcurrentSkipListSet<TUser>();

for(int i=0; i<list.size(); i++){

Object[] objects = list.get(i);

BpmParticipant bp = (BpmParticipant) objects[0];

BpmActivityPerformer bap = (BpmActivityPerformer) objects[1];

UserInterfaceService uiService = (UserInterfaceService) context.getBean(bp.getParticipantClass());

List<TUser> user = uiService.getUsers(processInstId, activityInstId, activityDefId, context, bap.getParticipantSelectParam(), dept.getId(), personId);

if(user.size() > 0)

users.addAll(user);

}

Iterator<TUser> u = users.iterator();

while(u.hasNext()){

TUser user = u.next();

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", "user-" +user.getId());

map1.put("text", user.getUserName());

map1.put("expanded", true);

map1.put("hasChildren", false);

map1.put("flag", true);

map1.put("userflag", user.getFlag());

lists.add(map1);

}

List<TDept> children = dept.getDepts();

if (children != null && children.size() > 0) {

for (int i = 0; i < children.size(); i++) {

Map<String, Object> c = processChildHy(children.get(i),activityDefId,processInstId, activityInstId, personId, list);

if (c != null)

lists.add(c);

}

}

if (lists.size() > 0) {

map.put("children", lists);

return map;

} else

return null;

}

private List<Map<String, Object>> processChildHyLevel1(TDept dept,String activityDefId,String processInstId,String activityInstId, String personId,List<Object[]> list){

//Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

// map.put("id", dept.getId());

// map.put("text", dept.getDeptName());

// map.put("expanded", true);

// map.put("hasChildren", true);

// Map<String, Object> attributesMap = new HashMap<String, Object>();

// attributesMap.put("flag", false);

//

// map.put("attributes", attributesMap);

SortedSet<TUser> users = new ConcurrentSkipListSet<TUser>();

for(int i=0; i<list.size(); i++){

Object[] objects = list.get(i);

BpmParticipant bp = (BpmParticipant) objects[0];

BpmActivityPerformer bap = (BpmActivityPerformer) objects[1];

UserInterfaceService uiService = (UserInterfaceService) context.getBean(bp.getParticipantClass());

List<TUser> user = uiService.getUsers(processInstId, activityInstId, activityDefId, context, bap.getParticipantSelectParam(), dept.getId(), personId);

if(user.size() > 0)

users.addAll(user);

}

Iterator<TUser> u = users.iterator();

while(u.hasNext()){

TUser user = u.next();

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", "user-" +user.getId());

map1.put("text", user.getUserName());

map1.put("expanded", true);

map1.put("hasChildren", false);

map1.put("flag", true);

map1.put("userflag", user.getFlag());

lists.add(map1);

}

List<TDept> children = dept.getDepts();

if (children != null && children.size() > 0) {

for (int i = 0; i < children.size(); i++) {

Map<String, Object> c = processChildHy(children.get(i),activityDefId,processInstId, activityInstId, personId, list);

if (c != null)

lists.add(c);

}

}

if (lists.size() > 0) {

// map.put("children", lists);

// return map;

return lists;

} else

return null;

}

private Map<String, Object> processChildReader(TDept dept,String activityDefId,String processInstId,String activityInstId, String personId,List<Object[]> list){

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("text", dept.getDeptName());

map.put("expanded", true);

map.put("hasChildren", true);

Map<String, Object> attributesMap = new HashMap<String, Object>();

attributesMap.put("flag", false);

map.put("attributes", attributesMap);

SortedSet<TUser> users = new ConcurrentSkipListSet<TUser>();

for(int i=0; i<list.size(); i++){

Object[] objects = list.get(i);

BpmParticipant bp = (BpmParticipant) objects[0];

BpmActivityReader bap = (BpmActivityReader) objects[1];

UserInterfaceService uiService = (UserInterfaceService) context.getBean(bp.getParticipantClass());

List<TUser> user = uiService.getUsers(processInstId, activityInstId, activityDefId, context, bap.getParticipantSelectParam(), dept.getId(), personId);

if(user.size() > 0)

users.addAll(user);

}

Iterator<TUser> u = users.iterator();

while(u.hasNext()){

TUser user = u.next();

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", "user-" +user.getId());

map1.put("text", user.getUserName());

map1.put("expanded", true);

map1.put("hasChildren", false);

map1.put("flag", true);

map1.put("userflag", user.getFlag());

lists.add(map1);

}

List<TDept> children = dept.getDepts();

if (children != null && children.size() > 0) {

for (int i = 0; i < children.size(); i++) {

Map<String, Object> c = processChildReader(children.get(i),activityDefId,processInstId, activityInstId, personId, list);

if (c != null)

lists.add(c);

}

}

if (lists.size() > 0) {

map.put("children", lists);

return map;

} else

return null;

}

@SuppressWarnings("unused")

private Map<String, Object> processChild(TDept dept, List<TUser> user) {

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("text", dept.getDeptName());

map.put("expanded", true);

map.put("hasChildren", true);

Map<String, Object> attributesMap1 = new HashMap<String, Object>();

attributesMap1.put("flag", false);

map.put("attributes", attributesMap1);

for (int i = 0; i < user.size(); i++) {

if (deptService.istheDept(dept.getId(), user.get(i).getId())) {

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", "user-" + user.get(i).getId());

map1.put("text", user.get(i).getUserName());

map1.put("expanded", true);

map1.put("hasChildren", false);

map1.put("flag", true);

lists.add(map1);

user.remove(user.get(i));

i--;

}

}

List<TDept> children = dept.getDepts();

if (children != null && children.size() > 0) {

for (int i = 0; i < children.size(); i++) {

Map<String, Object> c = processChild(children.get(i), user);

if (c != null)

lists.add(c);

}

}

if (lists.size() > 0) {

map.put("children", lists);

return map;

} else

return null;

}

private Map<String, Object> processDept(Object o) {

TDept dept = (TDept) o;

Map<String, Object> map = new HashMap<String, Object>();

List<Map<String, Object>> lists = new ArrayList<Map<String, Object>>();

map.put("id", dept.getId());

map.put("text", dept.getDeptName());

map.put("expanded", true);

map.put("hasChildren", true);

map.put("flag", false);

List<TDept> children = dept.getDepts();

Collections.sort(children, new Comparator<TDept>() {

public int compare(TDept arg0, TDept arg1) {

return (arg0.getDeptOrder()) - (arg1.getDeptOrder());

}

});

if (children != null && children.size() > 0) {

for (int i = 0; i < children.size(); i++) {

Map<String, Object> c = processDept(children.get(i));

if(c!=null)

lists.add(c);

}

} else {

map.put("hasChildren", false);

}

if (lists.size() > 0)

map.put("children", lists);

return map;

}

private Map<String, Object> processDept1(TDept dept) {

Map<String, Object> map = new HashMap<String, Object>();

map.put("id", dept.getId());

map.put("text", dept.getDeptName());

map.put("expanded", true);

map.put("hasChildren", false);

map.put("flag", false);

return map;

}

@RequestMapping(value = "/workflow/workflowtree.do")

public String orgtree(String checked, String disabled,

HttpServletRequest request) {

request.setAttribute("checked", checked);

request.setAttribute("disabled", disabled);

return "/workflow/workflowtree";

}

@RequestMapping(value = "/workflow/workflowreadertree.do")

public String readerorgtree(String checked, String disabled,

HttpServletRequest request) {

request.setAttribute("checked", checked);

request.setAttribute("disabled", disabled);

return "/workflow/workflowreadertree";

}

public IBpmActivityPerformerService getPerformerService() {

return performerService;

}

public void setPerformerService(

IBpmActivityPerformerService performerService) {

this.performerService = performerService;

}

public ApplicationContext getContext() {

return context;

}

public void setContext(ApplicationContext context) {

this.context = context;

}

public IDeptService getDeptService() {

return deptService;

}

public void setDeptService(IDeptService deptService) {

this.deptService = deptService;

}

public IBpmActivityReaderService getReaderService() {

return readerService;

}

public void setReaderService(IBpmActivityReaderService readerService) {

this.readerService = readerService;

}

public IUserService getUserService() {

return userService;

}

public void setUserService(IUserService userService) {

this.userService = userService;

}

}

package com.jiexunnet.workflow.controller.vo;

import java.io.Serializable;

import java.util.Date;

import com.jiexunnet.workflow.model.BPMProccessDef;

import com.jiexunnet.workflow.model.BPMProccessDefVersion;

@SuppressWarnings("serial")

public class WorkflowDefVo implements Serializable{

private String proccessuuid;

private String id;

private String proccessDefName;

private String classification;

private String systemString;

private String desc;

private Integer proccessDefType;

private Integer proccessDefVersion;

private String publicStatus;

private Date activeTime;

private Date freezeTime;

private Integer limitTime;

private Integer alertTime;

public WorkflowDefVo() {

super();

}

public WorkflowDefVo(BPMProccessDef proccessDef, BPMProccessDefVersion proccessDefVersion){

super();

this.copyData(proccessDef, proccessDefVersion);

}

public void copyData(BPMProccessDef proccessDef, BPMProccessDefVersion version){

this.setProccessuuid(proccessDef.getUuid());

this.setId(version.getUuid());

this.setProccessDefName(proccessDef.getProccessDefName());

this.setClassification(proccessDef.getClassification());

this.setSystemString(proccessDef.getSystemString());

this.setDesc(proccessDef.getDesc());

this.setProccessDefType(proccessDef.getProccessDefType());

this.setProccessDefVersion(version.getProccessDefVersion());

this.setPublicStatus(version.getPublicStatus());

this.setActiveTime(version.getActiveTime());

this.setFreezeTime(version.getFreezeTime());

this.setLimitTime(version.getLimitTime());

this.setAlertTime(version.getAlertTime());

}

public BPMProccessDef toBPMProccessDef(){

BPMProccessDef proccessDef = new BPMProccessDef(this.proccessuuid,

this.proccessDefName,

this.classification,

this.systemString,

this.desc,

this.proccessDefType);

return proccessDef;

}

public BPMProccessDefVersion toBPMProccessDefVersion(){

BPMProccessDefVersion version = new BPMProccessDefVersion(this.id,

this.proccessuuid,

this.proccessDefVersion,

this.publicStatus,

this.activeTime,

this.freezeTime,

this.limitTime,

this.alertTime);

return version;

}

public String getProccessuuid() {

return proccessuuid;

}

public void setProccessuuid(String proccessuuid) {

this.proccessuuid = proccessuuid;

}

public String getProccessDefName() {

return proccessDefName;

}

public void setProccessDefName(String proccessDefName) {

this.proccessDefName = proccessDefName;

}

public String getClassification() {

return classification;

}

public void setClassification(String classification) {

this.classification = classification;

}

public String getSystemString() {

return systemString;

}

public void setSystemString(String systemString) {

this.systemString = systemString;

}

public String getDesc() {

return desc;

}

public void setDesc(String desc) {

this.desc = desc;

}

public Integer getProccessDefType() {

return proccessDefType;

}

public void setProccessDefType(Integer proccessDefType) {

this.proccessDefType = proccessDefType;

}

public Integer getProccessDefVersion() {

return proccessDefVersion;

}

public void setProccessDefVersion(Integer proccessDefVersion) {

this.proccessDefVersion = proccessDefVersion;

}

public String getPublicStatus() {

return publicStatus;

}

public void setPublicStatus(String publicStatus) {

this.publicStatus = publicStatus;

}

public Date getActiveTime() {

return activeTime;

}

public void setActiveTime(Date activeTime) {

this.activeTime = activeTime;

}

public Date getFreezeTime() {

return freezeTime;

}

public void setFreezeTime(Date freezeTime) {

this.freezeTime = freezeTime;

}

public Integer getLimitTime() {

return limitTime;

}

public void setLimitTime(Integer limitTime) {

this.limitTime = limitTime;

}

public Integer getAlertTime() {

return alertTime;

}

public void setAlertTime(Integer alertTime) {

this.alertTime = alertTime;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((id == null) ? 0 : id.hashCode());

result = prime \* result

+ ((proccessuuid == null) ? 0 : proccessuuid.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null) {

return false;

}

if (!(obj instanceof WorkflowDefVo)) {

return false;

}

WorkflowDefVo other = (WorkflowDefVo) obj;

if (id == null) {

if (other.id != null) {

return false;

}

} else if (!id.equals(other.id)) {

return false;

}

if (proccessuuid == null) {

if (other.proccessuuid != null) {

return false;

}

} else if (!proccessuuid.equals(other.proccessuuid)) {

return false;

}

return true;

}

public void setId(String id) {

this.id = id;

}

public String getId() {

return id;

}

}