 /\*\*

     \* 发起审批流程

     \* @param defId 流程定义ID

     \* @param params 参数

     \* @return 操作结果

     \*/

    @PostMapping("start/{defId}")

    public Object startTheProcess(

        @PathVariable String defId,

        @RequestBody ProcessStartParamsVo params

    ) {

        String userId = UserUtil.getLoginUserId();

        String instanceId = processService.startProcess(defId, params, userId, null);        return R.ok("启动流程实例 " + instanceId + " 成功");

    }    /\*\*

     \* 获取审批流程实例待处理的任务列表

     \* @param instanceId 流程实例ID

     \* @return 列表数据

     \*/

    @GetMapping("{instanceId}/taskList")

    public Object getProcessInstanceTaskList(@PathVariable String instanceId) {

        return R.ok(processService.getProcessInstanceTaskList(instanceId));

    }    /\*\*

     \* 获取我发起的流程

     \* @param pageSize 每页数量

     \* @param pageNo 页码

     \* @param code 表单流程ID（流程定义KEY）

     \* @param finished 流程是否已经结束

     \* @return 列表数据

     \*/

    @GetMapping("mySubmitted")

    public Object getUserSubmittedList(

        @RequestParam(defaultValue = "20") Integer pageSize,

        @RequestParam(defaultValue = "1") Integer pageNo,

        @RequestParam(required = false) Boolean finished

    ) {

        return R.ok(processService.getUserSubmittedList(pageSize, pageNo, null, null, finished));

    }    /\*\*

     \* 获取应用关于我的流程

     \* @param code 表单流程ID（流程定义KEY）

     \* @param pageSize 每页数量

     \* @param pageNo 页码

     \* @return 列表数据

     \*/

    @GetMapping("involvedMe")

    public Object getInvolvedMeList(

        @RequestParam(required = true) String code,

        @RequestParam(defaultValue = "20") Integer pageSize,

        @RequestParam(defaultValue = "1") Integer pageNo,

        @RequestParam(required = false) String keywords,

        @RequestParam(required = false) String ids

    ) {

        return R.ok(processService.getInvolvedMeList(pageSize, pageNo, code, keywords, ids));

    }    /\*\*

     \* 关于我的所有流程

     \* @param pageSize 每页数量

     \* @param pageNo 页码

     \* @return 列表数据

     \*/

    @GetMapping("involvedMeAllApps")

    public Object getInvolvedMeListWithAllApps(

        @RequestParam(defaultValue = "20") Integer pageSize,

        @RequestParam(defaultValue = "1") Integer pageNo

    ) {

        return R.ok(processService.getInvolvedMeListWithAllApps(pageSize, pageNo));

    }    /\*\*

     \* 获取系统中所有已发起的流程实例

     \* @param code 表单流程ID（流程定义KEY）

     \* @param pageSize 每页数量

     \* @param pageNo 页码

     \* @param finished 流程是否已经结束

     \* @return 列表数据

     \*/

    @GetMapping("submittedList")

    public Object getSubmittedList(

        @RequestParam(required = false) String code,

        @RequestParam(defaultValue = "20") Integer pageSize,

        @RequestParam(defaultValue = "1") Integer pageNo,

        @RequestParam(required = false) Boolean finished

    ) {

        return R.ok(processService.getUserSubmittedList(pageSize, pageNo, "all", code,  finished));

    }

 @Override

    public List<String> getIsTrueConditions(ProcessConditionResolveParamsVo paramsVo) {

        WflowModelHistorys model = modelService.getModelByDefId(paramsVo.getProcessDfId());

        if (Objects.isNull(model)){

            throw new BusinessException("未找到该流程版本记录");

        }

        ProcessNode<?> node = WfCatchUtil.getCatch(paramsVo.getProcessDfId() + paramsVo.getConditionNodeId(), ProcessNode.class);

        if (Objects.isNull(node)){

            Map<String, ProcessNode<?>> nodeMap = nodeCatchService.reloadProcessByStr(model.getProcess());

            node = nodeMap.get(paramsVo.getConditionNodeId());

            //加载到缓存,2 分钟，为了后面的流程能够快速存取解析

            nodeMap.forEach((k, v) -> {

                if (NodeTypeEnum.CONDITIONS.equals(v.getType()) || NodeTypeEnum.INCLUSIVES.equals(v.getType())){

                    v.getBranchs().forEach(bc -> bc.setChildren(null));

                    WfCatchUtil.putCatch(paramsVo.getProcessDfId() + v.getId(), v, 2 \* 60000);

                }

            });

        }

        List<String> results = new ArrayList<>(node.getBranchs().size());

        String defaultCondition = null;

        for (ProcessNode<?> branch : node.getBranchs()) {

            ConditionProps props = (ConditionProps) branch.getProps();

            if (Objects.isNull(defaultCondition)

                    && ConditionModeEnum.SIMPLE.equals(props.getMode())

                    && CollectionUtil.isEmpty(props.getGroups())){

                //默认条件

                defaultCondition = branch.getId();

                continue;

            }

            Boolean compare = false;

            try {

                compare = uelTools.conditionCompare(props, paramsVo.getContext());

            } catch (Exception e) {

                log.warn("解析流程定义[ID = {}]的条件网关节点[{}]失败：[{}]", paramsVo.getProcessDfId(), paramsVo.getConditionNodeId(), e.getMessage());

            }

            if (compare) {

                log.info("查看返回的分支节点:{}",branch.getId());

                results.add(branch.getId());

                if (!paramsVo.getMultiple()) {

                    break;

                }

            }

        }

        if (CollectionUtil.isEmpty(results)){

            //没有满足的条件，返回默认条件

            results.add(defaultCondition);

        }

        return results;

    }    @Override

    public List<String> getIsTrueCondition(ProcessConditionResolveParamsVo paramsVo) {

        WflowModelHistorys model = modelService.getModelByDefId(paramsVo.getProcessDfId());

        if (Objects.isNull(model)){

            throw new BusinessException("未找到该流程版本记录");

        }

        ProcessNode<?> node = WfCatchUtil.getCatch(paramsVo.getProcessDfId() + paramsVo.getConditionNodeId(), ProcessNode.class);

        if (Objects.isNull(node)){

            Map<String, ProcessNode<?>> nodeMap = nodeCatchService.reloadProcessByStr(model.getProcess());

            node = nodeMap.get(paramsVo.getConditionNodeId());

            //加载到缓存,2 分钟，为了后面的流程能够快速存取解析

            nodeMap.forEach((k, v) -> {

                if (NodeTypeEnum.CONDITIONS.equals(v.getType()) || NodeTypeEnum.INCLUSIVES.equals(v.getType())){

                    v.getBranchs().forEach(bc -> bc.setChildren(null));

                    WfCatchUtil.putCatch(paramsVo.getProcessDfId() + v.getId(), v, 2 \* 60000);

                }

            });

        }

        List<String> results = new ArrayList<>(node.getBranchs().size());

        List<String> results1=new ArrayList<>(node.getBranchs().size());

        String defaultCondition = null;

        for (ProcessNode<?> branch : node.getBranchs()) {

            ConditionProps props = (ConditionProps) branch.getProps();

            if (Objects.isNull(defaultCondition)

                    && ConditionModeEnum.SIMPLE.equals(props.getMode())

                    && CollectionUtil.isEmpty(props.getGroups())){

                //默认条件

                defaultCondition = branch.getId();

                continue;

            }

            Boolean compare1 = false;

            Boolean compare = false;

            try {

                compare = uelTools.conditionCompare(props, paramsVo.getContext());

                compare1 = uelTools.conditionCompare1(props, paramsVo.getContext());

                // log.info("compare1侧说是:{}",compare1);

            } catch (Exception e) {

                log.warn("解析流程定义[ID = {}]的条件网关节点[{}]失败：[{}]", paramsVo.getProcessDfId(), paramsVo.getConditionNodeId(), e.getMessage());

            }

            if (compare) {

                results.add(branch.getId());

                if (!paramsVo.getMultiple()) {

                    break;

                }

            }else if(!compare1) {

                results1.add(branch.getId());

                if (!paramsVo.getMultiple()) {

                    break;

                }

            }

        }

        if (CollectionUtil.isEmpty(results)){

            //没有满足的条件，返回默认条件

            results.add(defaultCondition);

            if(CollectionUtil.isNotEmpty(results)){

                for (String in :results1){

                    results.add(in);

                }

            }

        }

        return results;

}

/\*\*

 \* 业务数据存储服务

 \*/

public class DefaultBusinessDataStorageServiceImpl implements BusinessDataStorageService {    @Autowired

    private RuntimeService runtimeService;    @Autowired

    private HistoryService historyService;    @Override

    public Map<String, Object> getInstanceFormData(String instanceId) {

        return historyService.createHistoricVariableInstanceQuery()

                .processInstanceId(instanceId)

                .variableNameLike("field%")

                .list().stream().collect(Collectors.toMap(

                        HistoricVariableInstance::getVariableName,

                        HistoricVariableInstance::getValue

                ));

    }    @Override

    public Map<String, Object> getInstanceFormFieldsById(String instanceId, String... fieldIds) {

        return null;

    }    @Override

    public void saveFormDataByInstance(String formId, String defId, String instanceId, Map<String, Object> formData) {

        runtimeService.setVariables(instanceId, formData);

    }    @Override

    public void updateFormDataByInstance(String formId, String defId, String instanceId, Map<String, Object> formData) {

        runtimeService.setVariables(instanceId, formData);

    }

}

 /\*\*

     \* 解析表单发起权限控制设置

     \*

     \* @param code        表单流程编号

     \* @param processNode 节点数据

     \*/

    private void reloadModelsPerm(String code, ProcessNode<?> processNode) {

        String companyId = UserUtil.getLoginTenantId();

        //清空之前的权限然后重新设置

        modelPermsMapper.delete(new LambdaQueryWrapper<WflowModelPerms>()

                .eq(WflowModelPerms::getFormId, code));        List<OrgUser> assignedUser = ((RootProps)processNode.getProps()).getAssignedUser();

        if (CollectionUtil.isNotEmpty(assignedUser)) {

            Date time = GregorianCalendar.getInstance().getTime();

            //开始解析所有选择的部门的子部门，深度遍历

            Set<String> deptSet = new HashSet<>();

            Set<String> userSet = new HashSet<>();

            assignedUser.forEach(org -> {

                if ("dept".equals(org.getType())) {

                    deptSet.add(org.getId());

                    deptSet.addAll(orgRepositoryService.getRecursiveSubDept(org.getId()));

                } else {

                    userSet.add(org.getId());

                }

            });

            List<WflowModelPerms> modelPerms = deptSet.stream()

                    .map(d -> WflowModelPerms.builder()

                            .id(IdUtil.objectId())

                            .companyId(companyId)

                            .formId(code)

                            .orgId(d)

                            .permType("dept")

                            .createTime(time)

                            .build()).collect(Collectors.toList());

            modelPerms.addAll(userSet.stream()

                    .map(u -> WflowModelPerms.builder()

                            .id(IdUtil.objectId())

                            .companyId(companyId)

                            .formId(code)

                            .orgId(u)

                            .permType("user")

                            .createTime(time)

                            .build()).collect(Collectors.toList()));

            modelPermsMapper.insertBatch(modelPerms);

        }

    }    @Override

    @Transactional

    public void delProcess(String code) {

        enableProcess(code, false);

        modelsMapper.deleteById(code);

    }

public interface UserDeptOrLeaderService {    /\*\*

     \* 批量获取用户信息

     \* @param userIds 用户ID

     \* @return 用户id->基本信息

     \*/

    Map<String, OrgUser> getUserMapByIds(Collection<String> userIds, String companyId);    /\*\*

     \* 校验用户是否在某部门下

     \* @param userId 用户ID

     \* @param deptId 部门ID

     \* @return 是/否

     \*/

    boolean userIsBelongToDept(String userId, String deptId);    /\*\*

     \* 校验部门是否属于某部门的子部门

     \* @param deptId 子部门ID

     \* @param parentDeptId 父部门ID

     \* @return 是/否

     \*/

    boolean deptIsBelongToDept(String deptId, String parentDeptId);    /\*\*

     \* 批量获取指定部门的主管

     \* @param deptIds 部门ID列表

     \* @return 部门主管列表

     \*/

    Set<String> getLeadersByDept(Collection<String> deptIds);    /\*\*

     \* 获取用户指定级别的主管(用户可能在多个部门下)

     \* @param userId 用户ID

     \* @param userDept 用户的部门ID

     \* @param level 级别

     \* @param skipEmpty 为空是否也算

     \* @return 主管

     \*/

    String getUserLeaderByLevel(String userId, String userDept, int level, boolean skipEmpty);    /\*\*

     \* 查询用户的上级主管，直接向上

     \* @param userDept 所在的直属部门，如果是本级部门领导，则是上级部门开始

     \* @param skipEmpty 为空时是否跳过

     \* @return 主管所在部门

     \*/

    DeptDo getUserDeptLeader(String userDept, boolean skipEmpty);    /\*\*

     \* 获取用户到指定级别为止的所有主管

     \* @param userId 用户ID

     \* @param userDept 用户的部门ID

     \* @param level 终止级别

     \* @param skipEmpty 为空是否也算

     \* @return 领导列表

     \*/

    List<String> getUserLeadersByLevel(String userId, String userDept, Integer level, boolean skipEmpty);    /\*\*

     \* 获取用户的所有直属部门（一个人可能在多个部门下）

     \* 为了在发起流程时候，选择以哪个部门身份发起

     \* @param userId 用户

     \* @return 所有的直属部门

     \*/

    List<DeptVo> getUserDepts(String userId);    /\*\*

     \* 获取指定角色的用户

     \* @param roles 角色列表

     \* @return 符合要求的用户

     \*/

    Set<String> getUsersByRoles(List<String> roles);    /\*\*

     \* 批量获取部门下用户

     \* @param deptIds 部门ID集合

     \* @return 部门下所有用户ID

     \*/

    Set<String> getUsersByDept(Collection<String> deptIds);    /\*\*

     \* 批量获取用户审批代理人，没有代理人则为自己

     \* @param userIds 用户ID集合

     \* @return 用户审批代理人

     \*/

    List<String> replaceUserAsAgent(Collection<String> userIds);    /\*\*

     \* 获取用户管理的部门id

     \* @param userId

     \* @return

     \*/

    Set<String> getManagedDepts(String userId);

}

package com.wflow.workflow.service.impl;import cn.hutool.cache.impl.TimedCache;

import cn.hutool.core.util.ObjectUtil;

import com.alibaba.fastjson2.JSONObject;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.wflow.bean.entity.WflowModels;

import com.wflow.mapper.WflowModelsMapper;

import com.wflow.workflow.WFlowToBpmnCreator;

import com.wflow.workflow.bean.process.ProcessNode;

import com.wflow.workflow.bean.process.enums.NodeTypeEnum;

import com.wflow.workflow.service.ProcessNodeCatchService;import lombok.extern.slf4j.Slf4j;

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

import org.springframework.stereotype.Service;import javax.validation.constraints.NotNull;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.stream.Collectors;/\*\*

 \* 内存实现流程节点缓存服务

 \*

 \*/

@Slf4j

@Service

public class MemoryProcessNodeCatchServiceImpl implements ProcessNodeCatchService {    @Autowired

    private WflowModelsMapper modelsMapper;    //缓存流程定义ID -> (节点ID -> 节点) 快速取数据

    public static final TimedCache<String, Map<String, ProcessNode<?>>> processModelNodeMap = new TimedCache<>(5 \* 60000);    @Override

    public ProcessNode<?> getProcessNodeById(String code, String nodeId) {

        Map<String, ProcessNode<?>> nodeMap = processModelNodeMap.get(code);

        if (null != nodeMap) {

            return nodeMap.get(nodeId);

        }

        return null;

    }    @Override

    public <T> ProcessNode<T> getProcessNodeById(Class<T> clazz, String code, String nodeId) {

        ProcessNode<?> node = getProcessNodeById(code, nodeId);

        if (null != node) {

            return (ProcessNode<T>) node;

        }

        return null;

    }    @Override

    public List<ProcessNode<?>> getTaskNodesByCode(String code) {

        try {

            return processModelNodeMap.get(code).values().stream().filter(n ->

                    NodeTypeEnum.ROOT.equals(n.getType())

                            || NodeTypeEnum.APPROVAL.equals(n.getType())

                            || NodeTypeEnum.CC.equals(n.getType())).collect(Collectors.toList());

        } catch (Exception e) {

            return Collections.emptyList();

        }

    }    @Override

    public ProcessNode<?> findSubNodeByIdFromRoot(@NotNull String nodeId, ProcessNode<?> node) {

        //TODO 此处应当利用LRU缓存实现高效取，日后再修改

        if (null != node && null != node.getId()){

            if (nodeId.equals(node.getId())){

                WFlowToBpmnCreator.coverProps(node);

                return node;

            }else if (NodeTypeEnum.CONCURRENTS.equals(node.getType())

                    || NodeTypeEnum.CONDITIONS.equals(node.getType())

                    || NodeTypeEnum.INCLUSIVES.equals(node.getType())) {

                for (ProcessNode<?> branch : node.getBranchs()) {

                    ProcessNode<?> subNode = findSubNodeByIdFromRoot(nodeId, branch);

                    if (ObjectUtil.isNotNull(subNode)) {

                        WFlowToBpmnCreator.coverProps(node);

                        return subNode;

                    }

                }

            }

            return findSubNodeByIdFromRoot(nodeId, node.getChildren());

        }

        return null;

    }    @Override

    public ProcessNode<?> findSubNodeByIdFromRoot(String nodeId, String code) {

        return findSubNodeByIdFromRoot(nodeId,

                JSONObject.parseObject(modelsMapper.selectById(code)

                        .getProcess(), ProcessNode.class));

    }    @Override

    public ProcessNode<?> findSubNodeByDefIdFromRoot(String nodeId, String defId) {

        return findSubNodeByIdFromRoot(nodeId,

                JSONObject.parseObject(modelsMapper.selectOne(new LambdaQueryWrapper<WflowModels>()

                        .eq(WflowModels::getProcessDefId, defId)).getProcess(), ProcessNode.class));

    }    @Override

    public void setProcessNodes(String code, Map<String, ProcessNode<?>> nodeMap) {

        processModelNodeMap.put(code, nodeMap);

    }    @Override

    public Map<String, ProcessNode<?>> reloadProcessByCode(String code) {

        Map<String, ProcessNode<?>> nodeMap = new LinkedHashMap<>();

        loadProcess(JSONObject.parseObject(modelsMapper.selectById(code).getProcess(), ProcessNode.class), nodeMap);

        processModelNodeMap.put(code, nodeMap);

        return nodeMap;

    }    @Override

    public Map<String, ProcessNode<?>> reloadProcessByStr(String process) {

        Map<String, ProcessNode<?>> nodeMap = new LinkedHashMap<>();

        loadProcess(JSONObject.parseObject(process, ProcessNode.class), nodeMap);

        return nodeMap;

    }    @Override

    public void unloadProcessByCode(String code) {

        processModelNodeMap.remove(code);

    }    private void loadProcess(ProcessNode<?> node, Map<String, ProcessNode<?>> nodeMap) {

        if (null != node && null != node.getId()) {

            WFlowToBpmnCreator.coverProps(node);

            nodeMap.put(node.getId(), node);

            if (NodeTypeEnum.CONCURRENTS.equals(node.getType())

                    || NodeTypeEnum.CONDITIONS.equals(node.getType())

                    || NodeTypeEnum.INCLUSIVES.equals(node.getType())) {

                node.getBranchs().forEach(n -> loadProcess(n, nodeMap));

            }

            loadProcess(node.getChildren(), nodeMap);

        }

    }}

@Slf4j

@Service

public class ProcessInstanceServiceImpl implements ProcessInstanceService { @Autowired

private OrgRepositoryService orgRepositoryService; @Autowired

private WflowModelsMapper modelsMapper; @Autowired

private WflowModelHistorysMapper modelHistorysMapper; @Autowired

private ProcessNodeCatchService nodeCatchService; @Autowired

private RepositoryService repositoryService; @Autowired

private RuntimeService runtimeService; @Autowired

private ProcessModelService processModelService; @Autowired

private HistoryService historyService; @Autowired

private TaskService taskService; @Autowired

private UserDeptOrLeaderService userDeptOrLeaderService; @Autowired

private ProcessTaskService processTaskService; @Autowired

private WflowCcTasksMapper ccTasksMapper; @Autowired

private FormService formService; @Autowired

private FormDataMapper formDataMapper; // @Autowired

// public Jedis jedis; @Autowired

public FormServiceImpl formServiceimp; @Autowired

public WechatWorkService wechatWorkService; @Autowired

private MenuService menuService; @Autowired

private WflowUsersMapper usersMapper; @Autowired

public CgManagerConfig cgManagerConfig; // @Autowired

// public JedisPool jedisPool; @Autowired

public Jedis jedis; // @PostConstruct

// public void initJedis() {

// this.jedis = jedisPool.getResource();

// } @Override

@Transactional

public String startProcess(String defId, ProcessStartParamsVo params, String userId, String companyId) {

try {

String currentTenantId = "";

if (companyId == null || companyId.equals("")) {

currentTenantId = UserUtil.getLoginTenantId();

} else {

currentTenantId = companyId;

} String userName = orgRepositoryService.getUserById(currentTenantId, userId).getUserName(); WflowModels wflowModel = modelsMapper.selectOne(new LambdaQueryWrapper<WflowModels>().eq(WflowModels::getProcessDefId, defId));

Map<String, Object> settings = JSONObject.parse(wflowModel.getSettings());

String businessKey = this.getBusinessKey(settings.get("prefix").toString(), 5); if (wflowModel.getIsFlow() == true) { Map<String, Object> processVar = new HashMap<>(); // List<Form> formItems = JSONArray.parseArray(wflowModel.getFormItems(), Form.class); // if (wflowModel.getFormId().equals("wf65bf20b0e4b0de990e1f0c2e") || wflowModel.getFormId().equals("wf65bf1ff7e4b0de990e1f0c2d") || wflowModel.getFormId().equals("wf65dee870e4b0e59ac55f67b3") || wflowModel.getFormId().equals("wf65e18ba6e4b0e59ac55f67b9")) {

// log.info("特殊wflowModel.getFormId(): {}",wflowModel.getFormId());

Map<String, Object> newFormData = formService.convertFieldValueToString(wflowModel, params.getFormData());

processVar.putAll(newFormData);

// } else {

// processVar.putAll(params.getFormData());

// } processVar.putAll(params.getProcessUsers()); ProcessDefinition processDefinition = repositoryService.createProcessDefinitionQuery()

.processDefinitionId(defId).latestVersion().singleResult(); // log.info("processDefinition: {}", ObjectUtil.isNotNull(processDefinition));

// log.info("processDefinition.isSuspended: {}", processDefinition.isSuspended()); if (ObjectUtil.isNotNull(processDefinition) && processDefinition.isSuspended()) {

throw new BusinessException("流程未启用，请先启用");

}

// 设置发起的人员及部门信息

String deptName = orgRepositoryService.getDeptById(params.getDeptId()).getDeptName(); processVar.put("owner", ProcessInstanceOwnerDto.builder()

.owner(userId)

.ownerName(userName)

.ownerDeptId(params.getDeptId())

.ownerDeptName(deptName)

.build()); // WflowModels wflowModels = modelsMapper.selectOne(new

// QueryWrapper<WflowModels>().eq("process\_def\_id", defId)); Map<String, ProcessNode<?>> nodeMap = nodeCatchService.reloadProcessByStr(wflowModel.getProcess()); Map<String, Object> propsMap = nodeMap.entrySet().stream().collect(Collectors.toMap(Map.Entry::getKey,

v -> null == v.getValue().getProps() ? new HashMap<>() : v.getValue().getProps())); // 将表单及流程配置设置为变量，跟随版本

processVar.put(WflowGlobalVarDef.WFLOW\_NODE\_PROPS, propsMap);

processVar.put(WflowGlobalVarDef.WFLOW\_FORMS, JSONArray.parseArray(wflowModel.getFormItems(), Form.class)); // processVar.put("\_FLOWABLE\_SKIP\_EXPRESSION\_ENABLED", false);

// processVar.put("approve", true);

processVar.put(WflowGlobalVarDef.INITIATOR, userId); // processVar.put("tenantId", currentTenantId); // 这样做貌似无效果，变量表不会多INITIATOR变量，但是流程表发起人有效

Authentication.setAuthenticatedUserId(userId); // ProcessInstance processInstance = runtimeService.startProcessInstanceById(defId, businessKey,

// processVar); ProcessInstance processInstance = runtimeService.createProcessInstanceBuilder().processDefinitionId(defId).tenantId(currentTenantId).businessKey(businessKey).variables(processVar).start(); log.info("启动 {} 流程实例 {} 成功", processInstance.getProcessDefinitionName(),

processInstance.getProcessInstanceId());

// 自动完成发起人节点任务

Task rootTask = taskService.createTaskQuery().processInstanceId(processInstance.getProcessInstanceId()).active().singleResult();

if (Objects.nonNull(rootTask)) {

taskService.complete(rootTask.getId());

}

Authentication.setAuthenticatedUserId(null); String jsonString = JSON.toJSONString(params.getFormData());

Number rows = formDataMapper.save(processInstance.getProcessInstanceId(), currentTenantId, businessKey,

wflowModel.getFormId(), params.getDeptId(), jsonString, new Date(), userId, userName);

// log.info("create formData success rows: {}", rows); return processInstance.getProcessInstanceId();

} else {

String uuid = IdUtil.randomUUID();

String jsonString = JSON.toJSONString(params.getFormData());

Number rows = formDataMapper.save(uuid, currentTenantId, businessKey, wflowModel.getFormId(), params.getDeptId(),

jsonString, new Date(), userId, userName);

// log.info("create formData success rows: {}", rows);

return uuid;

}

} catch (Exception e) {

// TODO: handle exception

log.error("---------------- start process error: {}", e.toString());

e.printStackTrace();

return null;

}

} public String padLeftZeros(String inputString, int length) {

if (inputString.length() >= length) {

return inputString;

}

StringBuilder sb = new StringBuilder();

while (sb.length() < length - inputString.length()) {

sb.append('0');

}

sb.append(inputString); return sb.toString();

} public String getBusinessKey(String prefix, int retry) {

try {

String businessKey = prefix;

String date = DateUtil.format(new Date(), "yyyyMMdd");

String today = DateUtil.format(new Date(), "yyyy-MM-dd");

String value = this.jedis.get(today);

if (value == null || value.equals("")) {

this.jedis.set(today, "0");

}

// log.info("jedis incrBy({}, 1)", today);

long number = this.jedis.incrBy(today, 1);

// log.info("jedis incrBy({}, 1), number: {}", today, number);

// log.info("getBusinessKey number: {}", number);

String key = businessKey + date + this.padLeftZeros(String.valueOf(number), 4);

return key;

} catch (Exception e) {

e.printStackTrace();

++retry;

log.info("getBusinessKey() failed, retry: {}", retry);

if (retry <= 10) {

return this.getBusinessKey(prefix, retry);

} else {

String millisecond = String.valueOf(System.currentTimeMillis());

return prefix + "-" + millisecond;

}

}

} @Override

public String getBeforeTask(String instanceId, String task) {

return null;

} @Override

@Transactional

public void delProcessInstance(String instanceId) {

// 删除流程实例

try {

runtimeService.deleteProcessInstance(instanceId, "删除");

} catch (Exception ignored) {}

// 删除流程历史实例

historyService.deleteHistoricProcessInstance(instanceId);

//删除的同时相关抄送表数据也要删除，不然分页数据会错

ccTasksMapper.delete(new LambdaQueryWrapper<WflowCcTasks>().eq(WflowCcTasks::getInstanceId, instanceId));

log.info("删除流程实例[{}]成功", instanceId);

} @Override

// @Transactional

public void batchDelProcessInstanceByFormId(String formId) {

// log.info("formId.size(): {}", formId);

// 删除流程实例

HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery();

query.processDefinitionKey(formId); List<HistoricProcessInstance> historicProcessInstances = query.orderByProcessInstanceStartTime().desc()

.orderByProcessInstanceEndTime().desc()

.list(); // log.info("historicProcessInstances.size(): {}", historicProcessInstances.size());

for (HistoricProcessInstance instance: historicProcessInstances) {

String instanceId = instance.getId();

runtimeService.deleteProcessInstance(instanceId, "删除");

// 删除流程历史实例

historyService.deleteHistoricProcessInstance(instanceId);

//删除的同时相关抄送表数据也要删除，不然分页数据会错

ccTasksMapper.delete(new LambdaQueryWrapper<WflowCcTasks>().eq(WflowCcTasks::getInstanceId, instanceId));

log.info("删除流程实例[{}]成功", instanceId);

}

} @Override

public ProcessProgressVo getInstanceProgress(String nodeId, String id) { FormData form = formDataMapper.queryFormDataDetail(id); if (ObjectUtil.isNull(form)) {

return null;

} String instanceId = form.getId();

// String instanceId = form.getProcessInstanceId(); JSONObject json = JSONObject.parseObject(form.getData()); Map<String, Object> formData = new HashMap<>();

// log.info("formData: {}",formData);

// if (form.getFormId().equals("wf65bf20b0e4b0de990e1f0c2e") || form.getFormId().equals("wf65bf1ff7e4b0de990e1f0c2d") || form.getFormId().equals("wf65225669e4b0802de9d7bdc4")) { for (Map.Entry<String, Object> entry : json.entrySet()) { // if(entry.getKey().equals("field3139346960176")||entry.getKey().equals("field3016195976779")){

// log.info("entry.getKey():{}, entry.getValue():{}", entry.getKey(), entry.getValue());

// }

// log.info("key: {}, value: {}", entry.getKey(), entry.getValue()); if(entry.getValue() != null){ Object value = null; try {

if (entry.getValue() instanceof Map) {

// log.info("entry.getValue() instanceof Map");

value = entry.getValue();

} else {

JSONObject v = JSONObject.parseObject(entry.getValue().toString());

value = v;

}

} catch (Exception e) {

// log.info("JSONObject.parseObject() failed, entry.getKey(): {}, entry.getValue(): {}", entry.getKey(), entry.getValue());

value = entry.getValue();

// formData.put(entry.getKey(), value);

}

// log.info("map.put(), key: {}, value: {}", entry.getKey(), value);

formData.put(entry.getKey(), value);

}else{

formData.put(entry.getKey(), null);

}

} // } else { // for (Map.Entry<String, Object> entry : json.entrySet()) {

// formData.put(entry.getKey(), entry.getValue());

// }

// }

// log.info("field3016195976779:{},field3139346960176:{}", formData.get("field3016195976779"),formData.get("field3139346960176")); // log.info("field3016195976779:{},field3139346960176:{}", formData.get("field3016195976779"),formData.get("field3139346960176"));

String companyId = UserUtil.getLoginTenantId(); // log.info("getInstanceProgress() instanceId: {}", instanceId); HistoricProcessInstance instance = historyService.createHistoricProcessInstanceQuery().processInstanceId(instanceId).singleResult(); log.info("instance: {}", instance); // log.info("instance.getProcessDefinitionId():{}",instance.getProcessDefinitionId());

//优化查询，把之前查好几次的一次性查出来然后再分类

List<HistoricVariableInstance> variables = historyService.createHistoricVariableInstanceQuery()

.processInstanceId(instanceId).executionId(instanceId).list();

//数据分类 表单配置及数据、审批任务结果、

ProcessInstanceOwnerDto owner = null;

List<Form> forms = Collections.emptyList();

// Map<String, Object> formDatas = formData;

Map<String, ProcessHandlerParamsVo.Action> approvalResults = new HashMap<>();

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

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

//遍历所有变量，将数据分类

for (HistoricVariableInstance var : variables) {

if (var.getVariableName().startsWith("approve\_")) {

approvalResults.put(var.getVariableName(), (ProcessHandlerParamsVo.Action) var.getValue());

} else if (var.getVariableName().startsWith("field")) {

// formDatas.put(var.getVariableName(), var.getValue());

} else if (var.getVariableName().startsWith("sign\_")) {

signs.put(var.getVariableName().substring(5), String.valueOf(var.getValue()));

} else if (WflowGlobalVarDef.WFLOW\_FORMS.equals(var.getVariableName())) {

forms = this.getNewestForm(instance.getProcessDefinitionId());

} else if (WflowGlobalVarDef.WFLOW\_NODE\_PROPS.equals(var.getVariableName())) {

nodeProps = (Map<String, Object>)var.getValue();

} else if ("owner".equals(var.getVariableName())) {

owner = (ProcessInstanceOwnerDto) var.getValue();

}

}

ProcessNode<?> currentNode = null;

WflowModelHistorys modelHistory = null;

String modelFormConfig = null;

ProcessProgressVo.InstanceExternSetting externSetting = null;

//log.info("field3016195976779:{},field3139346960176:{}", formData.get("field3016195976779"),formData.get("field3139346960176"));

if (StrUtil.isNotBlank(nodeId)) {

// 搜索当前版本流程的配置 modelHistory = modelHistorysMapper.selectOne(new LambdaQueryWrapper<>(WflowModelHistorys.builder()

.processDefId(instance.getProcessDefinitionId()).version(instance.getProcessDefinitionVersion()).build())); // log.info("modelHistory是否是最新的: {}", modelHistory); if (modelHistory == null) {

String oldDefId = instance.getProcessDefinitionId();

// log.info("oldDefId: {}", oldDefId);

modelHistory = modelHistorysMapper.selectOne(new QueryWrapper<WflowModelHistorys>().eq("process\_def\_id", oldDefId)); if(modelHistory == null){

modelHistory = processModelService.getLastVersionModel(form.getFormId());

}

}

currentNode = nodeCatchService.reloadProcessByStr(modelHistory.getProcess()).get(nodeId);

modelFormConfig = modelHistory.getFormConfig();

externSetting = getExternSetting(modelHistory.getSettings(), instance.getStartUserId());

}

// log.info("currentNode: {}", currentNode);

//log.info("field3016195976779:{},field3139346960176:{}", formData.get("field3016195976779"),formData.get("field3139346960176"));

UserDo users = orgRepositoryService.getUserById(companyId, instance.getStartUserId()); OrgUser startUser = OrgUser.builder().id(users.getUserId()).name(users.getUserName()).avatar(users.getAvatar()).build(); List<ProcessProgressVo.ProgressNode> taskRecords = getHisTaskRecords(instanceId, nodeProps, approvalResults, signs);

// log.info("taskRecords:{}",getCcTaskRecords(instanceId));

taskRecords.addAll(getCcTaskRecords(instanceId)); if (ObjectUtil.isNull(instance.getEndTime())) {

taskRecords.addAll(processTaskService.getFutureTask(instanceId));

}

taskRecords = taskRecords.stream()

.sorted(Comparator.comparing(ProcessProgressVo.ProgressNode::getStartTime))

.collect(Collectors.toList());

ProcessProgressVo.ProgressNode progressNode = taskRecords.stream()

.filter(n -> "root".equals(n.getNodeId()) && Objects.isNull(n.getResult()))

.findFirst().orElse(null); if (Objects.nonNull(progressNode)) {

// 辅助前端UI构建，第一条一般都是

progressNode.setResult(ProcessHandlerParamsVo.Action.agree);

progressNode.setNodeType(NodeTypeEnum.ROOT);

} else {

// 没有就进行构建，兼容之前的流程

taskRecords.add(0, ProcessProgressVo.ProgressNode.builder()

.nodeId("root")

.name("提交申请")

.user(startUser)

.nodeType(NodeTypeEnum.ROOT)

.startTime(instance.getStartTime())

.finishTime(instance.getStartTime())

.taskId("root")

.result(ProcessHandlerParamsVo.Action.agree)

.build());

}

//提取全量表单数据

// if (StrUtil.isBlank(modelFormConfig)) {

// modelFormConfig = modelHistory.getFormConfig();

// externSetting = getExternSetting(modelHistory.getSettings(), instance.getStartUserId());

// }

ProcessResultEnum processResult = ProcessResultEnum.resolveResult(instance.getEndActivityId());

// List<ProcessProgressVo.ProgressNode> taskRecords3=new CopyOnWriteArrayList<>();

//判断流程是否结束

// int index=taskRecords.size()-1;

// ProcessProgressVo.ProgressNode progressNodelist=taskRecords.get(index);

// Date endtime=progressNodelist.getFinishTime();

// if(endtime != null){

// for (ProcessProgressVo.ProgressNode progressNode1:taskRecords){

// taskRecords3.add(progressNode1);

// }

// }else {

// //补充流程后续节点

// WflowModels modelHistory1=modelsMapper.getModelsByFormId(form.getFormId());

// List<ProcessProgressVo.ProgressNode> list=new CopyOnWriteArrayList<>();

// JSONObject jsonObject= JSONObject.parseObject(modelHistory1.getProcess());

// List<ProcessProgressVo.ProgressNode> taskRecords1 =dG(jsonObject,list); // log.info("taskRecords1:{}",taskRecords1);

// JSONObject jsonObject1=new JSONObject();

// for (ProcessProgressVo.ProgressNode progressNode2:taskRecords1){

// int i=0;

// for (ProcessProgressVo.ProgressNode progressNode1:taskRecords){

// if(progressNode1.getNodeId().equals(progressNode2.getNodeId())){

// if(StrUtil.isNotBlank(progressNode1.getTaskId())){

// jsonObject1.put(progressNode1.getTaskId(),progressNode1.getNodeId());

// taskRecords3.add(progressNode1);

// i=i+1;

// } // }else {

// if(i==0){

// String type=null;

// type=dGT(jsonObject,progressNode1.getNodeId(),type);

// if(!("CONDITIONS".equals(type))||!("INCLUSIVES".equals(type))){

// if(!jsonObject1.containsKey(progressNode1.getTaskId())||!jsonObject1.getString(progressNode1.getTaskId()).equals(progressNode1.getNodeId())||jsonObject1.size()==taskRecords.size()){

// taskRecords3.add(progressNode2);

// i=i+1;

// }

// }else {

// if(StrUtil.isNotBlank(progressNode1.getTaskId())){

// taskRecords3.add(progressNode1);

// jsonObject1.put(progressNode1.getTaskId(),progressNode1.getNodeId());

// }

// }

// }

// } // }

// }

// }

//log.info("field3016195976779:{},field3139346960176:{}", formData.get("field3016195976779"),formData.get("field3139346960176"));

List<Form> formItems = formService.filterFormAndDataByPermConfig((List<Form>) forms, formData, currentNode);

// log.info("查看是否为空currentNode：{}",currentNode);

//log.info("field3016195976779:{},field3139346960176:{}", formData.get("field3016195976779"),formData.get("field3139346960176"));

return ProcessProgressVo.builder()

.instanceId(instanceId)

.businessKey(instance.getBusinessKey())

.version(instance.getProcessDefinitionVersion())

.formItems(formItems)

.formConfig(JSONObject.parseObject(modelFormConfig))

.formData(formData)

.processDefName(instance.getProcessDefinitionName())

.staterUser(startUser)

.starterDept(null == owner ? null : owner.getOwnerDeptName())

.status(processResult.getDesc())

.result(processResult)

.startTime(instance.getStartTime())

.finishTime(instance.getEndTime())

.process(this.getProcess1(instance))

.progress(taskRecords)

.operationPerm(getOperationPerm(currentNode))

.externSetting(externSetting)

.processDefId(instance.getProcessDefinitionId())

.build();

} /\*\*

\* 递归组建全套历史节点

\* @param jsonObject 历史模型

\* @param list 重组后数据

\* @return

\*/

public List<ProcessProgressVo.ProgressNode> dG(JSONObject jsonObject,List<ProcessProgressVo.ProgressNode> list) {

if (jsonObject.containsKey("children")) {

if (jsonObject.getString("type").equals("APPROVAL") || jsonObject.getString("type").equals("ROOT") || jsonObject.getString("type").equals("TASK")||jsonObject.getString("type").equals("CC")) {

ProcessProgressVo.ProgressNode progressNode = new ProcessProgressVo.ProgressNode();

progressNode.setNodeId(jsonObject.getString("id"));

progressNode.setTaskId("");

progressNode.setNodeType(NodeTypeEnum.valueOf(jsonObject.getString("type")));

progressNode.setName(jsonObject.getString("name"));

JSONObject jsonObject1 = JSONObject.parseObject(jsonObject.getString("props"));

if(jsonObject1.containsKey("mode")){

progressNode.setApprovalMode(ApprovalModeEnum.valueOf(jsonObject1.getString("mode")));

}

progressNode.setAction(ProcessHandlerParamsVo.Action.valueOf("await"));

progressNode.setSignature("");

List<TaskCommentVo> list1 = new ArrayList<>();

progressNode.setComment(list1);

//progressNode.setResult(ProcessHandlerParamsVo.Action.valueOf("await"));

if(jsonObject1.containsKey("nobody")){

if(jsonObject1.getString("assignedType").equals("ASSIGN\_USER")){

JSONArray jsonArray1 = jsonObject1.getJSONArray("assignedUser");

if (jsonArray1.size() > 0) {

for (Object A : jsonArray1) {

if (A instanceof JSONObject) {

OrgUser orgUser =zY(A);

progressNode.setUser(orgUser);

list.add(progressNode);

}else {

log.info("无法转义:{}",A);

}

}

}else{

list.add(progressNode);

}

}else{

//获取用户

JSONObject jsonObject2 = jsonObject1.getObject("nobody", JSONObject.class);

JSONArray jsonArray = jsonObject2.getJSONArray("assignedUser");

//判断数据是否存在

if (jsonArray.size() > 0) {

for (Object A : jsonArray) {

if (A instanceof JSONObject) {

OrgUser orgUser =zY(A);

progressNode.setUser(orgUser);

list.add(progressNode);

}

}

}else{

list.add(progressNode);

}

}

}else{

JSONArray jsonArray1 = jsonObject1.getJSONArray("assignedUser");

if (jsonArray1.size() > 0) {

for (Object A : jsonArray1) {

if (A instanceof JSONObject) {

OrgUser orgUser =zY(A);

progressNode.setUser(orgUser);

list.add(progressNode);

}else {

log.info("无法转义:{}",A);

}

}

}else{

list.add(progressNode);

}

}

if (jsonObject.containsKey("children")) {

JSONObject jsonObject3 = jsonObject.getJSONObject("children");

dG(jsonObject3, list);

}

} else if (jsonObject.getString("type").equals("CONCURRENTS")) {

JSONArray jsonObject1 = jsonObject.getJSONArray("branchs");

for (Object o : jsonObject1) {

JSONObject jsonObject2 = (JSONObject) o;

if (jsonObject2.containsKey("children")) {

JSONObject jsonObject3 = jsonObject2.getJSONObject("children");

dG(jsonObject3, list);

}

}

if (jsonObject.containsKey("children")) {

JSONObject jsonObject3 = jsonObject.getJSONObject("children");

dG(jsonObject3, list);

}

} else if(jsonObject.getString("type").equals("CONDITIONS") || jsonObject.getString("type").equals("INCLUSIVES")){

JSONArray jsonObject1 = jsonObject.getJSONArray("branchs");

for (Object o : jsonObject1) {

JSONObject jsonObject2 = (JSONObject) o;

if(jsonObject2.getString("name").equals("默认条件")){

if (jsonObject2.containsKey("children")) {

JSONObject jsonObject3 = jsonObject2.getJSONObject("children");

dG(jsonObject3, list);

}

}

}

if (jsonObject.containsKey("children")) {

JSONObject jsonObject3 = jsonObject.getJSONObject("children");

dG(jsonObject3, list);

}

} else if(jsonObject.getString("type").equals("EMPTY")){

if (jsonObject.containsKey("children")) {

JSONObject jsonObject3 = jsonObject.getJSONObject("children");

dG(jsonObject3, list);

}

}

}

return list;

} /\*\*

\* 解析用户信息

\* @param A

\* @return

\*/

public OrgUser zY(Object A){

OrgUser orgUser = new OrgUser();

if (A instanceof JSONObject) {

JSONObject jsonUser = (JSONObject) A;

orgUser.setId(jsonUser.getString("id"));

orgUser.setName(jsonUser.getString("name"));

orgUser.setAvatar(jsonUser.getString("avatar"));

orgUser.setCompanyId(jsonUser.getString("companyId"));

orgUser.setType(jsonUser.getString("type"));

orgUser.setGender(jsonUser.getString("gender"));

orgUser.setPhone(jsonUser.getString("phone"));

orgUser.setEmail(jsonUser.getString("email"));

orgUser.setPosition(jsonUser.getString("position"));

}

return orgUser;

} /\*\*

\* 递归判断是否为某条条件分支

\* @param jsonObject 模型数据

\* @param nodeId 节点id

\* @param type 节点类型

\* @return

\*/

public String dGT(JSONObject jsonObject,String nodeId,String type) {

if (jsonObject.containsKey("children")) {

if (jsonObject.getString("type").equals("APPROVAL") || jsonObject.getString("type").equals("root") || jsonObject.getString("type").equals("TASK")) {

if(jsonObject.getString("id")!=nodeId){

if (jsonObject.containsKey("children")) {

type=jsonObject.getString("type");

JSONObject jsonObject3 = jsonObject.getJSONObject("children");

dGT(jsonObject3,nodeId,type);

}

}else {

return type;

} } else if (jsonObject.getString("type").equals("CONCURRENTS")) {

JSONArray jsonObject1 = jsonObject.getJSONArray("branchs");

for (Object o : jsonObject1) {

JSONObject jsonObject2 = (JSONObject) o;

if (jsonObject2.containsKey("children")) {

type=jsonObject.getString("type");

JSONObject jsonObject3 = jsonObject2.getJSONObject("children");

dGT(jsonObject3,nodeId,type);

}

}

} else if(jsonObject.getString("type").equals("CONDITIONS") || jsonObject.getString("type").equals("INCLUSIVES")){

JSONArray jsonObject1 = jsonObject.getJSONArray("branchs");

for (Object o : jsonObject1) {

JSONObject jsonObject2 = (JSONObject) o;

if(jsonObject2.getString("name")!="默认条件"){

if (jsonObject2.containsKey("children")) {

type=jsonObject.getString("type");

JSONObject jsonObject3 = jsonObject2.getJSONObject("children");

dGT(jsonObject3,nodeId,type);

}

}

}

}

}

return null;

} public List<Form> getNewestForm(String defId) { // String oldDefId = instance.getProcessDefinitionId();

String oldDefId = defId;

// log.info("oldDefId: {}", oldDefId); WflowModelHistorys historyModel = modelHistorysMapper

.selectOne(new QueryWrapper<WflowModelHistorys>().eq("process\_def\_id", oldDefId)); String formId = historyModel.getFormId();

// log.info("formId: {}", formId); WflowModels wflowModels = modelsMapper.selectOne(new QueryWrapper<WflowModels>().eq("form\_id", formId)); // String defId = wflowModels.getProcessDefId();

// log.info("defId: {}", defId); // HistoricProcessInstance newInstance =

// historyService.createHistoricProcessInstanceQuery().processDefinitionId(defId).singleResult(); // HistoricVariableInstance forms =

// historyService.createHistoricVariableInstanceQuery()

// .processInstanceId(newInstance.getId()).variableName(WflowGlobalVarDef.WFLOW\_FORMS).singleResult(); // JSONObject obj = JSONObject.parseObject(wflowModels.getFormItems());

String formItems = "";

if (wflowModels != null) {

formItems = wflowModels.getFormItems();

} else {

formItems = historyModel.getFormItems();

}

List<Form> forms = JSONArray.parseArray(formItems, Form.class);

return forms;

} public String getProcess(HistoricProcessInstance instance) { String oldDefId = instance.getProcessDefinitionId();

// log.info("oldDefId: {}", oldDefId); WflowModelHistorys historyModel = modelHistorysMapper

.selectOne(new QueryWrapper<WflowModelHistorys>().eq("process\_def\_id", oldDefId)); String formId = historyModel.getFormId();

// log.info("formId: {}", formId); WflowModels wflowModels = modelsMapper.selectOne(new QueryWrapper<WflowModels>().eq("form\_id", formId));

// if(StrUtil.isNotBlank(historyModel.getFormId())){

// return historyModel.getProcess();

// }

return wflowModels.getProcess();

}

public String getProcess1(HistoricProcessInstance instance) { String oldDefId = instance.getProcessDefinitionId();

// log.info("oldDefId: {}", oldDefId); WflowModelHistorys historyModel = modelHistorysMapper

.selectOne(new QueryWrapper<WflowModelHistorys>().eq("process\_def\_id", oldDefId)); String formId = historyModel.getFormId();

// log.info("formId: {}", formId); WflowModels wflowModels = modelsMapper.selectOne(new QueryWrapper<WflowModels>().eq("form\_id", formId));

if(StrUtil.isNotBlank(historyModel.getFormId())){

return historyModel.getProcess();

}

return wflowModels.getProcess();

} /\*\*

\* 获取流程的审批历史记录

\*

\* @param instanceId 审批实例ID

\* @param nodeProps 节点设置

\* @return 历史记录列表

\*/

public List<ProcessProgressVo.ProgressNode> getHisTaskRecords(String instanceId, Map<String, Object> nodeProps,

Map<String, ProcessHandlerParamsVo.Action> varMap,

Map<String, String> signs) {

List<HistoricActivityInstance> list = historyService.createHistoricActivityInstanceQuery()

.processInstanceId(instanceId).orderByHistoricActivityInstanceStartTime().asc().list();

Set<String> userSet = new HashSet<>();

String companyId = UserUtil.getLoginTenantId();

// 获取节点处理结果

Map<String, List<TaskCommentVo>> commentMap = new HashMap<>();

// 统一处理所有评论数据，省的多次查询

List<TaskCommentVo> cmvos = taskService.getProcessInstanceComments(instanceId).stream().map(comment -> {

userSet.add(comment.getUserId());

TaskCommentVo commentVo = TaskCommentVo.builder()

.id(comment.getId())

.taskId(comment.getTaskId())

.commentType(comment.getType())

.type("COMMENT")

.createTime(comment.getTime())

.user(OrgUser.builder().id(comment.getUserId()).build())

.build();

ProcessHandlerParamsVo.ProcessComment processComment = JSONObject.parseObject(comment.getFullMessage(), ProcessHandlerParamsVo.ProcessComment.class);

commentVo.setText(processComment.getText());

commentVo.setAttachments(processComment.getAttachments());

return commentVo;

}).collect(Collectors.toList()); cmvos.forEach(cm -> {

// 把评论数据按照task进行归类

String taskId = Optional.ofNullable(cm.getTaskId()).orElse(instanceId);

List<TaskCommentVo> vos = commentMap.computeIfAbsent(taskId, k -> new ArrayList<>());

vos.add(cm);

});

// log.info("cmvos: {}", cmvos); List<ProcessProgressVo.ProgressNode> progressNodes = list.stream().filter(his -> ObjectUtil.isNotNull(his.getTaskId())).map(his -> {

Object props = nodeProps.get(his.getActivityId());

ApprovalModeEnum approvalMode = null;

if (props instanceof ApprovalProps) {

approvalMode = ((ApprovalProps) props).getMode();

}

userSet.add(his.getAssignee());

List<TaskCommentVo> commentVos = commentMap.getOrDefault(his.getTaskId(), Collections.emptyList());

return ProcessProgressVo.ProgressNode.builder()

.nodeId(his.getActivityId())

.name(his.getActivityName())

.nodeType(NodeTypeEnum.APPROVAL)

.user(OrgUser.builder().id(his.getAssignee()).build())

.startTime(his.getStartTime())

.finishTime(his.getEndTime())

.taskId(his.getTaskId())

.approvalMode(approvalMode)

.signature(signs.get(his.getTaskId()))

.comment(commentVos)

.result(varMap.get("approve\_" + his.getTaskId()))

.build();

}).collect(Collectors.toList()); //将非任务类的评论转换成流程节点

progressNodes.addAll(commentMap.getOrDefault(instanceId, Collections.emptyList()).stream().map(cm ->

ProcessProgressVo.ProgressNode.builder()

.nodeId(cm.getId())

.name("参与评论")

.user(cm.getUser())

.startTime(cm.getCreateTime())

.finishTime(cm.getCreateTime())

.taskId(cm.getId())

.comment(CollectionUtil.newArrayList(cm))

.result(ProcessHandlerParamsVo.Action.comment)

.build()).collect(Collectors.toList()));

if (CollectionUtil.isNotEmpty(userSet)) {

Map<String, OrgUser> map = userDeptOrLeaderService.getUserMapByIds(userSet, companyId);

progressNodes.forEach(n -> {

if (WflowGlobalVarDef.WFLOW\_TASK\_AGRRE.equals(n.getUser().getId())

|| WflowGlobalVarDef.WFLOW\_TASK\_REFUSE.equals(n.getUser().getId())) {

n.setUser(WflowGlobalVarDef.SYS);

} else {

n.setUser(map.get(n.getUser().getId()));

n.getComment().forEach(c -> c.setUser(map.get(c.getUser().getId())));

}

});

}

return progressNodes;

} @Override

public List<Task> getProcessInstanceTaskList(String instanceId) {

return taskService.createTaskQuery().processInstanceId(instanceId).active().list();

} @Override

public Page<ProcessInstanceVo> getUserSubmittedList(Integer pageSize, Integer pageNo, String startUser, String code, Boolean finished) { String currentUserId = UserUtil.getLoginUserId(); // log.info("currentUserId: {}", currentUserId); String currentTenantId = UserUtil.getLoginTenantId(); HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery(); if (currentTenantId.equals("fbd9654e-4d19-4d28-b525-e7477a8b28b9")) { query.or().processInstanceTenantId(currentTenantId).processInstanceTenantId("default").endOr(); } else {

query.processInstanceTenantId(currentTenantId);

} if (code != null && !code.equals("")) {

query.processDefinitionKey(code);

} if (startUser != null && startUser.equals("all")) { } else {

query.startedBy(currentUserId);

}

// query.notDeleted(); // Executor.builder()

// .ifNotBlankNext(code, query::processDefinitionKey)

// .ifNotBlankNext(currentUserId, query::involvedUser)

// // .ifNotBlankNext(processInstanceIds, query::processInstanceIds)

// .ifTrueNext(Boolean.FALSE.equals(finished), query::unfinished); List<HistoricProcessInstance> historicProcessInstances = query.orderByProcessInstanceStartTime().desc()

.orderByProcessInstanceEndTime().desc()

.listPage(pageSize \* (pageNo - 1), pageSize); // log.info("historicProcessInstances.size(): {}", historicProcessInstances.size()); Page<ProcessInstanceVo> page = new Page<>();

page.setTotal(query.count());

page.setCurrent(pageNo);

page.setSize(pageSize);

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

page.setRecords(this.getInstances(instanceNodeMap, historicProcessInstances, currentUserId));

return page;

} @Override

public Page<ProcessInstanceVo> getInvolvedMeList(Integer pageSize, Integer pageNo, String code, String keywords, String ids) { String currentUserId = UserUtil.getLoginUserId(); // log.info("currentUserId: {}", currentUserId); HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery();

if (ids != null) { String[] idArray = ids.split(","); Set<String> idSet = new HashSet<>(); for (String id : idArray) {

idSet.add(id);

}

query.processInstanceIds(idSet);

} else {

// log.info("code: {}", code); // String currentTenantId = UserUtil.getLoginTenantId(); // log.info("currentTenantId: {}", currentTenantId); List<String> adminIds = formService.getAdminIds(code);

// log.info("adminIds: {}", adminIds.toString());

WflowModels model = processModelService.getWflowModel(code);

Map<String, Object> settings = JSONObject.parse(model.getSettings());

int permission = (int) settings.get("permission"); List<String> memberIds = new ArrayList<>();

List<?> memberGroupIds = (List<?>) (settings.get("memberGroupIds"));

if (memberGroupIds.size() != 0) {

memberGroupIds.forEach(memberGroupId -> {

MemberGroup group = orgRepositoryService.getMemberGroupById(memberGroupId.toString());

List<OrgTreeVo> members = JSONArray.parseArray(group.getMembers(), OrgTreeVo.class);

for (OrgTreeVo user: members) {

memberIds.add(user.getId());

}

});

} // 如果是成员分组的成员，也可以看到所有的流程实例

boolean isMemberOfGroup = memberIds.contains(currentUserId);

// log.info("isMemberOfGroup: {}", isMemberOfGroup);

boolean isAdmin = ( adminIds.size() > 0 && adminIds.indexOf(currentUserId) >= 0 ) ? true : false;

// log.info("isAdmin: {}", isAdmin); // query.processDefinitionKey(code).notDeleted();

query.processDefinitionKey(code); // if (currentTenantId.equals("fbd9654e-4d19-4d28-b525-e7477a8b28b9")) { // log.info("It's fbd9654e-4d19-4d28-b525-e7477a8b28b9."); // query.or().processInstanceTenantId(currentTenantId).processInstanceTenantId("default").processInstanceWithoutTenantId().endOr(); // } else {

// query.processInstanceTenantId(currentTenantId);

// } if (isAdmin || isMemberOfGroup || permission == 0) { } else { List<WflowCcTasks> wflowCcTasks = ccTasksMapper.selectList(

new QueryWrapper<WflowCcTasks>()

.eq("user\_id", currentUserId)

.eq("code", code)); Set<String> processInstanceIds = wflowCcTasks.stream().map(WflowCcTasks::getInstanceId)

.collect(Collectors.toSet()); // log.info("processInstanceIds: {}", processInstanceIds.toString()); query.or().involvedUser(currentUserId); if (processInstanceIds.size() > 0) {

query.processInstanceIds(processInstanceIds);

} query.endOr();

} if (keywords != null && !keywords.equals("")) { // query.variableExists("field9425885449977");

// query.variableValueLikeIgnoreCase("field9425885449977", "%" + keywords + "%"); List<String> fields = formService.getFormFields(code);

query.or();

for (String field: fields) {

query.variableValueLikeIgnoreCase(field, "%" + keywords + "%");

}

query.endOr();

}

} // log.info("query: {}", query.toString()); // long beginTime = System.currentTimeMillis();

List<HistoricProcessInstance> historicProcessInstances = query.orderByProcessInstanceStartTime().desc()

.orderByProcessInstanceEndTime().desc()

.listPage(pageSize \* (pageNo - 1), pageSize);

// long endTime = System.currentTimeMillis();

// log.info("Elapsed Time: {}", endTime - beginTime); // long beginTime1 = System.currentTimeMillis(); Map<String, String> instanceNodeMap = new HashMap<>();

List<ProcessInstanceVo> processInstanceVos = this.getInstances(instanceNodeMap, historicProcessInstances, currentUserId); // long endTime1 = System.currentTimeMillis();

// log.info("Elapsed Time1: {}", endTime1 - beginTime1); Page<ProcessInstanceVo> page = new Page<>();

page.setTotal(query.count());

page.setCurrent(pageNo);

page.setSize(pageSize);

page.setRecords(processInstanceVos);

return page;

} @Override

public Page<ProcessInstanceVo> getInvolvedMeListWithAllApps(Integer pageSize, Integer pageNo) { String currentUserId = UserUtil.getLoginUserId(); // log.info("currentUserId: {}", currentUserId); String currentTenantId = UserUtil.getLoginTenantId(); HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery(); if (currentTenantId.equals("fbd9654e-4d19-4d28-b525-e7477a8b28b9")) { log.info("It's fbd9654e-4d19-4d28-b525-e7477a8b28b9."); query.or().processInstanceTenantId(currentTenantId).processInstanceTenantId("default").processInstanceWithoutTenantId().endOr(); } else {

query.processInstanceTenantId(currentTenantId);

} List<WflowCcTasks> wflowCcTasks = ccTasksMapper.selectList(

new QueryWrapper<WflowCcTasks>()

.eq("user\_id", currentUserId)); Set<String> processInstanceIds = wflowCcTasks.stream().map(WflowCcTasks::getInstanceId)

.collect(Collectors.toSet()); // log.info("processInstanceIds: {}", processInstanceIds.toString()); query.or().involvedUser(currentUserId); if (processInstanceIds.size() > 0) {

query.processInstanceIds(processInstanceIds);

} query.endOr(); List<HistoricProcessInstance> historicProcessInstances = query.orderByProcessInstanceStartTime().desc()

.orderByProcessInstanceEndTime().desc()

.listPage(pageSize \* (pageNo - 1), pageSize); // log.info("historicProcessInstances.size(): {}", historicProcessInstances.size()); Page<ProcessInstanceVo> page = new Page<>();

page.setTotal(query.count());

page.setCurrent(pageNo);

page.setSize(pageSize);

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

page.setRecords(this.getInstances(instanceNodeMap, historicProcessInstances, currentUserId));

return page;

} @Override

public Page<ProjectVo> getProjectList(Integer pageSize, Integer pageNo, String code, String keywords, String completeDate, String status) { String projectListKey = "getProjectListK";

// 使用 Hash 结构，将不同参数存储在不同字段中

String hashField = String.format("keywords:%s\_completeDate:%s\_status:%s", keywords, completeDate, status); String results = jedis.hget(projectListKey, hashField); Page<ProjectVo> page = null; if (results != null) { page = JSONObject.parseObject(results, new TypeReference<Page<ProjectVo>>() {});

return page;

} // return JSON.parseObject(results, Page.class);

int firstResult = pageSize \* (pageNo - 1); HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery();

// log.info("code: {}", code); // String currentUserId = UserUtil.getLoginUserId(); // log.info("currentUserId: {}", currentUserId); String currentTenantId = UserUtil.getLoginTenantId(); // log.info("currentTenantId: {}", currentTenantId); query.processDefinitionKey(code); query.processInstanceTenantId(currentTenantId); if (keywords != null && !keywords.equals("")) { List<String> fields = formService.getFormFields(code);

query.or();

for (String field : fields) {

query.variableValueLikeIgnoreCase(field, "%" + keywords + "%");

}

query.endOr();

} if (completeDate != null && !completeDate.equals("")) {

query.or();

query.variableValueLikeIgnoreCase("field1337127388578", "%" + completeDate + "%");

query.endOr();

} // request variables

query.includeProcessVariables(); List<HistoricProcessInstance> historicProcessInstances = query.orderByProcessInstanceStartTime().desc()

.orderByProcessInstanceEndTime().desc()

.list(); // Sorting the list in memory based on the form variable

historicProcessInstances.sort(Comparator.comparing(h -> {

// Access and return the value of the date form variable for each

// HistoricProcessInstance

Object variableValue = h.getProcessVariables() != null

? h.getProcessVariables().get("field1337127388578")

: null; if (variableValue instanceof Date) {

return (Date) variableValue;

} else {

// Handle the case when the variable value is not a Date (e.g., handle null or

// other types)

return null; // or return a default date, depending on your requirements

}

}, Comparator.nullsLast(Comparator.naturalOrder()))); // log.info("historicProcessInstances.size(): {}",

// historicProcessInstances.size()); List<HistoricProcessInstance> newInstances = new ArrayList<>(); if (status != null && !status.equals("")) { for (HistoricProcessInstance ins : historicProcessInstances) { Map<String, Object> variables = ins.getProcessVariables(); // 解析图号数据定义项目状态

boolean isLeastOneDelay = false;

boolean isAllDelay = true;

boolean isAllCompleted = true;

ProjectDetailVo projectDetail = this.getProjectDetail(ins.getId(), ins); for (TuhaoVo tuhao : projectDetail.getTuhaoList()) { if (tuhao.getStatus().startsWith("延期")) {

isLeastOneDelay = true;

}

if (!tuhao.getStatus().startsWith("延期")) {

isAllDelay = false;

}

if (!tuhao.getStatus().equals("已完成")) {

isAllCompleted = false;

}

} String state = "";

Object file = variables.get("field5529924521341");

if (file == null) {

state = "待创建产前计划";

} else if (isAllDelay) {

state = "已延期";

} else if (isAllCompleted) {

state = "已完成";

} else if (isLeastOneDelay) {

state = "延期风险";

} else {

state = "正常进行中";

} if (status.equals(state)) {

newInstances.add(ins);

}

}

} else {

newInstances = historicProcessInstances;

} // log.info("newInstances.size(): {}", newInstances.size()); // Applying pagination manually

int toIndex = Math.min(firstResult + pageSize, newInstances.size());

List<HistoricProcessInstance> paginatedList = newInstances.subList(firstResult, toIndex); // log.info("paginatedList: {}", paginatedList.size()); LocalDate currentDate = LocalDate.now(); List<ProjectVo> projectList = new ArrayList<>(); for (HistoricProcessInstance ins : paginatedList) { ProjectVo projectVo = new ProjectVo();

projectVo.setInstanceId(ins.getId()); Map<String, Object> variables = ins.getProcessVariables(); // 项目名称

projectVo.setName(

variables.get("field6232327254883") != null ? variables.get("field6232327254883").toString()

: null); // 项目令号

projectVo.setCode(

variables.get("field9645527299577") != null ? variables.get("field9645527299577").toString()

: null); // 计划完成时间

projectVo.setPlanCompleteDate(

variables.get("field1337127388578") != null ? variables.get("field1337127388578").toString()

: null); // 计划开始时间

projectVo.setPlanStartDate(

variables.get("field8532624451956") != null ? variables.get("field8532624451956").toString()

: null); // 管理员

projectVo.setManager(

variables.get("field4105224493800") != null ? variables.get("field4105224493800").toString()

: null); long outdaysAll = 0; // 产前准备计划

Object file = variables.get("field5529924521341");

if (file == null) {

projectVo.setStatus("待创建产前计划");

// calc days

if (variables.get("field8532624451956") != null) {

LocalDate pDate = LocalDate

.parse(variables.get("field8532624451956").toString().substring(0, 10));

long days = ChronoUnit.DAYS.between(pDate, currentDate);

projectVo.setDays(days);

}

outdaysAll = -1;

projectVo.setDaysAll(outdaysAll);

projectList.add(projectVo);

continue;

} // 解析图号数据定义项目状态

boolean isLeastOneDelay = false;

boolean isAllDelay = true;

boolean isAllCompleted = true;

ProjectDetailVo projectDetail = this.getProjectDetail(ins.getId(), ins); boolean zhoubaoStatus = this.getZhouBaoStatusLastWeek(

variables.get("field9645527299577") != null ? variables.get("field9645527299577").toString()

: null);

// log.info("zhoubaoStatus: {}", zhoubaoStatus);

projectVo.setZhoubaoStatus(zhoubaoStatus); projectVo.setTuhaoVos(projectDetail.getTuhaoList()); for (TuhaoVo tuhao : projectDetail.getTuhaoList()) { if (tuhao.getStatus().startsWith("延期")) {

isLeastOneDelay = true;

}

if (!tuhao.getStatus().startsWith("延期")) {

isAllDelay = false;

}

if (!tuhao.getStatus().equals("已完成")) {

isAllCompleted = false;

} outdaysAll += tuhao.getDaysAll();

// log.info("tuhao.getDaysAll(): {}", outdaysAll);

} projectVo.setDaysAll(outdaysAll); if (projectDetail.getTuhaoList().size() > 0 && isAllDelay) { projectVo.setStatus("已延期"); } else if (projectDetail.getTuhaoList().size() > 0 && isLeastOneDelay) { projectVo.setStatus("延期风险"); } else if (projectDetail.getTuhaoList().size() > 0 && isAllCompleted) { projectVo.setStatus("已完成"); } else {

projectVo.setStatus("正常进行中");

} if (projectVo.getStatus().equals("已完成")) {

LocalDate pDate = LocalDate.parse(variables.get("field8532624451956").toString().substring(0, 10));

LocalDate cDate = LocalDate.parse(variables.get("field1337127388578").toString().substring(0, 10));

long days = ChronoUnit.DAYS.between(pDate, cDate);

projectVo.setDays(days);

} else {

LocalDate pDate = LocalDate.parse(variables.get("field8532624451956").toString().substring(0, 10));

long days = ChronoUnit.DAYS.between(pDate, currentDate);

projectVo.setDays(days);

} projectList.add(projectVo);

} page = new Page<ProjectVo>();

page.setTotal(projectList.size());

page.setCurrent(pageNo);

page.setSize(pageSize);

page.setRecords(projectList); String json = JSON.toJSONString(page);

jedis.hset(projectListKey, hashField, json);

jedis.expire(projectListKey, 60 \* 60); // 设置过期时间，单位为秒

// jedis.hget(projectListKey, json);

// 使用事务设置 Hash 字段，并设置过期时间

// Transaction transaction = jedis.multi();

// transaction.hset(projectListKey, hashField, json);

// transaction.expire(projectListKey, 60 \* 60); // 设置过期时间，单位为秒

// transaction.exec();

return page;

}

// @Async

@Override

public ProjectDetailVo getProjectDetail(String id, HistoricProcessInstance historicProcessInstance) { String currentTenantId = UserUtil.getLoginTenantId(); // log.info("currentTenantId: {}", currentTenantId); String formId = "";

if (UserUtil.getEnv().equals("prod")) {

formId = "wf65bf20b0e4b0de990e1f0c2e";

} else {

formId = "wf65dee870e4b0e59ac55f67b3";

} HistoricProcessInstance instance = null; if (historicProcessInstance != null) { instance = historicProcessInstance;

} else {

HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery();

query.processDefinitionKey(formId); query.processInstanceId(id); query.processInstanceTenantId(currentTenantId); query.includeProcessVariables(); instance = query.singleResult();

} if (instance == null) {

log.info("未找到该流程实例, currentTenantId: {}, instanceId: {} not exist.", currentTenantId, id);

throw new BusinessException("未找到该流程实例, currentTenantId: " + currentTenantId + ", id: " + id);

} Map<String, Object> variables = instance.getProcessVariables(); // 项目名称

// String projectName = "";

// if (variables != null && variables.get("field6232327254883") != null) {

// projectName = variables.get("field6232327254883").toString();

// }

// log.info("projectName: {}", projectName); // 项目令号

String projectCode = "";

if (variables.get("field9645527299577") != null) {

projectCode = variables.get("field9645527299577").toString();

}

// log.info("projectCode: {}", projectCode); // 查询图号数据

String tuhaoFormId = "";

if (UserUtil.getEnv().equals("prod")) {

tuhaoFormId = "wf65bf1ff7e4b0de990e1f0c2d";

} else {

tuhaoFormId = "wf65e18ba6e4b0e59ac55f67b9";

} HistoricProcessInstanceQuery queryTuhao = historyService.createHistoricProcessInstanceQuery(); queryTuhao.processInstanceTenantId(currentTenantId); queryTuhao.processDefinitionKey(tuhaoFormId); // 令号

queryTuhao.variableValueLikeIgnoreCase("field5517038455013", "%" + projectCode + "%"); queryTuhao.includeProcessVariables(); List<HistoricProcessInstance> instances = queryTuhao.list();

List<TuhaoVo> tuhaoVos = new ArrayList<>(); // 各阶段分布

Map<String, Integer> stages = new HashMap<>(); for (HistoricProcessInstance ins: instances) { Map<String, Object> tuhaoVariables = ins.getProcessVariables(); if (tuhaoVariables != null && tuhaoVariables.get("field8324238372074") != null && tuhaoVariables.get("field5517038455013") != null) {

try {

TuhaoVo tuhaoVo = new TuhaoVo();

tuhaoVo.setInstanceId(ins.getId()); // 项目名称

// String tuhaoProjectNameJson = tuhaoVariables.get("field8324238372074").toString();

// String tuhaoProjectName = JSONObject.parseObject(tuhaoProjectNameJson).get("mainValue").toString();

// 项目令号

Object tuhaoProjectCodeJson = tuhaoVariables.get("field5517038455013");

// log.info("tuhaoProjectCodeJson: {}", tuhaoProjectCodeJson);

if(tuhaoProjectCodeJson==null || tuhaoProjectCodeJson.equals("")){

continue;

} String tuhaoProjectCode = JSONObject.parseObject(tuhaoProjectCodeJson.toString()).get("mainValue").toString();

if (projectCode.equals(tuhaoProjectCode)) {

Object tuhaoName = tuhaoVariables.get("field4831222975217");

tuhaoVo.setTuhaoName(tuhaoName != null ? tuhaoName.toString() : null); String stage = this.getProcessInstanceCurrentNodeIdAndName(ins.getId());

String stageId = "";

String stageName = "";

if (stage != null && stage.indexOf("：") > 0) {

stageId = stage.split("：")[0];

stageName = stage.split("：")[1]; tuhaoVo.setCurrentNodeId(stageId);

tuhaoVo.setCurrentNode(stageName); if (stages.get(stageName) == null) {

stages.put(stageName, 1);

} else {

int count = stages.get(stageName);

stages.put(stageName, count + 1);

}

}

// log.info("stage: {}", stage); Map<String, String> status = this.parseTuhaoStatus(tuhaoVariables, stageName);

tuhaoVo.setStatus(status.get("status"));

tuhaoVo.setPlanCompleteDate(status.get("planDate"));

tuhaoVo.setDaysAll(status.get("daysAll") != null ? Long.parseLong(status.get("daysAll")) : 0);

Task currentTask = taskService.createTaskQuery()

.processInstanceId(ins.getId())

.singleResult(); String assignee = currentTask.getAssignee(); UserDo user = orgRepositoryService.getUserById(currentTenantId, assignee);

tuhaoVo.setCurrentNodeUser(JSONObject.toJSONString(user)); tuhaoVos.add(tuhaoVo);

}

} catch (Exception e) {

log.info("JSONObject.parseObject failed, instanceId: {}, field5517038455013: {}", ins.getId(), tuhaoVariables.get("field5517038455013"));

// e.printStackTrace();

}

}

// 记录公司外

String[] name\_FB={"field9338122966126","field4831222975217","field2151823729730"}; //校验是否上传错误

Boolean uploading\_tp=false;

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

List<Object> data\_s=datas.get(i);

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

//log.info("查看当前数据:{}",data\_s.get(0).toString().indexOf("图号"));

if(data\_s.get(0).toString().indexOf("组件号")<0 && data\_s.get(0).toString().indexOf("分包生产")<0){

if(data\_s.get(0).toString().indexOf("单位")<0){

switch (type){

case "厂内":

JSONObject jsonObject=new JSONObject();

jsonObject.put("key",type);

jsonObject.put("value","0");

map.put("field8811323120432",jsonObject);

for (int j=0;j<data\_s.size();j++){

String values=data\_s.get(j).toString();

if(j>1){

map.put(name\_GS[j],paseDate(values));

}else{

map.put(name\_GS[j],values);

}

}

list\_v.add(map);

break;

case "分包":

JSONObject jsonObject1=new JSONObject();

jsonObject1.put("key",type);

jsonObject1.put("value","1");

map.put("field8811323120432",jsonObject1);

for (int j=0;j<data\_s.size();j++){

if(j<3){

String values=data\_s.get(j).toString();

if(j>1){

map.put(name\_FB[j],paseDate(values));

}else{

map.put(name\_FB[j],values);

}

}else {

list\_v.add(map);

} }

break;

}

}

}

if(data\_s.get(0).toString().indexOf(instanceId)>-1 && uploading\_tp == false){

uploading\_tp=true;

} for (int dj=0;dj<data\_s.size();dj++){

if(data\_s.get(dj).toString().indexOf("公司内生产部分")>-1){

type="厂内";

}else if(data\_s.get(dj).toString().indexOf("分包生产部分")>-1) {

type="分包";

}

}

}

if(uploading\_tp){

//循环更新流程

for(Map<String,Object> map:list\_v){

//查询需要变更的流程信息

String sql = "SELECT id, formId, data -> '$.field5517038455013' as SCLH , data -> '$.field9338122966126' as TH";

String filtration="data -> '$.field5517038455013' like '%"+instanceId+"%' and data -> '$.field9338122966126' like '%"+map.get("field9338122966126").toString()+"%'";

List<JSONObject> sclh\_list=formDataMapper.getFormFieldValueser(sql, formId\_TH,filtration);

if(sclh\_list.size()>0){

// log.info("查看更新递交数据Map:{}",map);

if(sclh\_list.get(0).getString("id")!=null&&sclh\_list.get(0).getString("id")!=""){

//更新

formServiceimp.updateInstanceFormData(sclh\_list.get(0).getString("id"),map);

}

}else{

if(map.get("field9338122966126").toString().indexOf("分包")<0){

JSONObject jsonObject=new JSONObject();

//查询令号及项目信息

String sqllh = "SELECT id, formId, data -> '$.field1824823231244' as SCLH , data -> '$.field5888423244436' as XMZJ";

String filtrationlh ="data -> '$.field1824823231244' like '%"+instanceId+"%' and data ->'$.field8183523195543' like '%四川川锅锅炉有限责任公司%'";

List<JSONObject> sclh\_list1=formDataMapper.getFormFieldValueser(sqllh, formId\_LH,filtrationlh);

JSONObject sclh\_JSON=sclh\_list1.get(0);

String sqlxm = "SELECT id, formId, data -> '$.field3693554254644' as XM , data -> '$.field8701554233497' as XMZJ";

String filtration1="data -> '$.field8701554233497' like '%"+sclh\_JSON.getString("XMZJ")+"%'";

List<JSONObject> xm\_list=formDataMapper.getFormFieldValueser(sqlxm, formId\_XM,filtration1);

JSONObject xm\_JSON=xm\_list.get(0);

for(int ic=0;ic<datas.size();ic++){

List<Object> data\_sj=datas.get(ic);

if(data\_sj.get(0).toString().equals(instanceId)){

for(int jc=1;jc<data\_sj.size();jc++){

//产品名称及型号

jsonObject.put("field7712822943014",data\_sj.get(jc).toString().replaceAll("\"", ""));

break;

}

// log.info("产品名称字段：{}",data\_sj);

}

if(data\_sj.get(0).toString().indexOf("客户")>-1){

for(int jc=1;jc<data\_sj.size();jc++){

if(data\_sj.get(jc).toString()!=null||!data\_sj.get(jc).toString().equals("")){

//客户

jsonObject.put("field4409622957251",data\_sj.get(jc).toString().replaceAll("\"", ""));

break;

}

}

}

if(data\_sj.get(0).toString().indexOf("交货期")>-1){

for(int jc=1;jc<data\_sj.size();jc++){

if(data\_sj.get(jc).toString()!=null||!data\_sj.get(jc).toString().equals("")){

//交货期

jsonObject.put("field6293223011604",paseDate(data\_sj.get(jc).toString().replaceAll("\"", "")));

break;

}

}

}

}

//令号

JSONObject jsonObject1=new JSONObject();

String[] str={""};

// log.info("下达计划数量processParams:{}",str);

jsonObject1.put("extraValue",str);

// log.info("xm\_JSON:{}",sclh\_JSON);

jsonObject1.put("mainValue",sclh\_JSON.getString("SCLH").replaceAll("\"", ""));

jsonObject1.put("mainId",sclh\_JSON.getString("id"));

jsonObject.put("field5517038455013",jsonObject1);

//项目

JSONObject jsonObject2=new JSONObject();

jsonObject2.put("extraValue",str);

// log.info("xm\_JSON:{}",xm\_JSON);

jsonObject2.put("mainValue",xm\_JSON.getString("XM").replaceAll("\"", ""));

jsonObject2.put("mainId",xm\_JSON.getString("id"));

jsonObject.put("field8324238372074",jsonObject2); jsonObject.put("field9338122966126",map.get("field9338122966126").toString());

jsonObject.put("field4831222975217",map.get("field4831222975217").toString()); if(map.get("field8811323120432").equals("厂内")||map.get("field8811323120432").toString().indexOf("厂内")>-1){

jsonObject.put("field5986223468574",map.get("field5986223468574").toString());

jsonObject.put("field2025023600135",map.get("field2025023600135").toString());

jsonObject.put("field4375323671268",map.get("field4375323671268").toString());

jsonObject.put("field8811323120432",map.get("field8811323120432").toString());

}else{

jsonObject.put("field2151823729730",map.get("field2151823729730").toString());

jsonObject.put("field8811323120432",map.get("field8811323120432").toString());

} ProcessStartParamsVo processParams = new ProcessStartParamsVo();

processParams.setDeptId("129");

processParams.setFormData(jsonObject);

processParams.setProcessUsers(new HashMap<>()); if(userId==null||userId.equals("")){

userId="301708";

} WflowModels model = processModelService.getWflowModel(formId\_TH);

String processdefId = model.getProcessDefId();

// log.info("jsonObject:{}",jsonObject);

//创建

this.startProcess(processdefId, processParams, userId, "");

}

}

}

}else{

return "不一致";

}

//刷新缓存

this.clearRedisCache(); return "更新完成";

} /\*\*

\* 获取需要更新或新增的参数

\* @param list\_Objects

\* @param fromId

\* @param userId

\* @return

\*/

public String setupdDistribute(List<JSONObject> list\_Objects, String instanceId, String userId){

String formId\_JH=null;

String formId\_TH=null;

String formId\_XM=null;

String formId\_LH=null;

if (UserUtil.getEnv().equals("prod")) {

formId\_JH = "wf65bf20b0e4b0de990e1f0c2e";

formId\_TH = "wf65bf1ff7e4b0de990e1f0c2d";

formId\_XM = "wf656eba8de4b0fec83c013808";

formId\_LH = "wf657f05b3e4b0de99f108a8b2";

} else {

formId\_JH = "wf65dee870e4b0e59ac55f67b3";

formId\_TH = "wf65e18ba6e4b0e59ac55f67b9";

formId\_XM = "wf65e18572e4b0e59ac55f67b7";

formId\_LH = "wf65e1869ae4b0e59ac55f67b8";

}

//循环更新数据

for(JSONObject json\_rt:list\_Objects){

if(json\_rt.getString("id")!=null||!json\_rt.getString("id").equals("")){

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

for(String key:json\_rt.keySet()){

map.put(key, json\_rt.getString(key));

}

//更新

formServiceimp.updateInstanceFormData(json\_rt.getString("id"),map);

}else{

//查询令号及项目信息

String sqllh = "SELECT id, formId, data -> '$.field1824823231244' as SCLH , data -> '$.field5888423244436' as XMZJ";

String filtrationlh ="data -> '$.field1824823231244' like '%"+instanceId+"%'";

List<JSONObject> sclh\_list1=formDataMapper.getFormFieldValueser(sqllh, formId\_LH,filtrationlh);

JSONObject sclh\_JSON=sclh\_list1.get(0);

String sqlxm = "SELECT id, formId, data -> '$.field3693554254644' as XM , data -> '$.field8701554233497' as XMZJ";

String filtration1="data -> '$.field8701554233497' like '%"+sclh\_JSON.getString("XMZJ")+"%'";

List<JSONObject> xm\_list=formDataMapper.getFormFieldValueser(sqlxm, formId\_XM,filtration1);

JSONObject xm\_JSON=xm\_list.get(0);

//遍历查询未配到的图号码

String sql\_succeed= "SELECT id, formId, data -> '$.field5517038455013' as SCLH , data -> '$.field9338122966126' as TH,data -> '$.field4831222975217' as MC, data -> '$.field7712822943014' as CPMC, data -> '$.field4409622957251' as KH, data -> '$.field6293223011604' as JHQ";

String filtration\_succeed="data -> '$.field5517038455013' like '%"+instanceId+"%'";

List<JSONObject> sclh\_list\_succeed=formDataMapper.getFormFieldValueser(sql\_succeed, formId\_TH,filtration\_succeed);

//产品名称及型号

json\_rt.put("field7712822943014",sclh\_list\_succeed.get(0).getString("CPMC").replaceAll("\"", ""));

//客户

json\_rt.put("field4409622957251",sclh\_list\_succeed.get(0).getString("KH").replaceAll("\"", ""));

//交货期

json\_rt.put("field6293223011604",sclh\_list\_succeed.get(0).getString("JHQ").replaceAll("\"", ""));

//令号

JSONObject jsonObject1=new JSONObject();

String[] str={""};

jsonObject1.put("extraValue",str);

jsonObject1.put("mainValue",sclh\_JSON.getString("SCLH").replaceAll("\"", ""));

jsonObject1.put("mainId",sclh\_JSON.getString("id"));

json\_rt.put("field5517038455013",jsonObject1);

//项目

JSONObject jsonObject2=new JSONObject();

jsonObject2.put("extraValue",str);

// log.info("xm\_JSON:{}",xm\_JSON);

jsonObject2.put("mainValue",xm\_JSON.getString("XM").replaceAll("\"", ""));

jsonObject2.put("mainId",xm\_JSON.getString("id"));

json\_rt.put("field8324238372074",jsonObject2); ProcessStartParamsVo processParams = new ProcessStartParamsVo();

processParams.setDeptId("129");

processParams.setFormData(json\_rt);

processParams.setProcessUsers(new HashMap<>()); if(userId==null||userId.equals("")){

userId="301708";

} WflowModels model = processModelService.getWflowModel(formId\_TH);

String processdefId = model.getProcessDefId();

//创建

this.startProcess(processdefId, processParams, userId, "");

}

}

return "保存完成";

} /\*\*

\* 更新排序

\* @param file

\* @param instanceId

\* @param userId

\* @return

\*/

public JSONObject getupdDistribute(MultipartFile file, String instanceId, String userId){

String formId\_JH=null;

String formId\_TH=null;

String formId\_XM=null;

String formId\_LH=null;

if (UserUtil.getEnv().equals("prod")) {

formId\_JH = "wf65bf20b0e4b0de990e1f0c2e";

formId\_TH = "wf65bf1ff7e4b0de990e1f0c2d";

formId\_XM = "wf656eba8de4b0fec83c013808";

formId\_LH = "wf657f05b3e4b0de99f108a8b2";

} else {

formId\_JH = "wf65dee870e4b0e59ac55f67b3";

formId\_TH = "wf65e18ba6e4b0e59ac55f67b9";

formId\_XM = "wf65e18572e4b0e59ac55f67b7";

formId\_LH = "wf65e1869ae4b0e59ac55f67b8";

}

//获取用户信息

StpUtil.login(userId);

String companyId = UserUtil.getLoginTenantId();

//将文件变成数组

List<List<Object>> datas = formServiceimp.parseExcel(file);

log.info("查看需要更新条数:{}",datas);

//创建计划存储变量

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

// 记录是否需要取值

String type="无";

// 记录公司内

String[] name\_GS={"field9338122966126","field4831222975217","field5986223468574","field2025023600135","field4375323671268"};

// 记录公司外

String[] name\_FB={"field9338122966126","field4831222975217","field2151823729730"};

//json字段

String[] js\_key={"id","field9338122966126","field4831222975217","field5986223468574","field2025023600135","field4375323671268","field2151823729730","field8811323120432"};

//校验是否上传错误

Boolean uploading\_tp=false;

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

List<Object> data\_s=datas.get(i);

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

//log.info("查看当前数据:{}",data\_s.get(0).toString().indexOf("图号"));

if(data\_s.get(0).toString().indexOf("图号")<0 && data\_s.get(0).toString().indexOf("公司内生产部分")<0){

if(data\_s.get(0).toString().indexOf("单位")<0){

switch (type){

case "厂内":

JSONObject jsonObject=new JSONObject();

jsonObject.put("key",type);

jsonObject.put("value","0");

map.put("field8811323120432",jsonObject);

for (int j=0;j<data\_s.size();j++){

String values=data\_s.get(j).toString();

if(j>1){

map.put(name\_GS[j],paseDate(values));

}else{

map.put(name\_GS[j],values);

}

}

list\_v.add(map);

break;

case "分包":

JSONObject jsonObject1=new JSONObject();

jsonObject1.put("key",type);

jsonObject1.put("value","1");

map.put("field8811323120432",jsonObject1);

for (int j=0;j<data\_s.size();j++){

if(j<3){

String values=data\_s.get(j).toString();

if(j>1){

map.put(name\_FB[j],paseDate(values));

}else{

map.put(name\_FB[j],values);

}

}else {

list\_v.add(map);

} }

break;

}

}

}

if(data\_s.get(0).toString().indexOf(instanceId)>-1 && uploading\_tp == false){

uploading\_tp=true;

} for (int dj=0;dj<data\_s.size();dj++){

if(data\_s.get(dj).toString().indexOf("公司内生产部分")>-1){

type="厂内";

}else if(data\_s.get(dj).toString().indexOf("分包生产部分")>-1) {

type="分包";

}

}

}

//创建返回参数

JSONObject json\_rt=new JSONObject();

json\_rt.put("verify",uploading\_tp);

json\_rt.put("key",js\_key);

List<Map<String,Object>> list\_rt\_succeed=new ArrayList<>();

List<Map<String,Object>> list\_rt\_unidentification=new ArrayList<>();

List<JSONObject> list\_rt\_unmatch=new ArrayList<>();

if(uploading\_tp){

//循环对数据进行分类

for(Map<String,Object> map:list\_v){

//查询数据库中是否存在

String sql = "SELECT id, formId, data -> '$.field5517038455013' as SCLH , data -> '$.field9338122966126' as TH";

String filtration="data -> '$.field5517038455013' like '%"+instanceId+"%' and data -> '$.field9338122966126' like '%"+map.get("field9338122966126").toString()+"%'";

List<JSONObject> sclh\_list=formDataMapper.getFormFieldValueser(sql, formId\_TH,filtration);

if(sclh\_list.size()>0){

// log.info("查看更新递交数据Map:{}",map);

if(sclh\_list.get(0).getString("id")!=null&&sclh\_list.get(0).getString("id")!=""){

map.put("id", sclh\_list.get(0).getString("id"));

list\_rt\_succeed.add(map);

}

}else{

if(sclh\_list.get(0).getString("id")!=null&&sclh\_list.get(0).getString("id")!=""){

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

list\_rt\_unidentification.add(map);

}

}

} json\_rt.put("succeed",list\_rt\_succeed);

json\_rt.put("unidentification",list\_rt\_unidentification);

//遍历查询未配到的图号码

String sql\_succeed= "SELECT id, formId, data -> '$.field5517038455013' as SCLH , data -> '$.field9338122966126' as TH,data -> '$.field4831222975217' as MC";

String filtration\_succeed="data -> '$.field5517038455013' like '%"+instanceId+"%'";

List<JSONObject> sclh\_list\_succeed=formDataMapper.getFormFieldValueser(sql\_succeed, formId\_TH,filtration\_succeed);

if(sclh\_list\_succeed.size()>0){

for(JSONObject rt:sclh\_list\_succeed){

boolean found = false;

for(Map<String,Object> map\_Map:list\_rt\_succeed){

if(rt.getString("TH").equals(map\_Map.get("field9338122966126").toString())){

found = true;

break;

}

}

if(!found){

JSONObject js\_tys=new JSONObject();

js\_tys.put("id",rt.getString("id"));

js\_tys.put("field9338122966126",rt.getString("TH").replaceAll("\"",""));

js\_tys.put("field4831222975217",rt.getString("MC").replaceAll("\"",""));

list\_rt\_unmatch.add(js\_tys);

}

}

} json\_rt.put("unmatch",list\_rt\_unmatch);

}else{ json\_rt.put("succeed",list\_rt\_succeed);

json\_rt.put("unidentification",list\_rt\_unidentification);

json\_rt.put("unmatch",list\_rt\_unmatch); return json\_rt;

}

return json\_rt;

}

public void checkPlanTimeExpired() { // 项目图号进度

String formId = "wf65bf1ff7e4b0de990e1f0c2d";

if (UserUtil.getEnv().equals("prod")) {

formId = "wf65bf1ff7e4b0de990e1f0c2d";

} else {

formId = "wf65dee870e4b0e59ac55f67b3";

} Set<String> ids\_plan = new HashSet<String>();

String userId\_plan = "301301";

String key\_plan = "isSent\_plan|" + formId; Set<String> ids\_craftsmanship = new HashSet<String>();

String userId\_craftsmanship = "301979";

String key\_craftsmanship = "isSent\_craftsmanship|" + formId; Set<String> ids\_show = new HashSet<String>();

String userId\_show = "302836";

String key\_show = "isSent\_show|" + formId; Set<String> ids\_material = new HashSet<String>();

String userId\_material = "302836";

String key\_material = "isSent\_material|" + formId; Set<String> ids\_product = new HashSet<String>();

String userId\_product = "301961";

String key\_product = "isSent\_product|" + formId; Set<String> ids\_ship = new HashSet<String>();

String userId\_ship = "301961";

String key\_ship = "isSent\_ship|" + formId; Set<String> ids\_complete = new HashSet<String>();

String userId\_complete = "300189";

String key\_complete = "isSent\_complete|" + formId; HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery();

query.processDefinitionKey(formId);

query.unfinished(); List<HistoricProcessInstance> historicProcessInstances = query.orderByProcessInstanceStartTime().desc()

.orderByProcessInstanceEndTime().desc()

.list(); // log.info("historicProcessInstances.size(): {}", historicProcessInstances.size()); LocalDate currentDate = LocalDate.now();

for (HistoricProcessInstance instance: historicProcessInstances) { try { // 设计计划完成时间

Object variable = runtimeService.getVariable(instance.getId(), "field5509623216808");

String planDate = variable != null ? variable.toString() : null;

// log.info("instance.getId(): {}, planDate: {}", instance.getId(), planDate);

// 设计实际完成时间

Object variable\_actually = runtimeService.getVariable(instance.getId(), "field5270923255922");

if (planDate != null && planDate.length() == 10 && variable\_actually == null) {

// log.info("planDate: {}", planDate);

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate.replaceAll("^\"|\"$", ""));

// log.info("planDateTime: {}", planDateTime);

int comparisonResult = planDateTime.compareTo(currentDate);

// log.info("comparisonResult: {}", comparisonResult);

if (comparisonResult < 0) {

ids\_plan.add(instance.getId());

userId\_plan = instance.getStartUserId();

// log.info("设计阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate);

}

}

// 工艺计划完成时间

Object variable1 = runtimeService.getVariable(instance.getId(), "field8341323299032");

String planDate1 = variable1 != null ? variable1.toString() : null;

// 工艺实际完成时间

Object variable1\_actually = runtimeService.getVariable(instance.getId(), "field6070623327073");

if (planDate1 != null && planDate1.length() == 10 && variable1\_actually == null) {

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate1.replaceAll("^\"|\"$", ""));

int comparisonResult = planDateTime.compareTo(currentDate);

if (comparisonResult < 0) {

ids\_craftsmanship.add(instance.getId());

userId\_craftsmanship = instance.getStartUserId();

// log.info("工艺阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate1);

}

}

// 晒图计划完成时间

Object variable2 = runtimeService.getVariable(instance.getId(), "field7112723392728");

String planDate2 = variable2 != null ? variable2.toString() : null;

// 晒图实际完成时间

Object variable2\_actually = runtimeService.getVariable(instance.getId(), "field2012323418072");

if (planDate2 != null && planDate2.length() == 10 && variable2\_actually == null) {

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate2.replaceAll("^\"|\"$", ""));

int comparisonResult = planDateTime.compareTo(currentDate);

if (comparisonResult < 0) {

ids\_show.add(instance.getId());

userId\_show = instance.getStartUserId();

// log.info("晒图阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate2);

}

}

// 材料投入计划完成时间

Object variable3 = runtimeService.getVariable(instance.getId(), "field5986223468574");

String planDate3 = variable3 != null ? variable3.toString() : null;

// 材料投入实际完成时间

Object variable3\_actually = runtimeService.getVariable(instance.getId(), "field4786823530583");

if (planDate3 != null && planDate3.length() == 10 && variable3\_actually == null) {

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate3.replaceAll("^\"|\"$", ""));

int comparisonResult = planDateTime.compareTo(currentDate);

if (comparisonResult < 0) {

ids\_material.add(instance.getId());

userId\_material = instance.getStartUserId();

// log.info("材料投入阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate3);

}

}

// 生产计划完成时间

Object variable4 = runtimeService.getVariable(instance.getId(), "field2025023600135");

String planDate4 = variable4 != null ? variable4.toString() : null;

// 生产实际完成时间

Object variable4\_actually = runtimeService.getVariable(instance.getId(), "field6785523613422");

if (planDate4 != null && planDate4.length() == 10 && variable4\_actually == null) {

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate4.replaceAll("^\"|\"$", ""));

int comparisonResult = planDateTime.compareTo(currentDate);

if (comparisonResult < 0) {

ids\_product.add(instance.getId());

userId\_product = instance.getStartUserId();

// log.info("生产阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate4);

}

}

// 发运计划完成时间

Object variable5 = runtimeService.getVariable(instance.getId(), "field4375323671268");

String planDate5 = variable5 != null ? variable5.toString() : null;

// 发运实际完成时间

Object variable5\_actually = runtimeService.getVariable(instance.getId(), "field1127223685439");

if (planDate5 != null && planDate5.length() == 10 && variable5\_actually == null) {

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate5.replaceAll("^\"|\"$", ""));

int comparisonResult = planDateTime.compareTo(currentDate);

if (comparisonResult < 0) {

ids\_ship.add(instance.getId());

userId\_ship = instance.getStartUserId();

// log.info("发运阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate5);

}

}

// 交货计划完成时间

Object variable6 = runtimeService.getVariable(instance.getId(), "field2151823729730");

String planDate6 = variable6 != null ? variable6.toString() : null;

// 交货实际完成时间

Object variable6\_actually = runtimeService.getVariable(instance.getId(), "field2045223747498");

if (planDate6 != null && planDate6.length() == 10 && variable6\_actually == null) {

// 比较日期

LocalDate planDateTime = LocalDate.parse(planDate6.replaceAll("^\"|\"$", ""));

int comparisonResult = planDateTime.compareTo(currentDate);

if (comparisonResult < 0) {

ids\_complete.add(instance.getId());

userId\_complete = instance.getStartUserId();

// log.info("交货阶段逾期, instanceId: {}, plan date: {}", instance.getId(), planDate6);

}

}

} catch (Exception e) {

e.printStackTrace();

}

} String isSent\_plan = jedis.get(key\_plan);

// log.info("ids\_plan.size(): {}", ids\_plan.size());

if (ids\_plan.size() > 0 && isSent\_plan == null) {

String title = "您有项目在设计阶段逾期，请修改计划完成日期。";

String description = ids\_plan.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_plan, null, title, description, String.join(",", ids\_plan));

jedis.setex(key\_plan, 24 \* 60 \* 60, "true");

} String isSent\_craftsmanship = jedis.get(key\_craftsmanship);

if (ids\_craftsmanship.size() > 0 && isSent\_craftsmanship == null) {

String title = "您有项目在工艺阶段逾期，请修改计划完成日期。";

String description = ids\_craftsmanship.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_craftsmanship, null, title, description, String.join(",", ids\_craftsmanship));

jedis.setex(key\_craftsmanship, 24 \* 60 \* 60, "true");

} String isSent\_show = jedis.get(key\_show);

if (ids\_show.size() > 0 && isSent\_show == null) {

String title = "您有项目在晒图阶段逾期，请修改计划完成日期。";

String description = ids\_show.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_show, null, title, description, String.join(",", ids\_show));

jedis.setex(key\_show, 24 \* 60 \* 60, "true");

} String isSent\_material = jedis.get(key\_material);

if (ids\_material.size() > 0 && isSent\_material == null) {

String title = "您有项目在材料投入阶段逾期，请修改计划完成日期。";

String description = ids\_material.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_material, null, title, description, String.join(",", ids\_material));

jedis.setex(key\_material, 24 \* 60 \* 60, "true");

} String isSent\_product = jedis.get(key\_product);

if (ids\_product.size() > 0 && isSent\_product == null) {

String title = "您有项目在生产阶段逾期，请修改计划完成日期。";

String description = ids\_product.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_product, null, title, description, String.join(",", ids\_product));

jedis.setex(key\_product, 24 \* 60 \* 60, "true");

} String isSent\_ship = jedis.get(key\_ship);

if (ids\_ship.size() > 0 && isSent\_ship == null) {

String title = "您有项目在发运阶段逾期，请修改计划完成日期。";

String description = ids\_ship.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_ship, null, title, description, String.join(",", ids\_ship));

jedis.setex(key\_ship, 24 \* 60 \* 60, "true");

} String isSent\_complete = jedis.get(key\_complete);

if (ids\_complete.size() > 0 && isSent\_complete == null) {

String title = "您有项目在交货阶段逾期，请修改计划完成日期。";

String description = ids\_complete.size() + "条数据逾期。";

wechatWorkService.sendNccTextCardMsg(formId, userId\_complete, null, title, description, String.join(",", ids\_complete));

jedis.setex(key\_complete, 24 \* 60 \* 60, "true");

}

}

public void checkLinghaoStart(){

NccProjectLx projectInfo = menuService.testNccData();

log.info("projectInfo {}", projectInfo);

String sclh = projectInfo.getSCLH();

if(sclh == null || sclh.equals("")){

return;

}

String formId = UserUtil.getEnv().equals("prod") ? "wf65bf20b0e4b0de990e1f0c2e" : "wf65dee870e4b0e59ac55f67b3"; // log.info("formId>>>>>>>>>> {}", formId);

WflowModels model = processModelService.getWflowModel(formId);

String defId = model.getProcessDefId();

// 如果项目经理为空，users为空

String xmjl = projectInfo.getXMJL(); log.info("xmjl>>>>>>>>>> {}", xmjl);

// if(xmjl == null || xmjl.equals("")){

// return;

// }

String companyId = "fbd9654e-4d19-4d28-b525-e7477a8b28b9";

// log.info("companyId： {}", companyId);

// StpUtil.login("302940");

String key = "projectinfo\_key";

String value = jedis.get(key);

// log.info("value>>>>>>>>>> {}", value);

// log.info("xm>>>>>>>>>> {},linghao:{}", projectInfo.getXM(),sclh);

if (value == null || !value.equals(projectInfo.getXM() + "\_" + sclh)) {

JSONObject data = new JSONObject();

String promoter = "301708";

if(xmjl != null && !xmjl.equals("")){

// 获取项目经理

List<OrgTreeVo> users = usersMapper.selectUsersByPy(xmjl,companyId);

if (!users.isEmpty()) {

promoter = users.get(0).getId();

JSONArray xmjlArray = new JSONArray();

JSONObject xmjlObject = new JSONObject();

xmjlObject.put("id", users.get(0).getId());

xmjlObject.put("name", users.get(0).getName());

xmjlObject.put("type", users.get(0).getType());

xmjlObject.put("avatar", users.get(0).getAvatar());

xmjlArray.add(xmjlObject);

data.put("field4105224493800", xmjlArray);

}

}else{

JSONArray xmjlArray = new JSONArray();

JSONObject xmjlObject = new JSONObject();

xmjlObject.put("id", "301708");

xmjlObject.put("name", "陈勇");

xmjlObject.put("type", "user");

xmjlObject.put("avatar", "http://wework.qpic.cn/bizmail/62rW1WlamdcyyRIicMcialxuHT02CoRia2WuMEhvl4tVrJwOZIibpRBDYQ/0");

xmjlArray.add(xmjlObject);

data.put("field4105224493800", xmjlArray);

}

data.put("field6232327254883", projectInfo.getXM());

data.put("field8532624451956", projectInfo.getJHKSRQ());

data.put("field1337127388578", projectInfo.getJHWCRQ());

data.put("field9645527299577", projectInfo.getSCLH());

//总图号

data.put("field6000024515147", "");

// 执行后续操作

ProcessStartParamsVo processParams = new ProcessStartParamsVo();

processParams.setDeptId("8");

processParams.setFormData(data);

processParams.setProcessUsers(new HashMap<>());

//这里掉的？

String instanceId = this.startProcess(defId, processParams, promoter, companyId);

// 1. 更新redis的值

jedis.set(key, projectInfo.getXM() + "\_" + sclh);

// 2. 执行后续操作

// ...提醒上传产前计划

String description = projectInfo.getXM() + "项目已创建，请尽快上传产前计划和生产执行计划。项目名称：" + projectInfo.getXM() + "，生产零号：" + sclh + "。";

wechatWorkService.sendAppTextCardMsg("wf65bf20b0e4b0de990e1f0c2e", promoter,"", "新项目创建提醒",instanceId, description, "node\_245289408353");

}

} public void testLinghaoStart(String name, String linghao, String endTime){

String formId = "wf65bf20b0e4b0de990e1f0c2e";

if (UserUtil.getEnv().equals("prod")) {

formId = "wf65bf20b0e4b0de990e1f0c2e";

} else {

formId = "wf65dee870e4b0e59ac55f67b3";

} log.info("formId>>>>>>>>>> {}", formId);

WflowModels model = processModelService.getWflowModel(formId);

String defId = model.getProcessDefId();

String companyId = "fbd9654e-4d19-4d28-b525-e7477a8b28b9";

JSONObject data = new JSONObject();

JSONArray xmjlArray = new JSONArray();

JSONObject xmjlObject = new JSONObject();

xmjlObject.put("id", "302940");

xmjlObject.put("name", "覃益佳");

xmjlObject.put("type", "user");

xmjlObject.put("avatar", "https://wework.qpic.cn/wwpic/384946\_9FjXNCY-Q7G3eWT\_1673314411/0");

xmjlArray.add(xmjlObject);

data.put("field4105224493800", xmjlArray);

data.put("field6232327254883", name);

data.put("field8532624451956", "2024-01-24 00:00:00");

data.put("field1337127388578", endTime);

data.put("field9645527299577", linghao);

//总图号

data.put("field6000024515147", "");

// 执行后续操作

ProcessStartParamsVo processParams = new ProcessStartParamsVo();

processParams.setDeptId("8");

processParams.setFormData(data);

processParams.setProcessUsers(new HashMap<>());

//这里掉的？

String instanceId = this.startProcess(defId, processParams, "302940", companyId);

// 1. 更新redis的值

// jedis.set(key, xmzj + sclh);

// 2. 执行后续操作

// ...提醒上传产前计划

String description = name + " 项目已创建，请尽快上传产前计划和生产执行计划。" + "项目名称：" + name + "，生产令号：" + linghao + "1260018。";

wechatWorkService.sendAppTextCardMsg(formId, "302940","", "新项目创建提醒",instanceId, description, "node\_245289408353");

// return;

// String key = "projectinfo\_key";

// String value = jedis.get(key);

// if (value == null || !value.equals(xmzj + sclh)) {

// String promoter = "301708";

// if(xmjl != null && !xmjl.equals("")){

// // 获取项目经理

// List<OrgTreeVo> users = usersMapper.selectUsersByPy(xmjl,companyId);

// promoter = users.get(0).getId();

// JSONArray xmjlArray = new JSONArray();

// JSONObject xmjlObject = new JSONObject();

// xmjlObject.put("id", users.get(0).getId());

// xmjlObject.put("name", users.get(0).getName());

// xmjlObject.put("type", users.get(0).getType());

// xmjlObject.put("avatar", users.get(0).getAvatar());

// xmjlArray.add(xmjlObject);

// data.put("field4105224493800", xmjlArray);

// }else{

// JSONArray xmjlArray = new JSONArray();

// JSONObject xmjlObject = new JSONObject();

// xmjlObject.put("id", "301708");

// xmjlObject.put("name", "陈勇");

// xmjlObject.put("type", "user");

// xmjlObject.put("avatar", "http://wework.qpic.cn/bizmail/62rW1WlamdcyyRIicMcialxuHT02CoRia2WuMEhvl4tVrJwOZIibpRBDYQ/0");

// xmjlArray.add(xmjlObject);

// data.put("field4105224493800", xmjlArray);

// }

// data.put("field6232327254883", projectInfo.getXM());

// data.put("field8532624451956", projectInfo.getJHKSRQ());

// data.put("field1337127388578", projectInfo.getJHWCRQ());

// data.put("field9645527299577", projectInfo.getSCLH());

// //总图号

// data.put("field6000024515147", "");

// // 执行后续操作

// ProcessStartParamsVo processParams = new ProcessStartParamsVo();

// processParams.setDeptId("8");

// processParams.setFormData(data);

// processParams.setProcessUsers(new HashMap<>());

// //这里掉的？

// String instanceId = processInstanceService.startProcess(defId, processParams, "302940", companyId);

// // 1. 更新redis的值

// jedis.set(key, xmzj + sclh);

// // 2. 执行后续操作

// // ...提醒上传产前计划

// String description = projectInfo.getXM() + "项目已创建，请尽快上传产前计划和生产执行计划。项目名称：" + projectInfo.getXM() + "，生产零号：" + sclh + "。";

// wechatWorkService.sendAppTextCardMsg("wf65bf20b0e4b0de990e1f0c2e", "302940","", "新项目创建提醒",instanceId, description, "node\_245289408353");

// }

} public void checkProjectYsk() {

String formId = "wf65bf1ff7e4b0de990e1f0c2d";

if (UserUtil.getEnv().equals("prod")) {

formId = "wf65bf1ff7e4b0de990e1f0c2d";

} else {

formId = "wf65dee870e4b0e59ac55f67b3";

}

JSONArray yskArray = formService.isFieldExist(formId, "");

// log.info("yskArray {}", yskArray);

// JSONObject getProjetcList = formService.getProjetcTaskList();

// String ids = "";

// String description = "";

// 循环遍历 yskArray，获取每个项目的名称和令号

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

JSONObject yskObject = yskArray.getJSONObject(i);

// 如果yskObject不包含field8324238372074和field5517038455013，或者为空，就不再提醒

if (!yskObject.containsKey("field8324238372074") || !yskObject.containsKey("field5517038455013")) {

continue;

}

String xm = yskObject.getJSONObject("field8324238372074").getString("mainValue");

String sclh = yskObject.getJSONObject("field5517038455013").getString("mainValue");

if(sclh == null || sclh.equals("")){

continue;

}

// log.info("xm {}, sclh {}", xm, sclh);

String key = "ysk\_key\_" + sclh + "\_" + xm;

String value = jedis.get(key);

// 如果 value 不为空，就不再提醒

if (value != null) {

continue;

}

// 如果 value 为空，就提醒

int yskCount = formService.yskCount(sclh);

log.info("yskCount {}", yskCount);

if(yskCount == 0){

//根据项目名称和令号查询项目id

String projectId = formDataMapper.getProjectDataId(xm, sclh);

String description = "<div>项目名称：" + xm + "</div>&nbsp;<div>生产零号：" + sclh + "</div><br>";

description = description.substring(0, description.length() - 4);

log.info("description {}", description);

wechatWorkService.sendAppTextCardMsg("wf65bf20b0e4b0de990e1f0c2e", "302940","", "上传生产计划提醒",projectId, description, "node\_263937293659");

}

jedis.setex(key, 24 \* 60 \* 60, "true");

}

// log.info("ids {}", ids);

// log.info("description {}", description);

// if(ids.length() > 0){

// // 去掉ids末尾逗号

// ids = ids.substring(0, ids.length() - 1);

// // 去掉description末尾<br>

// description = description.substring(0, description.length() - 4);

// log.info("description {}", description);

// wechatWorkService.sendAppTextCardMsg("wf65bf20b0e4b0de990e1f0c2e", "302940","", "上传生产计划提醒",ids, description, "node\_263937293659");

// // wechatWorkService.sendNccTextCardMsg("wf65bf1ff7e4b0de990e1f0c2d", "302940","", "上传生产计划提醒",description, ids);

// }

} public boolean getZhouBaoStatusLastWeek(String lhCode) {

boolean status = false;

String apiDomain = cgManagerConfig.getApidomain();

String url = String.format(apiDomain + "/v1/form/weekreportlast?linghaoId=%s",

lhCode);

HttpRequest request = HttpRequest.get(url);

HttpResponse response = request.timeout(50000).executeAsync();

Map<String, Object> result = JSONObject.parse(response.body());

status = (boolean) result.get("d");

return status;

} @Override

public String clearRedisCache() {

jedis.del("getProjectListKey");

this.getProjectList(999, 1, "wf65bf20b0e4b0de990e1f0c2e", null, null, null);

return "success";

} public Map<String, List<Object>> getZhouBaoLastWeekList(String lhCode) {

String apiDomain = cgManagerConfig.getApidomain();

String url = String.format(apiDomain + "/v1/form/weekreportlastlist?linghaoId=%s",

lhCode);

HttpRequest request = HttpRequest.get(url);

HttpResponse response = request.timeout(50000).executeAsync();

Map<String, Object> result = JSONObject.parse(response.body());

return (Map<String, List<Object>>) result.get("d");

} public String AISummary(String question, String prompts) {

String apiDomain = cgManagerConfig.getApidomain();

String url = String.format(apiDomain + "/v1/openai/summary?question=%s&prompts=%s", question, prompts);

HttpRequest request = HttpRequest.get(url);

HttpResponse response = request.timeout(50000).executeAsync();

Map<String, Object> result = JSONObject.parse(response.body());

return result.get("d").toString();

} public List<String> get30DaysList(){

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd"); // 获取今天的日期

LocalDate today = LocalDate.now(); // 创建一个存储日期的列表

List<String> pastThirtyDaysList = new ArrayList<>();

// 循环获取过去30天的日期，并存储在列表中

for (int i = 1; i <= 30; i++) {

LocalDate date = today.minusDays(i);

pastThirtyDaysList.add(date.format(formatter));

}

return pastThirtyDaysList;

} public String getTuhaoType(Map<String, Object> tuhaoVariables) { Object typeObj = tuhaoVariables.get("field8811323120432"); String typeJsonString = null; if (typeObj instanceof Map) {

typeJsonString = JSONObject.toJSONString(typeObj);

}else{

typeJsonString = typeObj.toString();

} return JSONObject.parseObject(typeJsonString).get("key").toString();

} public Map<String,Object> getAllProjects(){

LocalDate currnentDate = LocalDate.now();

LocalDate last60Date = currnentDate.minusDays(60); List<Map<String, String>> projects = new ArrayList<>(); List<HistoricProcessInstance> allProjects = this.AllProjects("fbd9654e-4d19-4d28-b525-e7477a8b28b9"); List<HistoricProcessInstance> allTuhaos = this.AllTuhaos("fbd9654e-4d19-4d28-b525-e7477a8b28b9"); for (HistoricProcessInstance instance : allProjects) { Map<String, Object> projectVariables = instance.getProcessVariables(); // 项目令号

String projectCode = "";

if (projectVariables.get("field9645527299577") != null) {

projectCode = projectVariables.get("field9645527299577").toString();

} // 项目名称

String projectName = "";

if (projectVariables != null && projectVariables.get("field6232327254883") != null) {

projectName = projectVariables.get("field6232327254883").toString();

} int tuhaoCount = 0; Boolean isFinished = true; List<String> cDates = new ArrayList<>(); Boolean needUse = true; String projectCompleteDate = null; for (HistoricProcessInstance ins: allTuhaos) { Map<String, Object> tuhaoVariables = ins.getProcessVariables(); // 项目名称

Object tuhaoProjectNameJson = tuhaoVariables.get("field8324238372074");

if(tuhaoProjectNameJson == null){

continue;

}

String tuhaoProjectName = JSONObject.parseObject(tuhaoProjectNameJson.toString()).get("mainValue").toString();

// 项目令号

Object tuhaoProjectCodeJson = tuhaoVariables.get("field5517038455013");

if(tuhaoProjectCodeJson == null){

continue;

}

String tuhaoProjectCode = JSONObject.parseObject(tuhaoProjectCodeJson.toString()).get("mainValue").toString();

if (projectCode.equals(tuhaoProjectCode) && projectName.equals(tuhaoProjectName)) { tuhaoCount++; String type = this.getTuhaoType(tuhaoVariables); Object cDateObj = null;

if (type.equals("厂内")) {

// 发运 实际完成时间

cDateObj = tuhaoVariables.get("field1127223685439");

} else {

// 交货 实际完成时间

cDateObj = tuhaoVariables.get("field2045223747498");

} if(cDateObj != null && !cDateObj.toString().equals("")){

cDates.add(this.fixDate(cDateObj.toString()).toString());

} else {

isFinished = false;

}

}

} // log.info("cDates: {}", cDates);

if (tuhaoCount == 0) {

needUse = false;

} else if (tuhaoCount > 0 && isFinished) { String maxDate = cDates

.stream()

.map(s -> LocalDate.parse(s, DateTimeFormatter.ofPattern("yyyy-MM-dd")))

.max(LocalDate::compareTo)

.get()

.toString(); projectCompleteDate = maxDate; LocalDate completeDate = this.fixDate(maxDate); Instant completeInstant = completeDate.atStartOfDay().toInstant(ZoneOffset.UTC);

long completeTimestamp = completeInstant.toEpochMilli();

// log.info("completeTimestamp: {}", completeTimestamp);

Instant last60DateInstant = last60Date.atStartOfDay().toInstant(ZoneOffset.UTC);

long last60DateTimestamp = last60DateInstant.toEpochMilli();

if (completeTimestamp > last60DateTimestamp) {

needUse = false;

}

} if (needUse) {

// String projectName = ((JSONObject) formData.get("field8324238372074")).get("mainValue").toString();

//项目经理

Object projectManagerObj = projectVariables.get("field4105224493800");

Object projectManager = projectManagerObj != null ? JSONArray.parseArray(projectManagerObj.toString()).get(0): null;

String projectManagerName = projectManager != null ? ((JSONObject) projectManager).getString("name"):"未知"; String projectStartDate = projectVariables.get("field8532624451956") != null

? projectVariables.get("field8532624451956").toString() : null; projectStartDate = this.fixDate(projectStartDate).format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));

// 同一个projectCode和projectName只需要add一次

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

project.put("projectCode", projectCode);

project.put("projectName", projectName);

project.put("completeDate", projectCompleteDate);

project.put("projectStartDate", projectStartDate);

project.put("projectManager", projectManagerName); Boolean isExist = false;

for (Map<String, String> p : projects) {

String key = projectCode + projectName;

if (key.equals(p.get("projectCode") + p.get("projectName"))) {

isExist = true;

break;

}

}

if (!isExist) {

projects.add(project);

}

}

} log.info("projects:{}", projects); Map<String,Object> datas = new HashMap<>();

datas.put("projects",projects);

datas.put("allTuhaos",allTuhaos);

return datas;

} public LocalDate fixDate(String date){

LocalDate newDate = null;

if(date == null || date.equals("")){

return newDate;

} if(date.length()>10){

return newDate = LocalDate.parse(date.substring(0, 10));

} if(date.indexOf("/") > 0){

// log.info("date:{}",date);

List<String> dateList = Arrays.asList(date.split("/"));

if(dateList.size() == 3){

String mm = dateList.get(0);

String dd = dateList.get(1);

String yy = dateList.get(2);

//date:2023/12/23 2023/11/7

if(mm.length() == 4){

if(yy.length() == 1){

yy = "0" + yy;

}

if(dd.length() == 1){

dd = "0" + dd;

}

date = mm + "-" + dd + "-" + yy;

return LocalDate.parse(date);

} //1/10/2024

if(mm.length() == 1){

mm = "0" + mm;

} //1/9/2024

if(dd.length() == 1){

dd = "0" + dd;

} //1/9/24

if(yy.length() == 2){

yy = "20" + yy;

}

date = yy + "-" + mm + "-" + dd;

newDate = LocalDate.parse(date);

}

} newDate = LocalDate.parse(date);

return newDate;

} public Map<String,String> getProjectPlanInfo(String projectName, String projectCode){

String formId = "";

if (UserUtil.getEnv().equals("prod")) {

formId = "wf65bf20b0e4b0de990e1f0c2e";

} else {

formId = "wf65dee870e4b0e59ac55f67b3";

}

HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery(); query.processInstanceTenantId("fbd9654e-4d19-4d28-b525-e7477a8b28b9");

query.processDefinitionKey(formId); query.variableValueLike("field6232327254883", projectName); query.variableValueLike("field9645527299577", projectCode); query.includeProcessVariables(); HistoricProcessInstance instance = query.singleResult(); Map<String, Object> variables = instance.getProcessVariables(); Map<String,String> result = new HashMap<>();

Object projectStartDate = variables.get("field8532624451956"); LocalDate projectStartDateLocalDate = this.fixDate(projectStartDate.toString());

String projectStartDateStr = projectStartDateLocalDate.format(DateTimeFormatter.ofPattern("yyyy-MM-dd")); result.put("projectStartDate", projectStartDateStr); //项目经理

Object projectManagerObj = variables.get("field4105224493800");

Object projectManager = projectManagerObj != null ? JSONArray.parseArray(projectManagerObj.toString()).get(0): null;

String projectManagerName = projectManager != null ? ((JSONObject) projectManager).getString("name"):"未知";

result.put("projectManager", projectManagerName); if(projectStartDate == null || projectStartDate.equals("")){

log.info("项目计划开始时间为空，忽略跳过，projectStartDate: {}", projectStartDate);

return null;

} return result;

} @Override

@SuppressWarnings("unchecked")

public Object getDatas(String type) { String projectHealthKey = "getProjectHealthData\_" + type; String cache = jedis.get(projectHealthKey); if (cache != null) {

return JSONObject.parseObject(cache);

} if (type.equals("projects") || type.equals("managers")) {

return getDatasByProjectName(type, projectHealthKey);

} List<String> pastThirtyDaysList = this.get30DaysList();

Map<String, Map<String, Map<String, Integer>>> result = new HashMap<>();

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

Map<String,Object> datas = this.getAllProjects();

List<Map<String, String>> projects = (List<Map<String, String>>) datas.get("projects"); List<HistoricProcessInstance> allTuhaos = (List<HistoricProcessInstance>) datas.get("allTuhaos");

// Map<String, Map<String, String>> projectPlanInfos = new HashMap<>(); // log.info("pastThirtyDaysList:{}",pastThirtyDaysList.size()); // log.info("projectSize:{}", projects.size()); // pastThirtyDaysList.parallelStream().forEach(date -> {

for (String date : pastThirtyDaysList) { LocalDate currnentDate = LocalDate.parse(date);

LocalDate last30Date = currnentDate.minusDays(30); Map<String, Map<String, Integer>> stagesDataByDate = new HashMap<>(); // projects.parallelStream().forEach(p -> {

for (Map<String, String> p : projects) { String projectCode = p.get("projectCode");

String projectName = p.get("projectName");

String completeDate = p.get("completeDate");

// String projectManager = p.get("projectManager");

String projectStartDate = p.get("projectStartDate"); if(projectStartDate == null || projectStartDate.equals("")){

continue;

} LocalDate projectStartDateLocalDate = this.fixDate(projectStartDate);

String projectStartDateStr = projectStartDateLocalDate.format(DateTimeFormatter.ofPattern("yyyy-MM-dd")); boolean shouldContinue = false; // 排除未开始项目

if (projectStartDateStr == null) {

shouldContinue = true;

} else {

// 排除未来项目

long currentDateTimestamp = currnentDate.atStartOfDay().toInstant(ZoneOffset.UTC).toEpochMilli();

long projectStartDateTimestamp = LocalDate.parse(projectStartDateStr).atStartOfDay()

.toInstant(ZoneOffset.UTC).toEpochMilli();

if (projectStartDateTimestamp > currentDateTimestamp) {

shouldContinue = true;

}

} // 排除过去项目

if (completeDate != null) { long completeDateTimestamp = LocalDate.parse(completeDate).atStartOfDay().toInstant(ZoneOffset.UTC)

.toEpochMilli();

long last30DateTimestamp = last30Date.atStartOfDay().toInstant(ZoneOffset.UTC).toEpochMilli();

if (completeDateTimestamp < last30DateTimestamp) {

shouldContinue = true;

}

} if (shouldContinue) {

continue;

} LocalDate projectStartNew = this.fixDate(projectStartDateStr); Map<String, Map<String, Integer>> stagesData = this.getProjectStageHealth(projectCode, projectName,

projectStartNew, LocalDate.parse(date), allTuhaos); for (Map.Entry<String, Map<String, Integer>> entry : stagesData.entrySet()) {

String stage = entry.getKey();

Map<String, Integer> stageValues = entry.getValue();

// 获取当前阶段在该日期的累加数据，如果不存在则创建新的累加数据

Map<String, Integer> accumulatedData = stagesDataByDate.computeIfAbsent(stage, k -> new HashMap<>());

// 累加total值

accumulatedData.merge("total", stageValues.getOrDefault("total", 0), Integer::sum);

// 累加delay值

accumulatedData.merge("delay", stageValues.getOrDefault("delay", 0), Integer::sum);

}

} result.put(date, stagesDataByDate); } // Calculate percentage

if (type.equals("overall")) {

return this.getTotalAndDelayByDate(result, projectHealthKey);

// Map<String, Map<String, Integer>> overAll = new HashMap<>();

// for (Map.Entry<String, Map<String, Map<String, Integer>>> entry : result.entrySet()) {

// Map<String, Integer> oneDay = new HashMap<>();

// Integer total = 0;

// Integer delay = 0;

// Map<String, Map<String, Integer>> stages = entry.getValue();

// for (Map.Entry<String, Map<String, Integer>> stage : stages.entrySet()) {

// Map<String, Integer> data = stage.getValue();

// total += data.get("total");

// delay += data.get("delay");

// // data.put("percent", (int) percent);

// }

// oneDay.put("total",total);

// oneDay.put("delay",delay); // double percent = this.calculatePercentage(total, delay);

// oneDay.put("percent", (int) percent); // overAll.put(entry.getKey(),oneDay);

// }

// return overAll; } else { for (Map.Entry<String, Map<String, Map<String, Integer>>> entry : result.entrySet()) {

Map<String, Map<String, Integer>> stages = entry.getValue();

for (Map.Entry<String, Map<String, Integer>> stage : stages.entrySet()) {

Map<String, Integer> data = stage.getValue();

double percent = this.calculatePercentage(data.get("total"), data.get("delay"));

data.put("percent", (int) percent);

}

} }

String resultJson = JSONObject.toJSONString(result); // 将结果存入 Redis 缓存，并设置过期时间为一天

jedis.setex(projectHealthKey, 60 \* 60 \* 24, resultJson); return result;

} public Map<String, Map<String, Integer>> getTotalAndDelayByDate(Map<String, Map<String, Map<String, Integer>>> result, String projectHealthKey) {

Map<String, Map<String, Integer>> totalAndDelayByDate = new HashMap<>(); for (Map.Entry<String, Map<String, Map<String, Integer>>> entry : result.entrySet()) {

String date = entry.getKey();

Map<String, Map<String, Integer>> stagesDataByDate = entry.getValue(); // 创建累加数据的Map

Map<String, Integer> accumulatedData = totalAndDelayByDate.computeIfAbsent(date, k -> new HashMap<>());

// 初始化总和和延迟值

int total = 0;

int delay = 0; // 遍历该日期的阶段数据，并累加total和delay值

for (Map<String, Integer> stageData : stagesDataByDate.values()) {

total += stageData.getOrDefault("total", 0);

delay += stageData.getOrDefault("delay", 0);

} // 计算百分比

int percent = total != 0 ? (total - Math.abs(delay)) \* 100 / total : 0; // 添加到结果中

accumulatedData.put("total", total);

accumulatedData.put("delay", Math.abs(delay));

accumulatedData.put("percent", percent);

} String resultJson = JSONObject.toJSONString(totalAndDelayByDate); // 将结果存入 Redis 缓存，并设置过期时间为一天

jedis.setex(projectHealthKey, 60 \* 60 \* 24, resultJson);

return totalAndDelayByDate;

} public Map<String, Map<String, Map<String, Integer>>> getDatasByProjectName(String type, String projectHealthKey) {

List<String> pastThirtyDaysList = get30DaysList();

// ConcurrentHashMap<String, Map<String, Map<String, Map<String, Integer>>>> result = new ConcurrentHashMap<>();

Map<String, Map<String, Map<String, Map<String, Integer>>>> result = new HashMap<>();

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

Map<String,Object> datas = this.getAllProjects();

List<Map<String, String>> projects = (List<Map<String, String>>) datas.get("projects"); List<HistoricProcessInstance> allTuhaos = (List<HistoricProcessInstance>) datas.get("allTuhaos");

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

// log.info("pastThirtyDaysList:{}",pastThirtyDaysList.size()); // log.info("projectSize:{}",projects.size()); for (String date : pastThirtyDaysList) {

// pastThirtyDaysList.parallelStream().forEach(date -> {

LocalDate currnentDate = LocalDate.parse(date);

LocalDate last30Date = currnentDate.minusDays(30); // projects.parallelStream().forEach(p -> {

for (Map<String, String> p : projects) { String projectCode = p.get("projectCode");

String projectName = p.get("projectName");

String completeDate = p.get("completeDate");

String projectManager = p.get("projectManager");

String projectStartDate = p.get("projectStartDate"); if(projectStartDate == null || projectStartDate.equals("")){

continue;

} LocalDate projectStartDateLocalDate = this.fixDate(projectStartDate);

String projectStartDateStr = projectStartDateLocalDate.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));

// Extracted common logic

// log.info("completeDate:{}",completeDate);

if (completeDate != null) {

long completeDateTimestamp = LocalDate.parse(completeDate).atStartOfDay().toInstant(ZoneOffset.UTC)

.toEpochMilli();

long last30DateTimestamp = last30Date.atStartOfDay().toInstant(ZoneOffset.UTC).toEpochMilli();

if (completeDateTimestamp < last30DateTimestamp) {

continue;

}

}

long currentDateTimestamp = currnentDate.atStartOfDay().toInstant(ZoneOffset.UTC).toEpochMilli();

long projectStartDateTimestamp = LocalDate.parse(projectStartDateStr).atStartOfDay()

.toInstant(ZoneOffset.UTC).toEpochMilli();

if (projectStartDateTimestamp > currentDateTimestamp) {

continue;

}

LocalDate projectStartNew = fixDate(projectStartDateStr.toString()); // long beginTime = System.currentTimeMillis(); Map<String, Map<String, Integer>> stagesData = getProjectStageHealth(projectCode, projectName,

projectStartNew, LocalDate.parse(date), allTuhaos); // long endTime = System.currentTimeMillis();

// log.info("project Elapsed Time: {}", endTime - beginTime);

String key = "";

if(type.equals("projects")){

key = projectName+"|" + projectCode;

}else if(type.equals("managers")){

key = projectManager;

}

result.computeIfAbsent(date, k -> new HashMap<>()).put(key, stagesData);

}

}

// long beginTime = System.currentTimeMillis(); Map<String, Map<String, Map<String, Integer>>> newResult = new HashMap<>(); for (Map.Entry<String, Map<String, Map<String, Map<String, Integer>>>> date : result.entrySet()) { Map<String, Map<String, Map<String, Integer>>> projets = date.getValue(); Map<String, Map<String, Integer>> sumMap = new HashMap<>(); for (Map.Entry<String, Map<String, Map<String, Integer>>> p : projets.entrySet()) {

Map<String, Map<String, Integer>> stages = p.getValue(); Map<String, Integer> totalMap = new HashMap<>();

Integer total = 0;

Integer delay = 0;

for (Map.Entry<String, Map<String, Integer>> stage : stages.entrySet()) {

Map<String, Integer> data = stage.getValue();

total += data.get("total");

delay += data.get("delay");

}

totalMap.put("total", total);

totalMap.put("delay", delay);

sumMap.put(p.getKey(), totalMap);

}

newResult.put(date.getKey(), sumMap);

} for (Map.Entry<String, Map<String, Map<String, Integer>>> entry : newResult.entrySet()) {

Map<String, Map<String, Integer>> pp = entry.getValue();

for (Map.Entry<String, Map<String, Integer>> p : pp.entrySet()) {

Map<String, Integer> data = p.getValue();

double percent = this.calculatePercentage(data.get("total"), data.get("delay"));

data.put("percent", (int) percent);

}

}

// long endTime = System.currentTimeMillis();

// log.info("project Elapsed Time: {}", endTime - beginTime); String resultJson = JSONObject.toJSONString(newResult); // 将结果存入 Redis 缓存，并设置过期时间为一天

jedis.setex(projectHealthKey, 60 \* 60 \* 24, resultJson);

return newResult;

}

public double calculatePercentage(int total, int delay) {

return ((double) (total - Math.abs(delay)) / total) \* 100;

} public Map<String, Map<String, Integer>> getProjectStageHealth(String projectCode, String projectName, LocalDate projectStartDate, LocalDate calcDate, List<HistoricProcessInstance> allTuhaos){ // String date = calcDate.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));; // log.info("projectName: {}, projectCode: {}, projectStartDate: {}, calcDate: {}", projectName, projectCode, projectStartDate, date);

Map<String, Map<String, Integer>> totalStages = new HashMap<>(); for (HistoricProcessInstance ins: allTuhaos) { Map<String, Object> tuhaoVariables = ins.getProcessVariables(); if (tuhaoVariables != null && tuhaoVariables.get("field8324238372074") != null && tuhaoVariables.get("field5517038455013") != null) {

try {

TuhaoVo tuhaoVo = new TuhaoVo();

tuhaoVo.setInstanceId(ins.getId());

// 项目令号

Object tuhaoProjectCodeJson = tuhaoVariables.get("field5517038455013");

if(tuhaoProjectCodeJson==null || tuhaoProjectCodeJson.equals("")){

continue;

}

String tuhaoProjectCode = JSONObject.parseObject(tuhaoProjectCodeJson.toString()).get("mainValue").toString();

// log.info("tuhaoProjectCodeJson: {}", tuhaoProjectCodeJson);

// 项目名称

String tuhaoProjectNameJson = tuhaoVariables.get("field8324238372074").toString();

if(tuhaoProjectNameJson==null || tuhaoProjectNameJson.equals("")){

continue;

}

String tuhaoProjectName = JSONObject.parseObject(tuhaoProjectNameJson).get("mainValue").toString();

// log.info("tuhaoProjectCode: {}", tuhaoProjectCode);

if (projectCode.equals(tuhaoProjectCode) && projectName.equals(tuhaoProjectName)) { String stageData = this.getProcessInstanceCurrentNodeIdAndName(ins.getId()); String stageName = "";

if (stageData != null && stageData.indexOf("：") > 0) {

stageName = stageData.split("：")[1];

}

List<String> projectStages = null; String type = this.getTuhaoType(tuhaoVariables); if (type.equals("厂内")) {

projectStages = List.of( "设计", "工艺" ,"资料晒发","材料投入","生产","发运");

} else {

projectStages = List.of( "设计", "工艺" ,"资料晒发","交货");

} int currentStageIndex = projectStages.indexOf(stageName) + 1; String lastStageCompleteDate = null;

for (int i = 0;i < currentStageIndex; i++){

// log.info("projectStages: {}", projectStages);

// log.info("i: {}", i);

String tempStage = projectStages.get(i);

Map<String, Integer> stage = totalStages.get(tempStage); if(stage == null){

stage = new HashMap<>();

} Map<String, String> datas = this.parseTuhaoStatus(tuhaoVariables, tempStage); String planDate = datas.get("planDate");

// log.info("planDate: {},tempStage:{},instanceId:{}", planDate,tempStage,ins.getId());

if(planDate == null || planDate.equals("")){

continue;

} String completeDate = datas.get("completeDate");

// log.info("completeDate: {}", completeDate); LocalDate pDate = this.fixDate(planDate); if (tempStage.equals("设计")) {

if (completeDate != null && completeDate.toString().length() >= 10) {

LocalDate cDate = this.fixDate(completeDate);

Long daysDifference = ChronoUnit.DAYS.between(cDate, projectStartDate);

// log.info("daysDifference: {}", daysDifference);

Integer total = Math.abs(daysDifference.intValue()); // daysDifference为负数，表示提前延期

if (daysDifference < 0) {

Integer delay = daysDifference.intValue();

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", delay);

} else {

total = stage.get("total") + total;

delay = stage.get("delay") + delay;

stage.put("total", total);

stage.put("delay", delay);

} // totalStages.put(tempStage, stage);

} else {

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", 0);

} else {

total = stage.get("total") + total;

stage.put("total", total);

stage.put("delay", 0);

} // totalStages.put(tempStage, stage);

}

lastStageCompleteDate = completeDate;

} else {

// 比较 date1 和 date2

int maxNum = pDate.compareTo(calcDate); LocalDate maxDate = maxNum < 0 ? calcDate : pDate; Long daysDifference = ChronoUnit.DAYS.between(maxDate, projectStartDate);

// Long total = 0L;

Integer total = Math.abs(daysDifference.intValue());

// Long yanqi = 0L;

if (daysDifference < 0) {

// Long delayDays = ChronoUnit.DAYS.between(pDate, maxDate);

Integer delay = daysDifference.intValue();

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", delay);

} else {

total = stage.get("total") + total;

delay = stage.get("delay") + delay;

stage.put("total", total);

stage.put("delay", delay);

} // totalStages.put(tempStage, stage);

} else {

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", 0);

} else {

total = stage.get("total") + total;

stage.put("total", total);

stage.put("delay", 0);

} // totalStages.put(tempStage, stage);

}

}

} else {

if(lastStageCompleteDate == null){

break;

}

// log.info("id:{},lastStageCompleteDate:{}",ins.getId(),lastStageCompleteDate);

// log.info("tempStage:{}",tempStage);

LocalDate tempLastStagLocalDate = LocalDate.parse(lastStageCompleteDate);

LocalDate startDate = tempLastStagLocalDate.plusDays(1); if (completeDate != null && completeDate.toString().length() >= 10) {

LocalDate cDate = this.fixDate(completeDate);

Long daysDifference = ChronoUnit.DAYS.between(cDate, startDate);

Integer total = Math.abs(daysDifference.intValue()); // daysDifference为负数，表示提前延期

if (daysDifference < 0) {

Integer delay = daysDifference.intValue();

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", delay);

} else {

total = stage.get("total") + total;

delay = stage.get("delay") + delay;

stage.put("total", total);

stage.put("delay", delay);

} // totalStages.put(tempStage, stage);

} else {

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", 0);

} else {

total = stage.get("total") + total;

stage.put("total", total);

stage.put("delay", 0);

} // totalStages.put(tempStage, stage);

}

lastStageCompleteDate = completeDate;

} else {

// 比较 date1 和 date2

int maxNum = pDate.compareTo(calcDate);

LocalDate maxDate = maxNum < 0 ? calcDate : pDate; Long daysDifference = ChronoUnit.DAYS.between(maxDate, startDate);

// Long total = 0L;

Integer total = Math.abs(daysDifference.intValue());

// Long yanqi = 0L;

if (daysDifference < 0) {

// Long delayDays = ChronoUnit.DAYS.between(pDate, currentDate);

Integer delay = daysDifference.intValue();

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", delay);

} else {

total = stage.get("total") + total;

delay = stage.get("delay") + delay;

stage.put("total", total);

stage.put("delay", delay);

} // totalStages.put(tempStage, stage);

} else {

if (totalStages.get(tempStage) == null) {

stage.put("total", total);

stage.put("delay", 0);

} else {

total = stage.get("total") + total;

stage.put("total", total);

stage.put("delay", 0);

} // totalStages.put(tempStage, stage);

}

}

} // log.info("stage: {}", stage); // log.info("totalStages: {}", totalStages); Map<String, Integer> existStageData = totalStages.get(tempStage);

if (existStageData == null) {

totalStages.put(tempStage, stage);

} else {

Integer total = existStageData.get("total");

Integer delay = existStageData.get("delay"); Integer newTotal = stage.get("total") + total;

Integer newDelay = stage.get("delay") + delay; Map<String, Integer> newStageData = new HashMap<>();

newStageData.put("total", newTotal);

newStageData.put("delay", newDelay);

totalStages.put(tempStage, newStageData);

}

}

}

} catch (Exception e) {

e.printStackTrace();

}

} }

return totalStages;

}}</style>

// 退出登录

const logout = () => {

  ElMessageBox.confirm("您是否确认退出登录?", "温馨提示", {

    confirmButtonText: "确定",

    cancelButtonText: "取消",

    type: "warning"

  }).then(async () => {

    // 执行退出登录接口

    // await logoutApi();    // 清除信息

    userStore.setToken("");

    userStore.setOrganization([]);

    userStore.setUserInfo({});

    userStore.setCompanyInfo({});

    sessionStorage.clear();

    // 重定向到登陆页

    router.replace(LOGIN\_URL);

    ElMessage.success("退出登录成功！");

  });

};