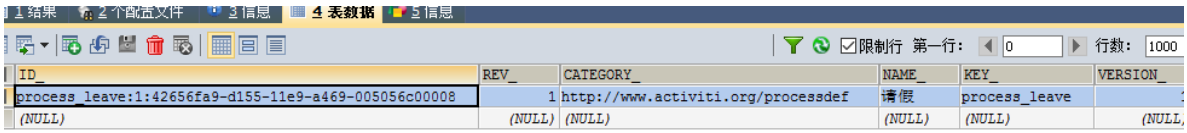
1 流程定义管理

在之前的课程,已经可以将一个bpmn文件上传到服务器中,并且部署到工作流引擎里。那么在activiti中就已经完成了流程定义（ProcessDefinition）。我们可以通过activiti提供的API对所有的流程定义（ProcessDefinition）进行操作：

- 查询所有已定义的流程
- 对流程的挂起/激活

流程定义表`act_re_procdef`



1.1 流程定义列表查询

（1）配置controller

```
// 查询所有的流程信息
@RequestMapping(value = "/process/definition", method = RequestMethod.GET)
public Result definitionList() {
    List list = processService.getProcessDefinitionList(companyId);
    return new Result(ResultCode.SUCCESS, list);
}
```

（2）配置service

```
//查询所有已部署流程
public List<ProcessDefinition> getProcessDefinitionList(String companyId) {
    return repositoryService.createProcessDefinitionQuery().
        processDefinitionTenantId(companyId).latestVersion().list();
}
```

1.2 流程定义的挂起与激活

从activit 5.11起开始，activiti已经支持流程挂起与激活的功能了。这也意味着如果某个流程暂时不用,则可以挂起功能暂停流程,这样的话,就避免流程删除引起的各类麻烦了。当流程被挂起之后，是不能继续操作此流程的实例的，否则会抛出异常

（1）配置controller

```
// 挂起和恢复流程
@RequestMapping(value = "/process/suspend/{processKey}", method =
RequestMethod.GET)
public Result setProcessAblitily(@PathVariable String processKey) {
    processService.suspendProcess(processKey);
    return new Result(ResultCode.SUCCESS);
}
```

```
//流程挂起/激活
public void suspendProcess(String processKey) {
    ProcessDefinition definition =
repositoryService.createProcessDefinitionQuery()
        .processDefinitionKey(processKey).latestVersion().singleResult();
    if(definition.isSuspended()) { //如果是挂起状态，这里激活此流程
        repositoryService.activateProcessDefinitionById(definition.getId());
    }else{
        //如果不是挂起状态，这里挂起此流程
        repositoryService.suspendProcessDefinitionById(definition.getId());
    }
}
```

1.3 加班流程分析

在一个人力资源系统中除了请假流程之外还有常见的加班以及离职流程。对于加班流程，主要包括加班当事人，当事人的直属领导，已经人力资源HR逐个进行审批，最后完成一个加班的流程。每个公司的加班流程细节上可能存在差异，但总体流程都差不多。下面我们一起来看一个简版的加班流程，如下：

- 当事人发起加班申请
- 直属领导根据实际情况进行审核，如果没有问题就通过，流程继续向后执行。
  如果有问题就可以不通过，流程就终止。
- 直属领导审批通过之后，进入人事部门复审阶段

1.4 离职流程分析

离职流程相对来说比较复杂，涉及人员较多，包含离职当事人，直属领导，人事HR，人事负责人，财务人员等

- 当事人发起离职申请
- 直属领导根据实际情况进行审核，如果没有问题就通过，流程继续向后执行。
- 直属领导审批通过之后，进入人事部门审核。
- 人事部门审核之后交由人事负责人复审
- 人事负责人审核完成之后，进入到财务进行终审，完成整个流程

2 业务流程概述

总所周知在activiti中具有自己的用户，用户组等概念。但是在实际的项目中很少用到，并且流程实例的发起，审核往往会和具体的业务挂钩。为了更好的进行流程控制并逐步审核，所以我们需要自己定义一些列表，结合已有的用户数据进行操作：

- 审批列表查询
- 流程申请信息查询
- 发起流程
- 提交流程（通过，驳回，撤销）

2.2 数据库表说明

（1）流程业务表

```
CREATE TABLE `proc_instance` (
  `process_id` varchar(45) NOT NULL COMMENT '流程实例ID',
  `process_key` varchar(45) DEFAULT NULL COMMENT '流程标识',
  `process_name` varchar(45) DEFAULT NULL COMMENT '流程名称',
  `process_definition_id` varchar(100) DEFAULT NULL COMMENT '流程定义ID',
  `process_state` varchar(3) DEFAULT NULL COMMENT '流程状态（0已提交；1审批中；2审批通过；3审批不通过；4撤销）',
  `user_id` varchar(45) DEFAULT NULL COMMENT '申请人ID',
  `username` varchar(50) DEFAULT NULL COMMENT '申请人',
  `proc_apply_time` datetime DEFAULT NULL COMMENT '申请时间',
  `proc_curr_node_user_id` varchar(45) DEFAULT NULL COMMENT '当前节点审批人ID',
  `proc_curr_node_user_name` varchar(45) DEFAULT NULL COMMENT '当前节点审批人',
  `proc_end_time` datetime DEFAULT NULL COMMENT '结束流程时间',
  `proc_data` longtext,
  `department_id` varchar(40) DEFAULT NULL,
  `department_name` varchar(40) DEFAULT NULL,
  `time_of_entry` datetime DEFAULT NULL,
  PRIMARY KEY (`process_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

（2）流程审批历史表

```
  `task_id` varchar(45) DEFAULT NULL COMMENT '任务实例ID',
  `task_key` varchar(45) NOT NULL COMMENT '任务节点key',
  `task_name` varchar(45) DEFAULT NULL COMMENT '任务节点',
  `should_user_id` varchar(45) DEFAULT NULL COMMENT '应审批用户ID',
  `should_user_name` varchar(2000) DEFAULT NULL COMMENT '应审批用户',
  `handle_user_id` varchar(2000) DEFAULT NULL COMMENT '实际处理用户ID',
  `handle_user_name` varchar(45) DEFAULT NULL COMMENT '实际处理用户',
  `handle_time` datetime DEFAULT NULL COMMENT '处理时间',
  `handle_opinion` varchar(45) DEFAULT NULL COMMENT '处理意见',
  `handle_type` varchar(45) DEFAULT NULL COMMENT '处理类型（2审批通过；3审批不通过；4
撤销）',
  PRIMARY KEY (`process_id`,`task_key`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

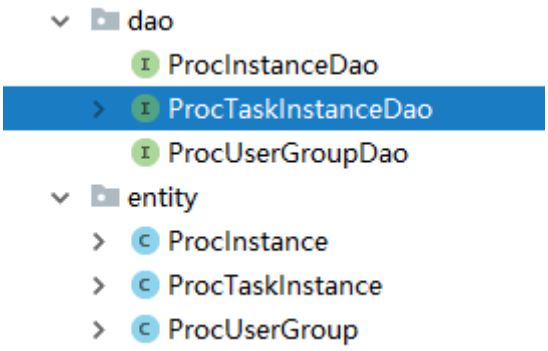
（3）自定义组表

```
CREATE TABLE `proc_user_group` (
  `id` varchar(45) NOT NULL COMMENT '主键',
  `name` varchar(100) DEFAULT NULL COMMENT '组名',
  `param` varchar(45) DEFAULT NULL COMMENT '入参',
  `isql` varchar(1000) DEFAULT NULL COMMENT '对应sql',
  `isvalid` varchar(2) DEFAULT NULL COMMENT '有效标记',
  `create_user` varchar(45) DEFAULT NULL COMMENT '创建人',
  `create_time` timestamp NULL DEFAULT NULL COMMENT '创建时间',
  `update_user` varchar(45) DEFAULT NULL COMMENT '最后更新人',
  `update_time` timestamp NULL DEFAULT NULL COMMENT '最后更新时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

为了更加灵活的处理流程，加入了自定义组表。在此表中通过SQL语句的形式和流程中的侯选组进行关联。也就意味着我们在操作流程的时候，需要通过组名称获取到待执行sql，执行得到获选人

2.3 环境搭建

将资料中提供的基本实体类，dao代码导入到项目中如下



3 审批管理

3.1 审批列表查询

（1）配置controller

```
RequestMethod.PUT)
public Result getProcessInstances(@RequestBody ProcInstance procInstance,int
page,int size) {
    //1.调用service完成查询
    Page pageUser = auditService.getProcessInstances(procInstance,page,size);
    //2.构造返回结果
    PageResult pageResult = new PageResult(pageUser.getTotalElements(),
pageUser.getContent());
    return new Result(ResultCode.SUCCESS, pageResult);
}
```

（2）配置service

```
//查询所有发起的流程
public Page getProcessInstances(ProcInstance in,int page,int size) {
    Specification<ProcInstance> spec = new Specification<ProcInstance>() {
        public Predicate toPredicate(Root<ProcInstance> root, CriteriaQuery<?>
criteriaQuery, CriteriaBuilder cb) {
            List<Predicate> list = new ArrayList<>();
            if(!StringUtils.isEmpty(in.getProcessState())) {
                Expression<String> exp = root.<String>get("processState");
                list.add(exp.in(in.getProcessState().split(",")));
            }
            if(!StringUtils.isEmpty(in.getUserId())) {

 list.add(cb.equal(root.get("userId").as(String.class),in.getUserId()));
            }
            if(!StringUtils.isEmpty(in.getProcessKey())) {

 list.add(cb.equal(root.get("processKey").as(String.class),in.getProcessKey()));
            }
            if(!StringUtils.isEmpty(in.getProcCurrNodeUserId())) {

 list.add(cb.like(root.get("procCurrNodeUserId").as(String.class),"%"+in.getProc
CurrNodeUserId()+"%"));
            }
            return cb.and(list.toArray(new Predicate[list.size()]));
        }
    };
    return procInstanceDao.findAll(spec, new PageRequest(page, size));
}
```

- 这里通过Specification的形式查询所有已经发起的流程
- 需要注意的时，对于待审批的流程需要传入参数（ProcCurrNodeUserId），这也就以为者此字段需要进行维护（如发起申请之后，需要查询下个节点的审批人，更新到此字段）。

3.2 审批详情

（1）配置controller

```
public Result detail(@PathVariable String id) {
    ProcInstance instance = auditService.findById(id);
    return new Result(ResultCode.SUCCESS, instance);
}
```

（2）配置service

```
//查询申请详情
public ProcInstance findById(String id) {
    return procInstanceDao.findById(id).get();
}
```

- 根据id获取发起的业务流程。在返回值中包含一个string类型字符串`procData`
- procData针对不同的流程，内容是不一样的。为了更加灵活的控制流程业务数据，需要将发起流程的所有信息已json的形式保存到此字段中

3.3 发起申请

3.3.1 需求分析

- 构造业务数据
  - 通用的业务属性保存到基本字段中
  - 不同申请的数据已json的形式存入到`ProcData`字段中
- 查询流程定义
  - 对于已经挂起的流程，抛出异常
- 开启流程
  - 需要设置流程参数（如请假时设置的请假天数days）
- 执行第一个task
  - 根据我们的流程定义，第一个task即为流程的发起
  - 保存业务明细数据
- 获取下个节点数据
  - 为了方便统一的进行查询，我们在业务表中定义了此流程的待审批人，此字段需要进行维护
  - 通过API查询下一个Task
  - 获取下一个task的侯选组
  - 根据侯选组查询已有的侯选组数据，获得对应的SQL语句
  - 执行SQL语句获取所有候选人数据

3.3.2 代码实现

（1）配置controller

```
//提交申请
@RequestMapping(value = "/process/startProcess", method = RequestMethod.POST)
public Result startProcess(@RequestBody Map map) {
    processService.startProcess(map);
    return new Result(ResultCode.SUCCESS);
}
```

```
//申请启动流程
public void startProcess(Map<String,Object> map) {
    String userId = (String)map.get("userId");
    String processKey =  (String)map.get("processKey");
    String processName =  (String)map.get("processName");
    User user = feignClientService.getUserInfoByUserId(userId);
    ProcInstance procInstance = new ProcInstance();
    BeanUtils.copyProperties(user,procInstance);
    procInstance.setUserId(userId);
    procInstance.setProcessId(idWorker.nextId()+"");
    procInstance.setProcApplyTime(new Date());
    procInstance.setProcessKey(processKey);
    procInstance.setProcessName(processName);
    procInstance.setProcessState("1");
    String data = JSON.toJSONString(map);
    procInstance.setProcData(data);

    //1.查询流程信息
    ProcessDefinition processDefinition =
repositoryService.createProcessDefinitionQuery().
        processDefinitionKey(processKey).latestVersion().singleResult();
    //2.开启流程
    Map<String, Object> startmap = new HashMap<>();
    ProcessInstance instance =
runtimeService.startProcessInstanceById(processDefinition.getId(),
procInstance.getProcessId(),startmap);
    //3.执行第一个节点(开始请假)
    Task task =
taskService.createTaskQuery().processInstanceId(instance.getId()).singleResult()
;

    taskService.complete(task.getId());
    //4.查询下个节点信息(查询执行人),存入到对象中
    Task next =
taskService.createTaskQuery().processInstanceId(instance.getId()).singleResult()
;
    //5.获取下一节点的候选人
    if(next != null) {
        List<User> users = findCurrUsers(next, user);
        String usernames = "", userIdS = "";
        for (User user1 : users) {
            usernames += user1.getUsername() + " ";
            userIdS += user1.getId();
        }
        procInstance.setProcCurrNodeUserId(userIdS);
        procInstance.setProcCurrNodeUserId(usernames);
    }
    procInstanceDao.save(procInstance);
}
```

3.4 提交申请

3.4.1 需求分析

- 根据流程id查询需要处理的task节点
- 根据不同的处理类型做响应的业务处理
  - 当操作类型为2：审核通过
  - 当操作类型为3：审核驳回，如果审核驳回可以将流程删除
  - 当操作类型为4：审核撤回，如果审核撤回可以将流程删除
- 更新业务流程信息
- 保存业务处理明细

3.4.2 代码实现

（1）配置controller

```
// 处理任务:2审核通过，3审核不通过，4撤销
@RequestMapping(value = "/process/instance/commit", method = RequestMethod.PUT)
public Result commitProcess(@RequestBody ProcTaskInstance procTaskInstance) {
    processService.commitProcess(procTaskInstance);
    return new Result(ResultCode.SUCCESS);
}
```

（2）配置service

```
    //提交流程
    public void commitProcess(ProcTaskInstance procTaskInstance) {
        //1.查询流程信息
        String processId = procTaskInstance.getProcessId();
        ProcInstance instance = procInstanceDao.findById(processId).get();
        instance.setProcessState(procTaskInstance.getHandleType());
        //2.获取待处理节点数据
        Task task =
taskService.createTaskQuery().processInstanceId(processId).singleResult();
        //3.处理节点
        String handleType = procTaskInstance.getHandleType();
        switch (handleType) {
            // 审核通过，完成当前任务节点
            case "2":
                taskService.complete(task.getId());
                Task next =
taskService.createTaskQuery().processInstanceId(processId).singleResult();
                //5.获取下一节点的候选人
                if(next != null) {
                    User user =
feignClientService.getUserInfoByUserId(instance.getUserId());
                    List<User> users = findCurrUsers(next, user);
                    String usernames = "", userIdS = "";
                    for (User user1 : users) {
                        usernames += user1.getUsername() + " ";
                        userIdS += user1.getId();
                    }
                    instance.setProcCurrNodeUserId(userIdS);
                    instance.setProcCurrNodeUserId(usernames);
                }
                break;
            // 审核不通过，删除流程。流程可以在历史库中查询
```

```
procTaskInstance.getHandleOpinion());
                break;
            // 撤销，删除流程。流程可以在历史库中查询
            case "4":
                runtimeService.deleteProcessInstance(processId,
procTaskInstance.getHandleOpinion());
                break;
        }
        procInstanceDao.save(instance);
        procTaskInstance.setHandleTime(new Date());
        procTaskInstanceDao.save(procTaskInstance);
    }
```