以请假流程为例

当部署了请假流程之后，只会有一个部署ID

流程部署对象：Deployment， 流程部署对象表 ACT_RE_DEPLOYMENT

当部署完了一个流程对象之后，会在流程定义表（ACT_RE_PROCDEF）添加一条数据，这个表的ID 由 表中 key：版本：随机数值 组合生成，DEPLOYMENT_ID_（部署ID） 同 流程部署表ID一致

有了流程定义就可以开启流程实例了

流程定义只有一个（相当于请假流程模板）

流程实例可以同时启动多个，相当于同时提交多个请假单，每一个请假单提交都会生成一个流程实例 ProcessInstance 获取 流程实例ID（getId），获取流程定义ID（getProcessDefinitionId）

每一个流程实例可以有一个或者多个执行对象（Execution），取决于流程是否执行分支。

任务(Task)是指流程执行到某一个环节产生的任务信息，包括当前任务执行人，开始时间，结束时间等

流程变量是指：比如提交一个请假单，带上请假天数，请假时间，请假原因等一些变量信息 就叫做流程变量.

- ACT_GE_PROPERTY ： 指定下一次部署流程的ID
- ACT_RE_DEPLOYMENT （部署对象表）： 存放流程定义的显示名和部署时间，每部署一次增加一条记录，ID_ 即为上面表所指定的值
- ACT_RE_PROCDEF（流程定义表）：

存放流程的属性信息，部署每个新的流程定义都会在这张表中增加一条记录，当流程定义的key相同的情况下，表中的版本字段会+1 。

- ACT_GE_BYTEARRAY （资源文件表）： 存储流程定义相关的部署信息。即流程定义文档的存放地。每部署一次就会增加两条记录，一条是关于bpmn

规则文件的，一条是图片的（如果部署时只指定了bpmn一个文件，activiti会在部署时解析bpmn文件内容自动生成流程图）。两个文件不是很大，都是以二进制形式存储在数据库中

- ACT_RU_EXECUTION（正在执行的执行对象表）
- ACT_HI_PROCINST（流程实例历史表） 流程没有走完也会有数据，只不过没有结束时间
- ACT_RU_TASK（正在执行的任务表）指定了任务状态，任务处理人等信息，如果一个任务执行完毕，那么表中就不会存在数据
- ACT_HI_TASKINST（任务历史表）只存放UserTask节点产生的任务，同样任务产生并没有结束就会存在数据，
- ACT_HI_ACTINST（所有活动节点的历史表）包括开始节点，结束节点，UserTask
- ACT_RU_VARIABLE（正在执行的流程变量表） 正在执行的任务流程变量表，如果一个任务执行完毕，那么不会存在该表中
- ACT_HI_VARINST（历史的流程变量表）

流程部署：

```
@Test
    public void deploymentProcessDefinition(){
        Deployment deployment =
processEngine.getRepositoryService()//与流程定义和部署对象相关的Service
                .createDeployment()//创建一个部署对象
                .name("hello入门程序02")//添加部署的名称

.addClasspathResource("bpmn/hello.bpmn")//从classpath的资源中加载，一次只能加载一个文件
                .addClasspathResource("bpmn/hello.png")//
从classpath的资源中加载，一次只能加载一个文件
                .deploy();//完成部署
        System.out.println("部署ID："+deployment.getId());//1
        System.out.println("部署名
称："+deployment.getName());//helloworld入门程序
```

```
}
```

查询流程定义：

```
@Test
    public void findProcessDefinition(){
        List<ProcessDefinition> list =
processEngine.getRepositoryService()//与流程定义和部署对象相关的Service
                            .createProcessDefinitionQuery()//创建一个流程定
义的查询
                            /**指定查询条件,where条件*/
//                          .deploymentId(deploymentId)//使用部署对象ID查
询
//                          .processDefinitionId(processDefinitionId)//使用流
程定义ID查询
//                          .processDefinitionKey(processDefinitionKey)//使
用流程定义的key查询
//
.processDefinitionNameLike(processDefinitionNameLike)//使用流程定义的名
称模糊查询

                            /**排序*/
                            .orderByProcessDefinitionVersion().asc()//按照版
本的升序排列
//                          .orderByProcessDefinitionName().desc()//按照流
程定义的名称降序排列

                            /**返回的结果集*/
                            .list();//返回一个集合列表，封装流程定义
//                          .singleResult();//返回惟一结果集
//                          .count();//返回结果集数量
//                          .listPage(firstResult, maxResults);//分页查询
```

```
            if(list!=null && list.size()>0){
                for(ProcessDefinition pd:list){
                    System.out.println("流程定义ID:"+pd.getId());//流程定义的key+版本+随机生成数
                    System.out.println("流程定义的名称:"+pd.getName());//对应hello.bpmn文件中的name属性值
                    System.out.println("流程定义的key:"+pd.getKey());//对应hello.bpmn文件中的id属性值
                    System.out.println("流程定义的版本:"+pd.getVersion());//当流程定义的key值相同的相同下，版本升级，默认1
                    System.out.println("资源名称bpmn文件:"+pd.getResourceName());
                    System.out.println("资源名称png文件:"+pd.getDiagramResourceName());
                    System.out.println("部署对象ID：  "+pd.getDeploymentId());

System.out.println("########################################################################");
                }
            }
        }
```

删除流程定义：

```
@Test
    public void deleteProcessDefinition(){
        //使用部署ID，完成删除
        String deploymentId = "501";
        /**
         * 不带级联的删除
         *   只能删除没有启动的流程，如果流程启动，就会抛出异常
         */
//        processEngine.getRepositoryService()//
//                                .deleteDeployment(deploymentId);
```

```
        /**
         * 级联删除
         *      不管流程是否启动，都能可以删除
         */
        processEngine.getRepositoryService()//
                                .deleteDeployment(deploymentId, true);
        System.out.println("删除成功！");
    }
```

查看流程文件（流程图）：

```
/**查看流程图
     * @throws IOException */
    @Test
    public void viewPic() throws IOException{
        /**将生成图片放到文件夹下*/
        String deploymentId = "1";
        //获取图片资源名称
        List<String> list = processEngine.getRepositoryService()//
                                .getDeploymentResourceNames(deploymentId);
        //定义图片资源的名称
        String resourceName = "";
        if(list!=null && list.size()>0){
            for(String name:list){
                if(name.indexOf(".png")>=0){
                    resourceName = name;
                }
            }
        }


        //获取图片的输入流
```

```
            InputStream in = processEngine.getRepositoryService()//
                                    .getResourceAsStream(deploymentId,
resourceName);

            //将图片生成到D盘的目录下
            File file = new File("D:/"+resourceName);
            //将输入流的图片写到D盘下
            FileUtils.copyInputStreamToFile(in, file);
        }
```

查询最新版本的流程定义：

```
/***附加功能：查询最新版本的流程定义*/
        @Test
        public void findLastVersionProcessDefinition(){
            List<ProcessDefinition> list =
processEngine.getRepositoryService()//
                                    .createProcessDefinitionQuery()//
                                    .orderByProcessDefinitionVersion().asc()//使用流
程定义的版本升序排列
                                    .list();
            /**
             * Map<String,ProcessDefinition>
 map集合的key：流程定义的key
 map集合的value：流程定义的对象
 map集合的特点：当map集合key值相同的情况下，后一次的值将替换前一次的
值
             */
            Map<String, ProcessDefinition> map = new LinkedHashMap<String,
ProcessDefinition>();
            if(list!=null && list.size()>0){
                for(ProcessDefinition pd:list){
```

```
                    map.put(pd.getKey(), pd);
                }
            }
            List<ProcessDefinition> pdList = new ArrayList<ProcessDefinition>
(map.values());
            if(pdList!=null && pdList.size()>0){
                for(ProcessDefinition pd:pdList){
                    System.out.println("流程定义ID:"+pd.getId());//流程定义的
key+版本+随机生成数
                    System.out.println("流程定义的名称:"+pd.getName());//对
应helloworld.bpmn文件中的name属性值
                    System.out.println("流程定义的key:"+pd.getKey());//对应
helloworld.bpmn文件中的id属性值
                    System.out.println("流程定义的版本:"+pd.getVersion());//当
流程定义的key值相同的相同下，版本升级，默认1
                    System.out.println("资源名称bpmn文
件:"+pd.getResourceName());
                    System.out.println("资源名称png文
件:"+pd.getDiagramResourceName());
                    System.out.println("部署对象ID："+pd.getDeploymentId());

System.out.println("##################################################
################");
                }
            }
        }
```

删除流程定义（删除key相同的所有不同版本的流程定义）：

```
@Test
    public void deleteProcessDefinitionByKey(){
        //流程定义的key
```

```
            String processDefinitionKey = "helloworld";
            //先使用流程定义的key查询流程定义，查询出所有的版本
            List<ProcessDefinition> list =
processEngine.getRepositoryService()//
                                    .createProcessDefinitionQuery()//
                                    .processDefinitionKey(processDefinitionKey)//使
用流程定义的key查询
                                    .list();
            //遍历，获取每个流程定义的部署ID
            if(list!=null && list.size()>0){
                for(ProcessDefinition pd:list){
                    //获取部署ID
                    String deploymentId = pd.getDeploymentId();
                    processEngine.getRepositoryService()//
                                        .deleteDeployment(deploymentId, true);
                }
            }
        }
```

启动流程实例:

```
@Test
    public void startProcessInstance(){
            System.out.println("启动流程实例");
            //流程定义的key
            String processDefinitionKey = "helloworld";//就是hello.bpmn在xml视
图中的 process 中的 id属性
            ProcessInstance pi = processEngine.getRuntimeService()//与正在执
行的流程实例和执行对象相关的Service

.startProcessInstanceByKey(processDefinitionKey);//使用流程定义的key启动
流程实例，key对应helloworld.bpmn文件中id的属性值，使用key值启动，默认
是按照最新版本的流程定义启动
```

```
        System.out.println("流程实例ID:"+pi.getId());//流程实例ID    101
        System.out.println("流程定义ID:"+pi.getProcessDefinitionId());//流程定义ID   helloworld:1:4

    }
```

查询当前人的个人任务:

```
@Test
    public void findMyPersonalTask(){
        System.out.println("查询当前人的个人任务");
        String assignee = "王五";
        List<Task> list = processEngine.getTaskService()//与正在执行的任务管理相关的Service
                                .createTaskQuery()//创建任务查询对象
                                .taskAssignee(assignee)//指定个人任务查询，指定办理人
                                .list();
        if(list!=null && list.size()>0){
            for(Task task:list){
                System.out.println("任务ID:"+task.getId());
                System.out.println("任务名称:"+task.getName());
                System.out.println("任务的创建时间:"+task.getCreateTime());
                System.out.println("任务的办理人:"+task.getAssignee());
                System.out.println("流程实例ID：  "+task.getProcessInstanceId());
                System.out.println("执行对象ID:"+task.getExecutionId());
                System.out.println("流程定义ID:"+task.getProcessDefinitionId());

System.out.println("################################################################");
            }
```

```
        }
    }
```

完成我的任务：

```
@Test
    public void completeMyPersonalTask(){
        System.out.println("完成我的任务");
        //任务ID
        String taskId = "302";
        processEngine.getTaskService()//与正在执行的任务管理相关的
Service
                .complete(taskId);
        System.out.println("完成任务：任务ID："+taskId);
    }
```

查询流程状态（判断流程正在执行，还是结束）：

```
    @Test
    public void isProcessEnd(){
        String processInstanceId = "701";
        ProcessInstance pi = processEngine.getRuntimeService()//表示正在
执行的流程实例和执行对象
                .createProcessInstanceQuery()//创建流程实例查
询
                .processInstanceId(processInstanceId)//使用流程
实例ID查询
                .singleResult();
        if(pi==null){
            System.out.println("流程已经结束");
```

```
        }
        else{
            System.out.println("流程没有结束");
        }
    }
```

设置流程变量：

设置流程变量支持多种类型，可以把一个java对象序列化之后存储。但是一定要在java对象 加上

private static final long serialVersionUID = 9025927816142981l3L;

否则一旦java类的字段变化了，反序列化的时候是没有办法提取数据的，会抛出异常

```
/**
     * 设置流程变量
     */
    @Test
    public void setProcessVar() {
//        runtimeService.setVariable(executionId, variableName, value)//表示使用执行对象ID，和流程变量的名称，设置流程变量的值（一次只能设置一个值）
//        runtimeService.setVariables(executionId, variables)//表示使用执行对象ID，和Map集合设置流程变量，map集合的key就是流程变量的名称，map集合的value就是流程变量的值（一次设置多个值）

//        taskService.setVariable(taskId, variableName, value)//表示使用任务ID，和流程变量的名称，设置流程变量的值（一次只能设置一个值）
//        taskService.setVariables(taskId, variables)//表示使用任务ID，和Map集合设置流程变量，map集合的key就是流程变量的名称，map集合的value就是流程变量的值（一次设置多个值）

//        runtimeService.startProcessInstanceByKey(processDefinitionKey,
```

```
variables);//启动流程实例的同时，可以设置流程变量，用Map集合
//          taskService.complete(taskId, variables)//完成任务的同时，设置流程变量，用Map集合

            TaskService taskService = processEngine.getTaskService();
            String taskId = "1804";
            taskService.setVariable(taskId, "天数", 3);
            taskService.setVariableLocal(taskId, "个人数据", "个人value");
            //taskService.setVariableLocal()  这里如果用local，那么该变量值只有当前任务看到。 如果 A提交的这个任务给B审核，B看不到这个属性，只有A可以在当前这个任务看到
            taskService.setVariable(taskId, "时间", new Date());
            taskService.setVariable(taskId, "原因", "旅2游");
            System.out.println("设置流程变量成功！");
        }
```

获取流程变量：

```
/**
     * 获取流程变量
     */
    @Test
    public void getProcessVar() {
//          runtimeService.getVariable(executionId, variableName);//使用执行对象ID和流程变量的名称，获取流程变量的值
//          runtimeService.getVariables(executionId);//使用执行对象ID，获取所有的流程变量，将流程变量放置到Map集合中，map集合的key就是流程变量的名称，map集合的value就是流程变量的值
//          runtimeService.getVariables(executionId, variableNames);//使用执行对象ID，获取流程变量的值，通过设置流程变量的名称存放到集合中，获取指定流程变量名称的流程变量的值，值存放到Map集合中

//          taskService.getVariable(taskId, variableName);//使用任务ID和流程变量的名称，获取流程变量的值
```

```
//          taskService.getVariables(taskId);//使用任务ID，获取所有的流程变量，将流程变量放置到Map集合中，map集合的key就是流程变量的名称，map集合的value就是流程变量的值
//          taskService.getVariables(taskId, variableNames);//使用任务ID，获取流程变量的值，通过设置流程变量的名称存放到集合中，获取指定流程变量名称的流程变量的值，值存放到Map集合中
            TaskService taskService = processEngine.getTaskService();
            String taskId = "2002";
            Integer days = (Integer) taskService.getVariable(taskId, "天数");
            Date date = (Date) taskService.getVariable(taskId, "时间");
            String str = (String) taskService.getVariable(taskId, "原因");
            String pd = (String) taskService.getVariable(taskId, "个人数据");
            System.out.println("天数：" + days + " , 时间：  " + date + " , 原因：" + str + " , 个人数据 : " + pd);
        }
```

查询历史任务:

```
    @Test
    public void findHistoryTask(){
            String taskAssignee = "张三";
            List<HistoricTaskInstance> list = processEngine.getHistoryService()//与历史数据（历史表）相关的Service
                                    .createHistoricTaskInstanceQuery()//创建历史任务实例查询
                                    .taskAssignee(taskAssignee)//指定历史任务的办理人
                                    .list();
            if(list!=null && list.size()>0){
                for(HistoricTaskInstance hti:list){
                    System.out.println(hti.getId()+"    "+hti.getName()+"    "+hti.getProcessInstanceId()+"   "+hti.getStartTime()+"
```

```
"+hti.getEndTime()+"   "+hti.getDurationInMillis());

System.out.println("################################");
                }
            }
        }
```

查询历史流程：

```
@Test
    public void findHistoryProcessInstance(){
        String processInstanceId = "1001";
        HistoricProcessInstance hpi = processEngine.getHistoryService()//
与历史数据（历史表）相关的Service
                .createHistoricProcessInstanceQuery()//创建历史
流程实例查询
                .processInstanceId(processInstanceId)//使用流程
实例ID查询
                .singleResult();
        System.out.println(hpi.getId()+"    "+hpi.getProcessDefinitionId()+"
"+hpi.getStartTime()+"    "+hpi.getEndTime()+"    "+hpi.getDurationInMillis());
    }
```