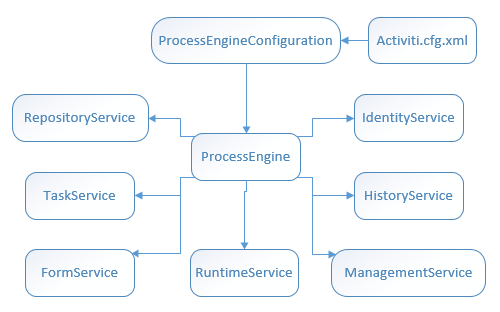
工作流，就是“业务过程的部分或整体在计算机应用环境下的自动化”，它主要解决的是“使在多个参与者之间按照某种欲定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”。

Activity：覆盖了业务流程管理、工作流、服务协作等领域的一个开源的、灵活的、易扩展的可执行流程语言框架

为什么选择activiti



核心7大接口、28张表



RespositoryService：提供一系列管理流程部署和流程定义的api

RuntimeService：在流程运行时对流程实例进行管理与控制

TaskService：对流程任务进行管理，例如任务提醒、任务完成和创建任务等

IdentityService：提供对流程角色数据进行管理的api，这些角色数据包括用户组、用户及它们之间的关系

ManagementService：提供对流程引擎进行管理和维护的服务

HistoryService：对流程的历史数据进行操作，包括查询、删除这些历史数据

FormService：表单服务

acitiviti数据库

act\_ge\_通用数据表，ge是general的缩写

|  |  |
| --- | --- |
| 数据表 | 描述 |
| act\_ge\_property | 属性表（保存流程引擎的kv键值属性） |
| act\_ge\_bytearray | 资源表（存储流程定义相关的资源） |

act\_hi\_历史数据表，hi是history的缩写，对应historyService接口

|  |  |
| --- | --- |
| 数据表 | 描述 |
| act\_hi\_procinst | 历史流程实例表 |
| act\_hi\_actinst | 历史节点信息表 |
| act\_hi\_taskinst | 历史任务表 |
| act\_hi\_varinst | 历史变量 |
| act\_hi\_identitylink | 历史参与者 |
| act\_hi\_detail | 历史变更 |
| act\_hi\_attachment | 附件 |
| act\_hi\_comment | 评论 |
| act\_hi\_log | 事件日志 |

act\_id\_身份数据表，id是identity的缩写，对应identityService接口

|  |  |
| --- | --- |
| 数据表 | 描述 |
| act\_id\_user | 用户的基本信息 |
| act\_id\_info | 用户的扩展信息 |
| act\_id\_group | 群组 |
| act\_id\_membership | 用户与群组关系 |

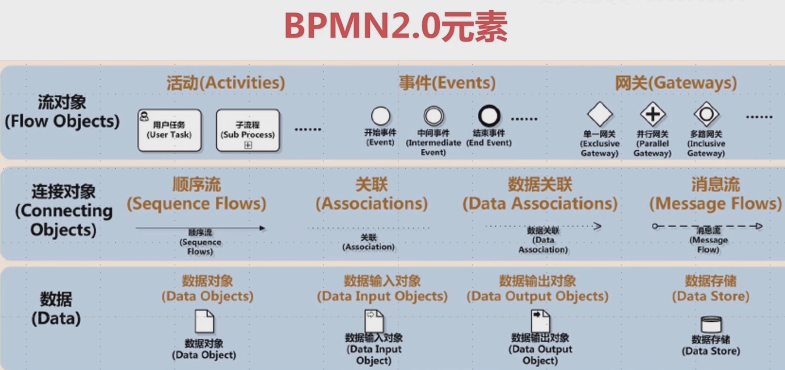
act\_re\_流程存储表，repository的缩写，对应repositoryService接口，存储流程部署和流程定义等静态数据

|  |  |
| --- | --- |
| 数据表 | 描述 |
| act\_re\_deployment | 流程部署记录表 |
| act\_re\_procdef | 流程定义信息表 |
| act\_re\_model | 模型信息表（用于web设计器） |
| act\_procdef\_info | 流程定义动态改变信息表 |

act\_ru\_运行时数据表，ru是runtime的缩写，对应runtimeService接口和taskService接口，存储流程实例和用户任务等动态数据。

|  |  |
| --- | --- |
| 数据表 | 描述 |
| act\_ru\_execution | 流程实例与分支执行信息 |
| act\_ru\_task | 用户任务信息 |
| act\_ru\_variable | 变量信息 |
| act\_ru\_identitylink | 参与者相关信息 |
| act\_ru\_event\_subscr | 事件监听表 |
| act\_ru\_job | 作业表 |
| act\_ru\_timer\_job | 定时器表 |
| act\_ru\_suspended\_job | 暂停作业表 |
| act\_ru\_deadletter\_job | 死信表 |

BPMN2.0规范













Activiti核心api

RepositoryService

流程存储服务：管理流程定义文件xml及静态资源的服务；对特定流程的暂停和激活；流程定义启动权限管理

部署文件构造器deploymentBuilder；部署文件查询器deploymentQuery；流程定义文件查询对象processDefinitionQuery。

可以获取到的对象：

流程部署文件对象deployment；流程定义文件对象ProcessDefinition；流程定义的java格式BpmnModel

用classpath方式部署流程定义文件

@Test  
**public void** testClasspathDeployment() **throws** Exception {  
 *//定义classpath* String bpmnClasspath = **"processes/candidateGroupProccess5-1.bpmn20.xml"**;  
 *//创建部署构建器* DeploymentBuilder deploymentBuilder = **repositoryService**.createDeployment();  
 *//添加资源* deploymentBuilder.addClasspathResource(bpmnClasspath);  
 *//执行部署* deploymentBuilder.deploy();  
 *//验证是否部署成功* ProcessDefinitionQuery processDefinitionQuery = **repositoryService**.createProcessDefinitionQuery();  
 **long** count = processDefinitionQuery.processDefinitionKey(**"userAndGroupInUserTask"**).count();  
 *assertEquals*(1, count);  
}

用inputstream方式部署流程资源文件

String filePath = **"E:\\workplace\\activiti\_cs\\src\\main\\resources\\processes\\candidateGroupProccess5-1.bpmn20.xml"**;  
*//读取classpath的资源为一个输入流*FileInputStream fileInputStream = **new** FileInputStream(filePath);  
**repositoryService**.createDeployment()  
 .addInputStream(**"candidateGroupProccess5-1.bpmn20.xml"**, fileInputStream).deploy();

用字符串方式部署

String text = **"<?xml version=…………"**;  
**repositoryService**.createDeployment()  
 .addString(**"candidateGroupProccess5-1.bpmn20.xml"**, text);

用压缩包方式部署

InputStream zipStream = getClass().getClassLoader()  
 .getResourceAsStream(**"process\_res/myprocess.zip"**);  
**repositoryService**.createDeployment()  
 .addZipInputStream(**new** ZipInputStream(zipStream)).deploy();

runtimeService

流程运行控制服务：启动流程及对流程数据的控制；流程实例（ProcessInstance）与执行流（Execution）查询；触发流程操作、接收消息和信号

RuntimeService启动流程及变量管理

RuntimeService启动流程及变量管理

启动流程的常用方式（id，key，message）；启动流程可选参数（businessKey，variables，tanantId）；变量（variables）的设置和获取

也可以通过processInstanceBuild.…….start()来完成启动

流程实例与执行流

流程实例（ProcessInstance）表示一次工作流业务的数据实体；

执行流（Execution）表示流程实例中具体的执行路径

流程实例接口继承于执行流

也可以通过message、signal启动

启动流程

ProcessInstance processInstance = **runtimeService**.startProcessInstanceByKey(**"userAndGroupInUserTask"**);

taskService

任务管理服务：对用户任务（UserTask）管理和流程的控制；设置用户任务（UserTask）的权限信息（拥有者、候选人、办理人）；针对用户任务添加任务附件、任务评论和事件记录

taskService对task管理与流程控制

task对象的创建、删除；查询task，并驱动task节点完成执行；task相关参数变量（variable）设置。

taskService.getVariables(taskId) taskService.getVariablesLoacal(taskId)

runtimeService.getVariables(taskId)

*//根据角色查询任务*Task task = **taskService**.createTaskQuery().taskCandidateUser(**"hiker"**).singleResult();  
*//签收任务***taskService**.claim(task.getId(), **"hiker"**);  
**taskService**.complete(task.getId());

添加批注信息（comment）

@Test  
**public void** addTaskComment() {  
 Task task = **taskService**.createTaskQuery().taskId(**"17520"**).singleResult();  
 *//利用任务对象获取任务实例id* String processInstancesId = task.getProcessInstanceId();  
 Authentication.*setAuthenticatedUserId*(**"dy"**);  
 String taskId = task.getId();  
 **taskService**.addComment(taskId, processInstancesId, **"agree"**);  
  
 Map<String, Object> variables = **new** HashMap<>();  
 variables.put(**"hrApproved"**, **true**);  
 **taskService**.complete(taskId, variables);  
}

获取批注信息

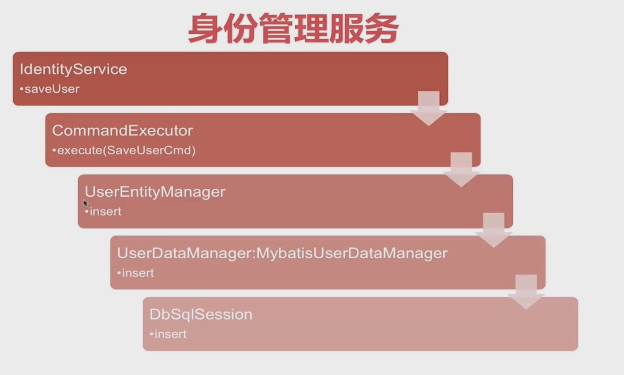
@Test  
**public void** findCommentByTaskId() {  
 String taskId = **"22507"**;  
 List list = **new** ArrayList();  
 *//Task task = taskService.createTaskQuery().taskId(taskId).singleResult();  
 //获取流程实例id  
 //String processInstanceId = task.getProcessInstanceId();  
 //使用流程实例id，查询历史任务，获取历史任务对应的每个任务id* List<HistoricTaskInstance> hitList = **historyService**.createHistoricTaskInstanceQuery().processFinished()  
 .list();  
 *//遍历集合，获取每个任务id* **if** (hitList != **null** && hitList.size() > 0) {  
 **for** (HistoricTaskInstance hti : hitList) {  
 *//任务id* String htaskId = hti.getId();  
 *//获取批注信息* List taskList = **taskService**.getTaskComments(htaskId);  
 list.addAll(taskList);  
 }  
 }  
  
 */\*List<Comment> comments = taskService.getProcessInstanceComments(processInstanceId);  
 for (Comment com : comments) {  
 System.out.println("ID:"+com.getId());  
 System.out.println("Message:"+com.getFullMessage());  
 System.out.println("TaskId:"+com.getTaskId());  
 System.out.println("ProcessInstanceId:"+com.getProcessInstanceId());  
 System.out.println("UserId:"+com.getUserId());  
 }\*/* System.***out***.println(list);  
}

*//根据当前人的ID查询*List<Task> todoList = **taskService**.createTaskQuery().processDefinitionKey(**"process\_ordinary\_form\_service"**).taskAssignee(userId).list();  
*//根据当前人未签收的任务*List<Task> unsignTasks = **taskService**.createTaskQuery().processDefinitionKey(**"process\_ordinary\_form\_service"**).taskCandidateUser(userId).list();

identityService

身份管理服务：管理用户（User）；管理用户组（Group）；用户与用户组的关系（Membership）

接口调用过程



创建一个用户对象：

*//创建一个用户对象*User user = **identityService**.newUser(**"hiker"**);  
user.setFirstName(**"chai"**);  
user.setLastName(**"shuai"**);  
user.setEmail(**"chaishuai@hikvision.com.cn"**);  
*//保存用户到数据库***identityService**.saveUser(user);  
*//验证用户是否保存成功*User userInDb = **identityService**.createUserQuery()  
 .userId(**"hiker"**).singleResult();  
*assertNotNull*(userInDb);  
*//删除用户***identityService**.deleteUser(**"hiker"**);

创建一个组对象

*//创建一个组对象*Group group = **identityService**.newGroup(**"deptLeader"**);  
group.setName(**"部门领导"**);  
group.setType(**"assignment"**);  
*//保存组***identityService**.saveGroup(group);  
*//验证组是否已成功保存*List<Group> groupList = **identityService**.createGroupQuery().groupId(**"deptLeader"**).list();  
*assertEquals*(1, groupList.size());  
*//s删除组***identityService**.deleteGroup(**"deptLeader"**);  
*//验证是否删除成功*groupList = **identityService**.createGroupQuery().groupId(**"deptLeader"**).list();  
*assertEquals*(0, groupList.size());

设置用户与组的关系

*//把用户加到组中***identityService**.createMembership(**"hiker"**, **"deptLeader"**);  
*//查询属于组的用户*User userInGroup = **identityService**.createUserQuery().memberOfGroup(**"deptLeader"**).singleResult();  
*assertNotNull*(userInGroup);  
*assertNotNull*(**"hiker"**, userInGroup.getId());  
*//查询所属组*Group groupContainsHiker = **identityService**.createGroupQuery().groupMember(**"hiker"**).singleResult();  
*assertNotNull*(groupContainsHiker);  
*assertEquals*(**"deptLeader"**, groupContainsHiker.getId());

清理用户与组

**identityService**.deleteMembership(**"hiker"**, **"deptLeader"**);  
**identityService**.deleteGroup(**"deptLeader"**);  
**identityService**.deleteUser(**"hiker"**);

FormService

表单管理服务：解析流程定义中表单项的配置；提交表单的方式驱动用户节点流转；获取自定义外部表单key

获取当前任务节点的form数据

List<FormProperty> lists = **formService**.getTaskFormData(task.getId()).getFormProperties();  
  
**if**(lists!=**null** && lists.size()>0) {  
 **for** (FormProperty formProperty : lists) {  
 System.***out***.println(formProperty.getId() + **" "** + formProperty.getName() + **" "** + formProperty.getValue());  
 }  
}

HistoryService

历史管理服务：管理流程实例结束后的历史数据；构建历史数据的查询对象；根据流程实例id删除流程历史数据

HistoryService历史数据实体

|  |  |
| --- | --- |
| 历史数据实体 | 描述 |
| HistoricProcessInstance | 历史流程实例实体类 |
| HistoricVariableInstance | 流程或任务变量值的实体 |
| HistoricActivityInstance | 单个活动节点执行的信息 |
| HistoricTaskInstance | 用户任务实例的信息 |
| HistoricDetail | 历史流程活动任务详细信息 |

HistoryService构建历史查询对象

create[历史数据实体]Query；createNative[历史数据实体]Query；createProcessInstanceHistoryLogQuery

historyService删除历史操作： deleteHistoricProcessInstance；deleteHistoricTaskInstance

获取用户任务实例的信息

*//使用流程实例id，查询历史任务，获取历史任务对应的每个任务id*List<HistoricTaskInstance> hitList = **historyService**.createHistoricTaskInstanceQuery().processFinished()  
 .list();

获取单个活动节点执行的信息

@Test  
**public void** historyActInstanceTest() {  
 List<HistoricActivityInstance> list = **historyService** *// 历史相关Service* .createHistoricActivityInstanceQuery() *// 创建历史活动实例查询* .processInstanceId(**"292549"**) *// 执行流程实例id* .finished()  
 .list();  
 **for** (HistoricActivityInstance hai : list) {  
 System.***out***.println(**"活动id"** + hai.getId());  
 System.***out***.println(**"流程实例ID:"** + hai.getProcessInstanceId());  
 System.***out***.println(**"活动名称："** + hai.getActivityName());  
 System.***out***.println(**"办理人："** + hai.getAssignee());  
 System.***out***.println(**"开始时间："** + hai.getStartTime());  
 System.***out***.println(**"结束时间："** + hai.getEndTime());  
 System.***out***.println(**"================================="**);  
 }  
}

获取历史流程活动任务详细信息

List<HistoricDetail> historicDetailsForm = **historyService**.createHistoricDetailQuery()  
 *//.taskId(hti.getId())  
 // .formProperties()* .processInstanceId(**"292549"**)  
 .list();  
  
**for**(HistoricDetail historicDetail : historicDetailsForm){  
 System.***out***.println(historicDetail.toString());  
 HistoricVariableUpdate historicVariableUpdate = (HistoricVariableUpdate) historicDetail;  
 String variableName = historicVariableUpdate.getVariableName();  
 Object value = historicVariableUpdate.getValue();  
 System.***out***.println(**"\*\*\*\*\*"**+variableName + **", \*\*\*\*\*"** + value);

ManagementService

Job任务管理；数据库相关通用操作；执行流程引擎命令（Command）

Job任务查询

|  |  |
| --- | --- |
| 工作查询对象 | 描述 |
| JobQuery | 查询一般工作 |
| TimerJobQuery | 查询定时工作 |
| SuspendedJobQuery | 查询中断工作 |
| DeadLetterJobQuery | 查询无法执行的工作 |

数据库相关操作

查询表结构元数据（TableMetaData）；通用表查询（TablePageQuery）；执行自定义的sql查询（executeCustomeSql）

获取工作流引擎配置参数相关信息：

**managementService**.getProperties();

异常策略

ActivitiException

|  |  |
| --- | --- |
| 异常名称 | 描述 |
| ActivitiWrongDbException | 引擎与数据库版本不匹配 |
| ActivitiOptimisticLockingException | 并发导致乐观锁异常 |
| ActivitiClassLoadingException | 加载类异常 |
| ActivitiObjectNotFoundException | 操作对象不存在 |
| ActivitiIllegalArgumentException | 非法的参数 |
| ActivitiTaskAlreadyClaimedException | 任务被重新声明代理人 |
| BpmnError | 定义业务异常，控制流程 |

candidateUsers、candidateGroup、assignee

分别表示候选人、候选人组、代理者。

当其中一个候选人签收了任务后，其自动成为代理人；另一个候选人再以候选者的身份查询此项任务时就查不到了

如果是通过指定候选人组的方式，则所有属于此组的候选者都可以签收任务。通过createMembership设置用户与组的关系。

只能指定一位代理者。

可以通过“,”分割多位候选者。

动态指定时，如下：

<**userTask id="deptExamine" name="部门审核" activiti:candidateUsers="${deptUserId}"**>

<**userTask id="modifyApply" name="调整申请" activiti:candidateGroup="${deptGroup}"**>

<**userTask id="modifyApply" name="调整申请" activiti:assignee="${applyUserId}"**>

*//用来设置启动流程的人员ID，引擎会自动把用户ID保存到activiti：initiator中***identityService**.setAuthenticatedUserId(userId);

*//設置部门审核人*variables.put(**"deptUserId"**, **"bill,candy"**); *//bill和candy都可以作为候选人*

ProcessInstance processInstance = **runtimeService**.startProcessInstanceByKey(**"process\_ordinary\_form\_service"**, businessKey, variables);

在任务节点上设置任务属性

<**userTask id="deptExamine" name="部门审核" activiti:candidateUsers="${deptUserId}"**>  
 <**extensionElements**>  
 <**activiti:formProperty id="deptMessage" name="申请信息" type="string" required="true"**></**activiti:formProperty**>  
 <**activiti:formProperty id="deptApproved" name="部门审批结果" type="string" required="true"**></**activiti:formProperty**>  
 </**extensionElements**>  
</**userTask**>

然后当前节点的代理人在处理此任务时，需要通过variables传入参数

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

variables.put(**"deptMessage"**, **"我同意"**);

variables.put(**"deptApproved"**, **"Y"**);

**taskService**.complete(taskId, variables);

然后传给gateway网关对象时，其顺序流就可以根据属性的取值去向不同的节点了。

<**sequenceFlow id="flow\_7" name="资源同意" sourceRef="gateway\_resource" targetRef="platformExamine"**>  
 <**conditionExpression xsi:type="tFormalExpression"**><![CDATA[${resourceApproved=="Y" || resourceApproved=="y"}]]></**conditionExpression**>  
</**sequenceFlow**>

<**sequenceFlow id="flow\_6" name="资源不同意" sourceRef="gateway\_resource" targetRef="receive\_reply"**>  
 <**extensionElements**>  
 <**activiti:executionListener event="end" delegateExpression="${reportBackEndProcessor}"**></**activiti:executionListener**>  
 </**extensionElements**>  
 <**conditionExpression xsi:type="tFormalExpression"**><![CDATA[${resourceApproved=="N" || resourceApproved=="n"}]]></**conditionExpression**>  
</**sequenceFlow**>

也可以为顺序流通过设置<**activiti:executionListener/>**监听，根据其事件类型可以选择其继承自TaskListener（complete等）或者ExecutionListener（end等）

使用ServiceTask，通过activiti:class 指定其关联到具体的java类

<**serviceTask id="receive\_reply" name="接收答复" activiti:class="pers.chai.activiti.common.ReceiveReplyTask"**>  
 <**extensionElements**>  
 <**activiti:field name="deptApproved"**>  
 <**activiti:expression**><![CDATA[${deptApproved}]]></**activiti:expression**>  
 </**activiti:field**>  
 <**activiti:field name="resourceApproved"**>  
 <**activiti:expression**><![CDATA[${resourceApproved}]]></**activiti:expression**>  
 </**activiti:field**>  
 <**activiti:field name="platApproved"**>  
 <**activiti:expression**><![CDATA[${platApproved}]]></**activiti:expression**>  
 </**activiti:field**>  
 </**extensionElements**>  
</**serviceTask**>

ReceiveReplyTask 可以继承JavaDelegate

**public class** ReceiveReplyTask **implements** JavaDelegate {  
  
 *//流程变量* **private** Expression **deptApproved**;  
 **private** Expression **resourceApproved**;  
 **private** Expression **platApproved**;  
  
 @Override  
 **public void** execute(DelegateExecution delegateExecution) **throws** Exception {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*可以在这里执行接收答复的逻辑"**);  
  
 String currentActivityName = delegateExecution.getCurrentActivityName();  
 String superExecutionId = delegateExecution.getSuperExecutionId();  
 String parentId = delegateExecution.getParentId();  
 Map<String, VariableInstance> variableInstances = delegateExecution.getVariableInstances();  
  
 **if** (variableInstances.containsKey(**"deptApproved"**)) {  
 **if** (**deptApproved** != **null**) {  
 String deptApprovedStr = (String) **deptApproved**.getValue(delegateExecution);  
 **if** (**"n"**.equalsIgnoreCase(deptApprovedStr)) {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*部门审核没通过"**);  
 }  
 }  
 }  
  
 **if** (variableInstances.containsKey(**"resourceApproved"**)) {  
 **if** (**resourceApproved** != **null**) {  
 String resourceApprovedStr = (String) **resourceApproved**.getValue(delegateExecution);  
 **if** (**"n"**.equalsIgnoreCase(resourceApprovedStr)) {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*资源审核没通过"**);  
 }  
 }  
 }  
 **if** (variableInstances.containsKey(**"platApproved"**)) {  
 **if** (**platApproved** != **null**) {  
 String platApprovedStr = (String) **platApproved**.getValue(delegateExecution);  
 **if** (**"n"**.equalsIgnoreCase(platApprovedStr)) {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*平台审核没通过"**);  
 }  
 }  
 }  
  
 String currentActivityId = delegateExecution.getCurrentActivityId();  
 String processDefinitionId = delegateExecution.getProcessDefinitionId();  
 String processInstanceId = delegateExecution.getProcessInstanceId();  
 String processBusinessKey = delegateExecution.getProcessBusinessKey();  
  
 System.***out***.println(**"currentActivityId——"** + currentActivityId);  
 System.***out***.println(**"processDefinitionId——"** + processDefinitionId);  
 System.***out***.println(**"processInstanceId——"** + processInstanceId);  
 System.***out***.println(**"processBusinessKey——"** + processBusinessKey);  
  
 }  
}