Activiti2

课前默写

1. 请写出BPM的意思
2. 请写出BPMN
3. 请写出Activiti的使用步骤

课程回顾

1. 工作流概述
2. BPM和BPMN
3. Activiti的应用
4. Activiti结构图

今日内容

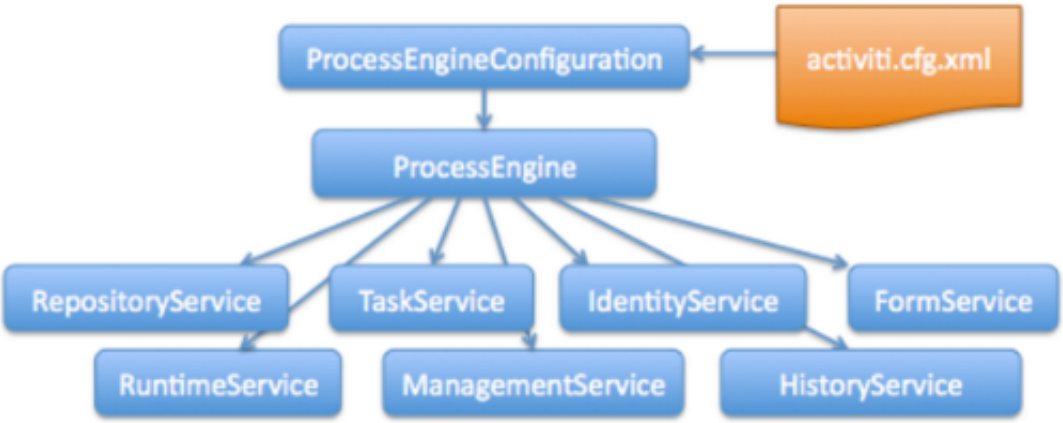
1. 工作流的基本理论
2. Activiti简介及BPMN
3. 使用IDEA搭建Activiti开发环境
4. Activiti的整体结构
5. Process流程定义和部署维护

教学目标

1. 掌握工作流的基本理论
2. 掌握Activiti简介及BPMN
3. 掌握使用IDEA搭建Activiti开发环境
4. 掌握Activiti的整体结构
5. 掌握Process流程定义和部署维护

总结

第六章 Activiti核心配置



ProcessEngine流程引擎是Activiti的核心。Activiti默认读取activiti.cfg.xml文件，获得ProcessEngineConfiguration对象，通过ProcessEngineConfiguration创建ProcessEngine。

6.1 ProcessEngineConfiguration配置

6.1.1 不使用配置文件

```
@Test
public void createEngineWithoutXml() {
ProcessEngineConfiguration processEngineConfiguration =
ProcessEngineConfiguration.createStandaloneInMemProcessEngineConfiguration(
);

    processEngineConfiguration.setJdbcDriver("com.mysql.jdbc.Driver");


 processEngineConfiguration.setJdbcUrl("jdbc:mysql://localhost:3306/activit
i?characterEncoding=utf-8");

    processEngineConfiguration.setJdbcUsername("root");

    processEngineConfiguration.setJdbcPassword("");

    processEngineConfiguration.setDatabaseSchemaUpdate("true");
    ProcessEngine processEngine =
processEngineConfiguration.buildProcessEngine();
    System.out.println(processEngine);
}
```

6.1.2 读取配置文件

配置文件activiti.cfg.xml，注意配置XML文件其实是一个spring的配置文件。 **但不是说Activiti只能用在Spring环境中！** 我们只是利用了Spring的解析和依赖注入功能 来构建引擎。

读取配置文件一共有五个方法：

1) ProcessEngineConfiguration.createProcessEngineConfigurationFromResourceDefault();

2) ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource);

3) ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource, String beanName);

4) ProcessEngineConfiguration.createProcessEngineConfigurationFromInputStream(InputStream inputStream);

5) ProcessEngineConfiguration.createProcessEngineConfigurationFromInputStream(InputStream inputStream, String beanName);

第一种方式：不带数据源DataSource

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="processEngineConfiguration"
class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">

    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/activiti?
useUnicode=true&amp;characterEncoding=UTF-8" />
    <property name="jdbcDriver" value="com.mysql.jdbc.Driver" />
    <property name="jdbcUsername" value="root" />
    <property name="jdbcPassword" value="xph666" />

    <property name="databaseSchemaUpdate" value="true" />

    <property name="jobExecutorActivate" value="false" />
    <property name="asyncExecutorEnabled" value="true" />
    <property name="asyncExecutorActivate" value="false" />

    <property name="mailServerHost" value="mail.my-corp.com" />
    <property name="mailServerPort" value="5025" />
  </bean>

</beans>
```

第二种方式：带有数据源DataSource

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
    <!--配置数据源-->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <!--配置数据库的中文乱码-->
        <property name="url" value="jdbc:mysql://localhost:3306/activiti?
useUnicode=true&amp;characterEncoding=UTF-8"/>
        <property name="username" value="root"/>
        <property name="password" value="root"/>
        <property name="maxActive" value="3"/>
        <property name="maxIdle" value="1"/>
    </bean>
    <bean id="processEngineConfiguration"
class="org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfigurati
on">
        <property name="dataSource" ref="dataSource"/>
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>
</beans>
```

测试类

```
@Test
    public void createEngineWithXml() {
        //默认读取的是activiti.cfg.xml，并且bean的id必须是
processEngineConfiguration
  ProcessEngineConfiguration processEngineConfiguration =
ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("ac
tiviti.cfg.xml");
        ProcessEngine processEngine =
processEngineConfiguration.buildProcessEngine();
        System.out.println(processEngine);
    }
```

如果自定义bean的id，可以这样读取：

```
ProcessEngineConfiguration processEngineConfiguration =

 ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("a
ctiviti.cfg.xml","configuration");
```

6.1.3 ProcessEngineConfiguration子类

创建ProcessEngineConfiguration时，目前可以使用如下类，以后会更多：

1) **org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration**: 单独运行的流程引擎。Activiti会自己处理事务。 默认，数据库只在引擎启动时检测 （如果没有Activiti的表或者表结构不正确就会抛出异常）。

2) **org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration**: 单元测试时的辅助类。Activiti会自己控制事务。 默认使用H2内存数据库。数据库表会在引擎启动时创建，关闭时删除。 使用它时，不需要其他配置（除非使用job执行器或邮件功能）。

3) **org.activiti.spring.SpringProcessEngineConfiguration**: 在Spring环境下使用流程引擎。

4) **org.activiti.engine.impl.cfg.JtaProcessEngineConfiguration**: 单独运行流程引擎，并使用JTA事务。

6.1.4 数据库配置

Activiti支持如下数据库：

数据库类型	url示例
h2	jdbc:h2:tcp://localhost/activiti
mysql	jdbc:mysql://localhost:3306/activiti?autoReconnect=true
oracle	jdbc:oracle:thin:@localhost:1521:xe
postgres	jdbc:postgresql://localhost:5432/activiti
db2	jdbc:db2://localhost:50000/activiti
mssql	jdbc:sqlserver://localhost:1433;databaseName=activiti (jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver) *OR* jdbc:jtds:sqlserver://localhost:1433/activiti (jdbc.driver=net.sourceforge.jtds.jdbc.Driver)

Activiti可能使用两种方式配置数据库。 第一种方式是定义数据库配置参数，另一种是直接配置DataSource。

直接配置数据库参数，有如下参数：

· **jdbcUrl**: 数据库的JDBC URL。

· **jdbcDriver**: 对应不同数据库类型的驱动。

· **jdbcUsername**: 连接数据库的用户名。

· **jdbcPassword**: 连接数据库的密码。

基于JDBC参数配置的数据库连接 会使用默认的MyBatis连接池。 下面的参数可以用来配置连接池（来自MyBatis参数）：

· **jdbcMaxActiveConnections**: 连接池中处于被使用状态的连接的最大值。默认为10。

· **jdbcMaxIdleConnections**: 连接池中处于空闲状态的连接的最大值。

· **jdbcMaxCheckoutTime**: 连接被取出使用的最长时间，超过时间会被强制回收。 默认为20000（20秒）。

· **jdbcMaxWaitTime**: 这是一个底层配置，让连接池可以在长时间无法获得连接时， 打印一条日志，并重新尝试获取一个连接。（避免因为错误配置导致沉默的操作失败）。 默认为20000（20秒）。

无论你使用JDBC还是DataSource的方式，都可以设置下面的配置：

· **databaseType**: 一般不用设置，因为可以自动通过数据库连接的元数据获取。 只有自动检测失败时才需要设置。 可能的值有：{h2, mysql, oracle, postgres, mssql, db2}。这个配置会决定使用哪些创建/删除脚本和查询语句。

· **databaseSchemaUpdate**: 设置流程引擎启动和关闭时如何处理数据库表，可能值如下：

1) false（默认）： 检查数据库表的版本和依赖库的版本， 如果版本不匹配就抛出异常。

2) true: 构建流程引擎时，执行检查，如果需要就执行更新。 如果表不存在，就创建。

3) create-drop: 构建流程引擎时创建数据库表， 关闭流程引擎时删除这些表。

使用数据源的配置参考各种DataSource的使用说明，ProcessEngineConfiguration的dataSource属性引用定义的数据源即可。如2.1.2中使用了DBCP的数据源。

6.2 ProcessEngine创建

```
@Test
public void createDefaultEngine() {
    //默认读取的是activiti.cfg.xml，并且bean的id必须是processEngineConfiguration
    ProcessEngine processEngine= ProcessEngines.getDefaultProcessEngine();
    System.out.println(processEngine);
}
```

ProcessEngines.getDefaultProcessEngine()方法读取的是activiti.cfg.xml，且bean的id必须是processEngineConfiguration。

使用ProcessEngineConfiguration创建：

```
ProcessEngineConfiguration processEngineConfiguration =

 ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("a
ctiviti.cfg.xml");
ProcessEngine processEngine =
processEngineConfiguration.buildProcessEngine();
System.out.println(processEngine);
```

6.3 数据库说明

Activiti的后台是有数据库的支持，所有的表都以ACT_开头。 第二部分是表示表的用途的两个字母标识。用途也和服务的API对应。

ACT_RE_*: 'RE'表示repository。 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。

ACT_RU_*: 'RU'表示runtime。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。

ACT_ID_*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。

ACT_HI_*: 'HI'表示history。 这些表包含历史数据，比如历史流程实例， 变量，任务等等。

ACT_GE_*: 通用数据， 用于不同场景下，如存放资源文件。

资源库流程规则表

1) act_re_deployment 部署信息表

2) act_re_model 流程设计模型部署表

3) act_re_procdef 流程定义数据表

4) act_re_event_subscr 事件监听

运行时数据库表

1) act_ru_execution 运行时流程执行实例表

2) act_ru_identitylink 运行时流程人员表，主要存储任务节点与参与者的相关信息

3) act_ru_task 运行时任务节点表

4) act_ru_variable 运行时流程变量数据表

5) act_ru_job 异步作业

历史数据库表

1) act_hi_actinst 历史节点表

2) act_hi_attachment 历史附件表

3) act_hi_comment 历史意见表，评论

4) act_hi_identitylink 历史流程人员表

5) act_hi_detail 历史详情表，提供历史变量的查询

6) act_hi_procinst 历史流程实例表

7) act_hi_taskinst 历史任务实例表

8) act_hi_varinst 历史变量表

组织机构表

1) act_id_group 用户组信息表

2) act_id_info 用户扩展信息表

3) act_id_membership 用户与用户组对应信息表

4) act_id_user 用户信息表

这四张表很常见，基本的组织机构管理，关于用户认证方面建议还是自己开发一套，组件自带的功能太简单，使用中有很多需求难以满足

通用数据表

1) act_ge_bytearray 二进制数据表，所有二进制内容保存到这个表，比如bpmn文件。

2) act_ge_property 属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录，

其它数据表

1) act_evt_log 事件日志，默认不开启

2) act_procdef_info 流程定义的动态变更信息

6.4 Activiti对象说明

a) 几个和流程相关的对象

Deployment：部署对象，和部署表(act_re_deployment)对应

ProcessDefinition：流程定义对象，和流程定义表(act_re_procdef)对应

ProcessInstance：流程实例对象，和流程实例表(act_ru_execution)对应

Task：任务对象，和任务表(act_ru_task)对应

b) 几个Service对象

RepositoryService：操作部署、流程定义等静态资源信息

RuntimeService：操作流程实例，启动流程实例、查询流程实例、删除流程实例等动态信息

TaskService：操作任务，查询任务、办理任务等和任务相关的信息

HistoryService：操作历史信息的，查询历史信息

IdentityService：操作用户和组

c) 几个Query对象

DeploymentQuery：对应查询部署表(act_re_deployment)

ProcessDefinitionQuery：对应查询流程定义表(act_re_procdef)

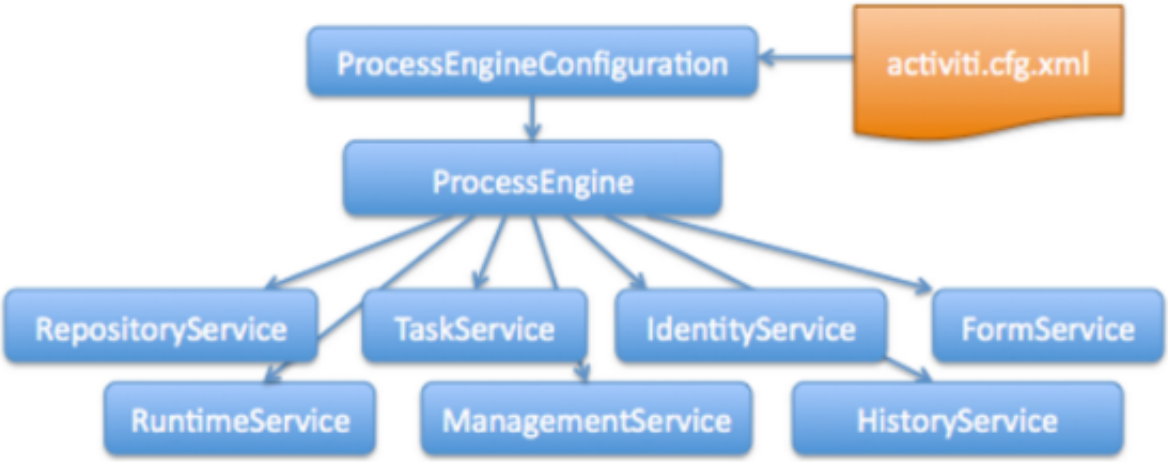
ProcessInstanceQuery：对应查询流程实例表(act_ru_execution)

TaskQuery：对应查询任务表(act_ru_task)

bpmn通过RepositoryService部署，引擎会把bpmn解析成可执行的东西，生成一个Deployment(部署)，对应的生成流程定义(ProcessDefinition)。

通过RuntimeService启动流程定义，得到一个流程实例。同时会生Task(任务)，任务是绑定到用户的，记录当前操作者是谁，任务状态是什么，用TaskService操作任务。

第七章 Service服务



所有的Service都通过流程引擎获得。

7.1 仓库服务

RepositoryService

是存储相关的服务，一般用来部署流程文件，获取流程文件（bpmn和图片），查询流程定义信息等操作，是引擎中的一个重要的服务。部署流程定义操作的数据库表有：部署表(act_re_deployment)、流程定义表(act_re_procdef)和二进制表(act_ge_bytearray)

```
/获取仓库服务
RepositoryService repositoryService= processEngine.getRepositoryService();
//部署流程
Deployment deployment =repositoryService.createDeployment()
        .name("请假流程")
        .addClasspathResource("leave.bpmn")
        .deploy(); // 完成部署
ProcessDefinitionQuery query =
repositoryService.createProcessDefinitionQuery();
// 根据流程定义的key来过滤
query.processDefinitionKey("leave2");
// 添加排序条件
query.orderByProcessDefinitionVersion().desc();
//分页查询，从哪开始，查询几条
query.listPage(0, 2);
// 查询的是所有的流程定义
List<ProcessDefinition> list = query.list();
for (ProcessDefinition pd : list) {
    System.out.println(pd.getId() + "-" + pd.getName() + "-" +
pd.getVersion());
}
//根据id删除流程
repositoryService.deleteDeployment("1005");
```

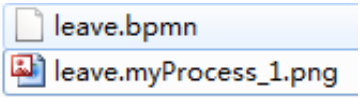
部署文件实际是一个bpmn文件和一个png图片

ID_	REV_	NAME_	DEPLOYMENT_ID_	BYTES_	GENERATED_
12502	1	leave.bpmn	12501	(BLOB)	0
12503	1	leave.myProcess_1.png	12501	(BLOB)	1

部署bpmn的时候，系统自动生成了一个png图片，可以下载查看：

```
RepositoryService repositoryService = processEngine.getRepositoryService();
String deploymentId = "72501";//部署id
List<String> names =
repositoryService.getDeploymentResourceNames(deploymentId);
for (String name : names) {
    System.out.println(name);
    // 获得两个流程定义文件对应的输入流
    InputStream in = repositoryService.getResourceAsStream(deploymentId,
name);
    // 读取输入流写到指定的本地磁盘上
    FileCopyUtils.copy(in, new FileOutputStream("F:/" + name));
    in.close();
}
```

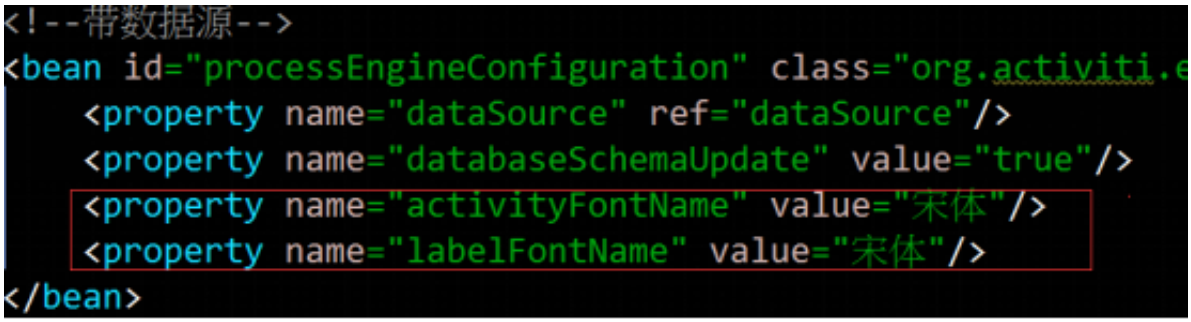
去F盘下可以看到两个文件：



自动生成的png图片中文会乱码，解决这个问题需要在processEngineConfiguration的bean中配置字体，如图：

```
<property name="activityFontName" value="宋体"/>

<property name="labelFontName" value="宋体"/>
```



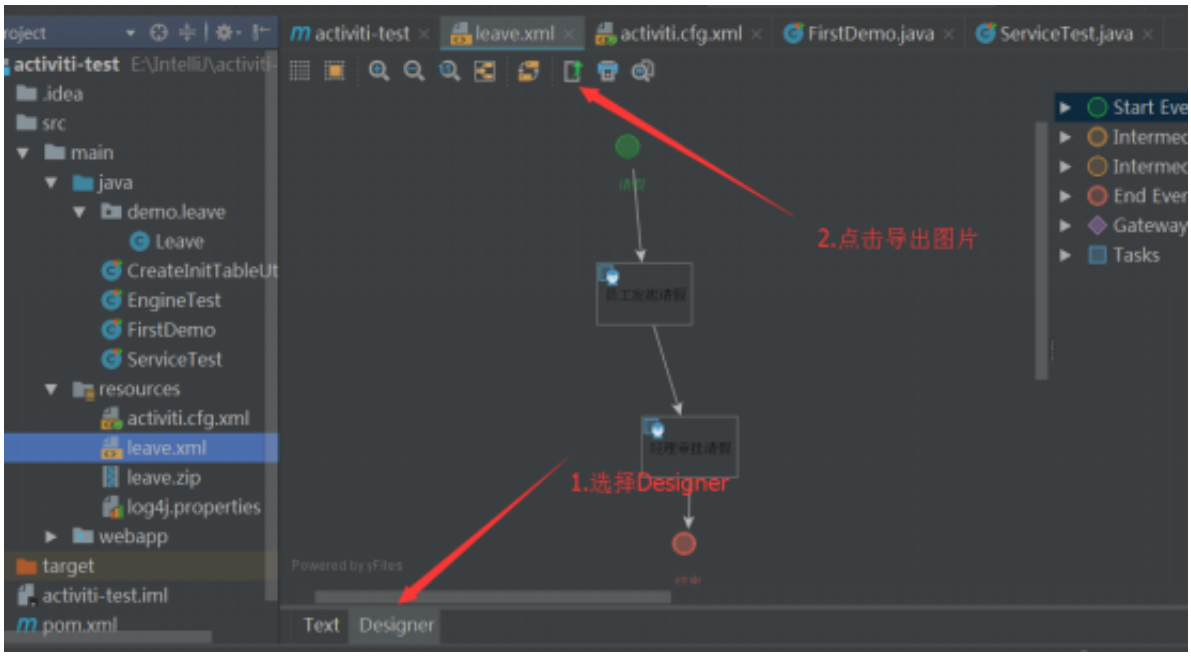
如果不想生成png图片，可以配置如下属性：

还可以将png和bpnm添加到zip压缩包中，部署zip压缩包。

```
// 从类路径下读取leave.zip压缩文件，并把它包装成一个输入流
ZipInputStream zipInputStream = new ZipInputStream(this.getClass()
        .getClassLoader().getResourceAsStream("leave.zip"));
Deployment deployment = repositoryService.createDeployment()
        .addZipInputStream(zipInputStream)
        .deploy(); // 完成部署
```

Activiti会把压缩包内的bpmn和png添加到数据库，即数据库里存的是解压后的两个文件而不是zip。

使用idea生成bpmn对应的png，需要把bpmn的文件后缀名改成xml



根据流程定义id获取文件流：

```
String processDefinitionId = "leave2:5:62504"; // 流程定义id
// 直接获得png图片的名称
// 根据流程定义id查询流程定义对象
ProcessDefinitionQuery query =
repositoryService.createProcessDefinitionQuery();
query.processDefinitionId(processDefinitionId);
ProcessDefinition processDefinition = query.singleResult();
// 根据流程定义对象获得png图片的名称,getResourceName()是获得bpmn
String pngName = processDefinition.getDiagramResourceName();
// 直接获得png图片对应的输入流
InputStream pngStream =
repositoryService.getProcessDiagram(processDefinitionId);
// 读取输入流写到指定的本地磁盘上
FileCopyUtils.copy(pngStream, new FileOutputStream("F:/" + pngName));
pngStream.close();
```

7.2 运行时服务

RuntimeService

流程运行时的流程实例，流程定义，流程版本，流程节点等信息，使用运行时服务操作，是引擎中的一个重要的服务，启动流程实例操作的数据表有流程实例表(act_ru_execution)、任务表(act_ru_task)。

流程运行时的流程实例，流程定义，流程版本，流程节点等信息，使用运行时服务操作，是引擎中的一个重要的服务

```
RuntimeService runtimeService = processEngine.getRuntimeService();
//key是act_re_procdef中的KEY_,即bpmn文件的id
runtimeService.startProcessInstanceByKey("leave2");
//或者根据流程定义id启动，id是act_re_procdef中的ID_，推荐使用
runtimeService.startProcessInstanceById("leave2:5:62504");

一个KEY_对应多个ID_，即一个流程定义可以产生多个流程实例。
//查询流程，操作的是流程实例表(act_ru_execution)
ProcessInstanceQuery query = runtimeService.createProcessInstanceQuery();
List<ProcessInstance> list = query.list();
删除一个流程实例
String processInstanceId = "1001"; // 流程实例id
String deleteReason = "不请假了"; // 删除原因，任君写
runtimeService.deleteProcessInstance(processInstanceId, deleteReason);
```

7.3 任务服务

TaskService

流程运行时的会产生任务，接收、办理、完成等操作使用任务服务完成，是引擎中的一个重要的服务

对任务进行查询、接收、办理、完成等操作。查询任务操作的数据表是任务表(act_ru_task)

```
TaskService taskService = processEngine.getTaskService();
//创建查询对象,指定查询人,进行查询
List<Task> taskList
=taskService.createTaskQuery().taskAssignee("ZhangSan").list();

//提交任务到下一个代理人
taskService.complete(task.getId());
```

7.4 认证服务

IdentityService

流程运行过程中的一些用户信息，组信息等操作使用认证服务，但是认证服务一般只作为辅助，每一个系统都有一个比较完整的人员系统，创建用户和用户组等操作。一般不使用自带的认证。

```
//添加用户组
IdentityService identityService = processEngine.getIdentityService();
Group groupEntity = identityService.newGroup("1001");
groupEntity.setName("超级管理员");
groupEntity.setType("administrator");
identityService.saveGroup(groupEntity);//保存用户组
//删除用户组
identityService.deleteGroup("1");

//添加用户
User user = identityService.newUser("10001");
user.setEmail("admin@sina.com");
user.setFirstName("zhang");
user.setLastName("san");
user.setPassword("admin");
identityService.saveUser(user);//保存用户
identityService.deleteUser("10001");//删除用户
```

7.5 历史服务

HistoryService

流程运行时，和运行完成之后的一些历史信息，包括历史任务，历史节点等，是引擎中的一个重要的服务

```
HistoryService historyService = processEngine.getHistoryService();
//查询历史运行信息
HistoricProcessInstance historicProcessInstance =
        historyService
                .createHistoricProcessInstanceQuery()
                .processInstanceId("40001").singleResult();
System.out.println("开始时间: " + historicProcessInstance.getStartTime());
System.out.println("结束时间: " + historicProcessInstance.getEndTime());
```

7.6 表单服务

FormService

流程运行时的任务表单信息，是引擎中的一个辅助的服务

```
/** 表单服务 */
FormService formService = engine.getFormService();
```

7.7 ManagementService

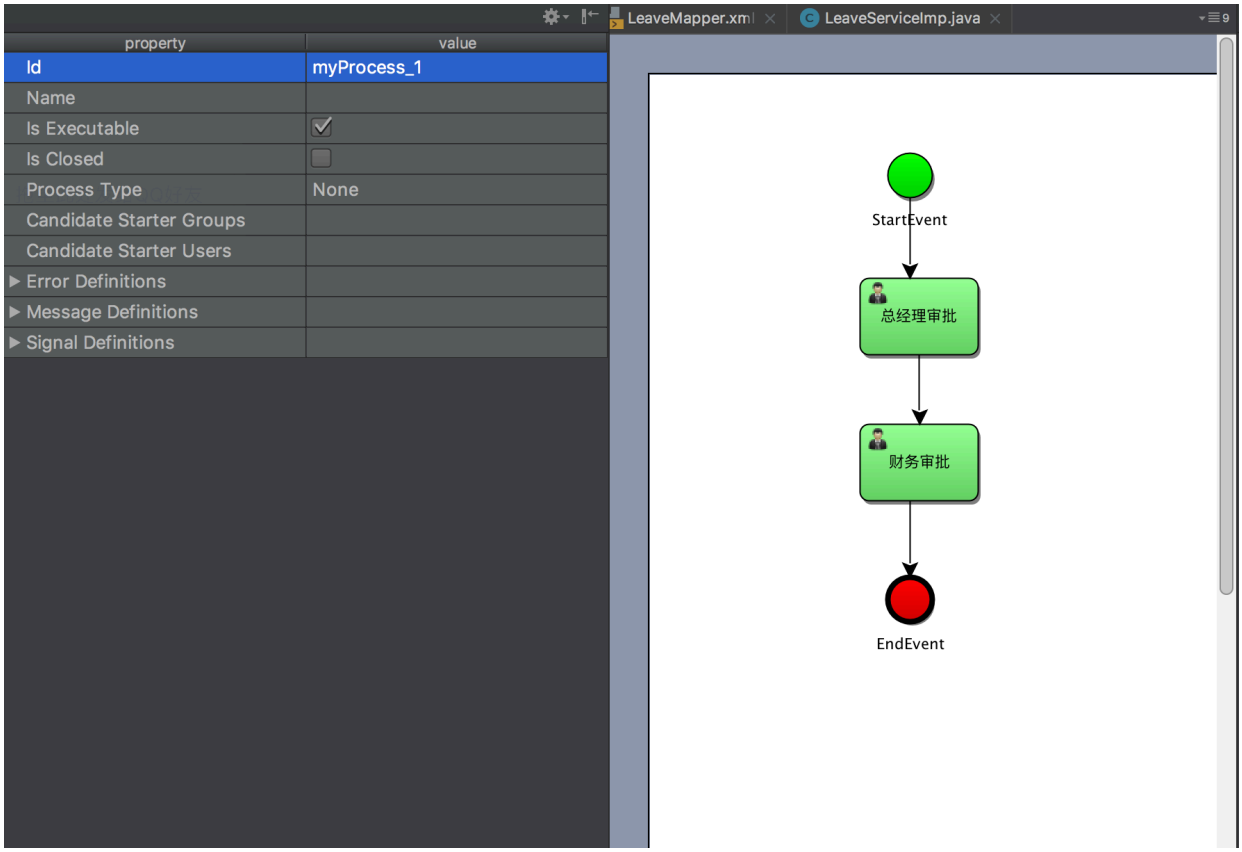
Management Service 提供了对 Activiti 流程引擎的管理和维护功能，这些功能不在工作流驱动的应用程序中使用，主要用于 Activiti 系统的日常维护。

第八章 任务管理

8.1 启动个人任务的三种方式PersonalTask

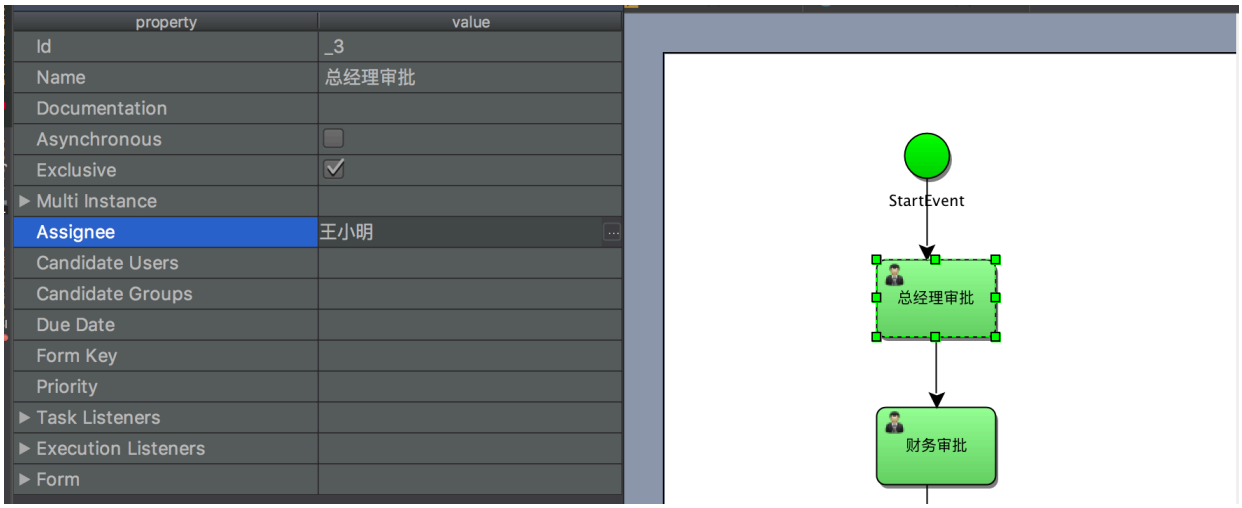
个人任务PersonalTask指的是操作的任务，个人任务是当前节点的执行者。

BPMN流程图



第一种：直接指定办理人

在流程图中直接配置任务节点，指定办理人为“王小明”，这个中方式仅仅限于测试代码使用，实际开发中并不常用。



测试代码

1、发布流程

```
/**
 * 发布流程
 *
 * @param processFileName
 */
public void deployProcess(String processFileName) {
    InputStream inputStream =

 this.getClass().getClassLoader().getResourceAsStream(processFileName);
    RepositoryService repositoryService = manager.getRepositoryService();
    repositoryService.createDeployment().addInputStream(processFileName,
inputStream).deploy();
}
```

2、启动流程

```
public void startProcessInstance2(){
    RuntimeService runtimeService = manager.getRuntimeService();
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey("myProcess_1");
    System.out.println("流程实例ID:" + processInstance.getId());
    System.out.println("流程定义ID:" +
processInstance.getProcessDefinitionId());

}
```

3、查看我的个人待办事宜

```
TaskService taskService = manager.getTaskService();
TaskQuery taskQuery = taskService.createTaskQuery();
String userID = "王小明";
//获取运行时状态
RuntimeService runtimeService = manager.getRuntimeService();
//如何获取业务流程中的bussinessKey，可以通过个人待办任务获取流程实例的ID，最终来获取
bussinessKey
taskQuery.taskAssignee(userID);
taskQuery.processDefinitionKey(processKey);
List<Task> list = taskQuery.list();
        for (Task task : list) {
            System.out.println(task.getId());
            System.out.println(task.getAssignee());
            System.out.println(task.getName());
            System.out.println(task.getOwner());
            System.out.println(task.getCategory());
            System.out.println(task.getCreateTime());
            System.out.println(task.getExecutionId());
            System.out.println(task.getTaskDefinitionKey());
            System.out.println(task.getProcessDefinitionId());
        }
```

4、完成我的个人待办事宜

```
@Test
public void completeTask(){
    String taskId = "3209";
    processEngine.getTaskService()//
                .complete(taskId);//
    System.out.println("完成任务");
}
```

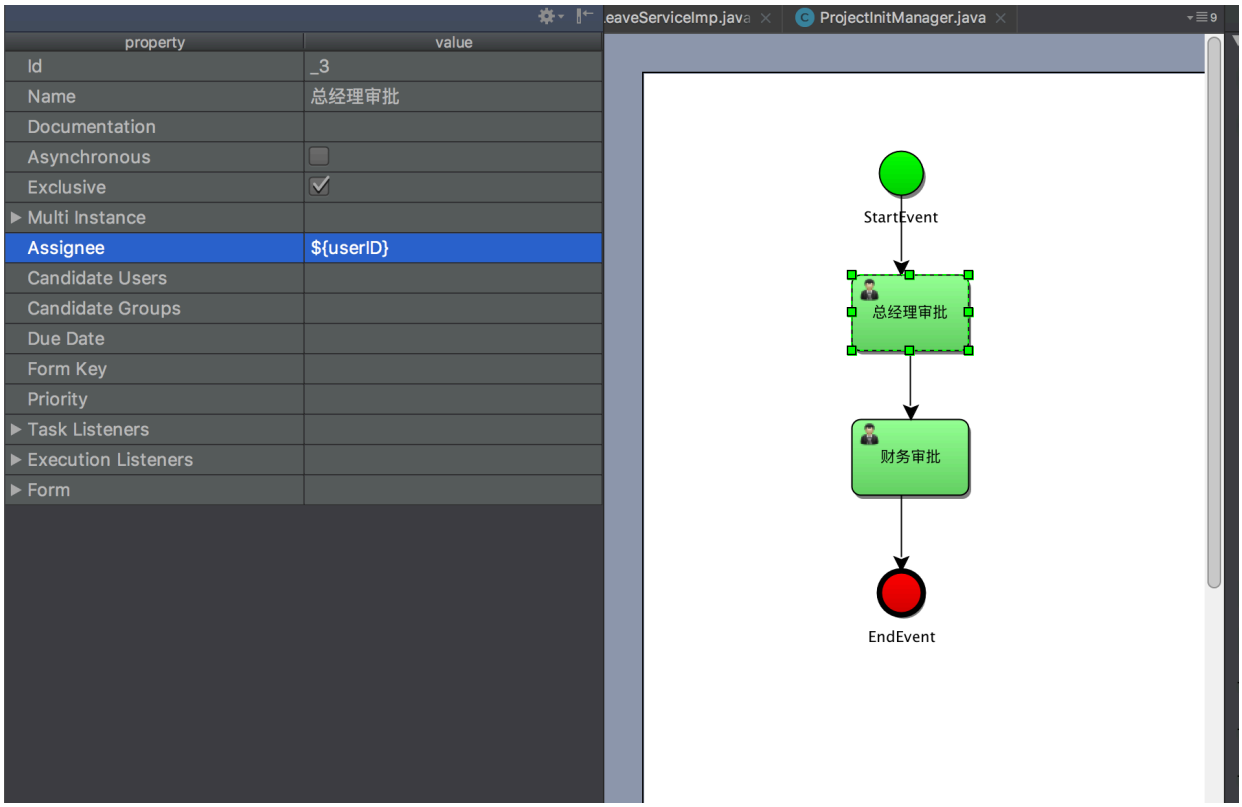
第二种：使用Java EE6规范的UEL表达式

UEL是Java EE6规范的一部分，UEL(Unified Expression Language) 即统一的表达式语言，Activiti支持两种UEL表达式：UEL-VALUE、UEL-METHOD方式。

使用步骤：

1. 在任务的节点，不直接指定处理人的ID，设置处理人的表达式为${assignee}
2. 在启动一个流程实例的时候设置启动动态的变量的assignee的值
3. 在程序执行过程中，使用UEL表达式获取指定的办理人

BPMN流程图



代码演示

```
public void startProcessInstance2(String processName) {
    RuntimeService runtimeService = manager.getRuntimeService();
    Map<String,Object> variable = new HashMap<>();
    //指定当前节点的办理人是zhangsan
    variable.put("userID","zhangsan");
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey(processName,variable);
    System.out.println("---->>"+processInstance.getId());
    System.out.println("----
>>"+processInstance.getProcessDefinitionName());
    System.out.println("---->>"+processInstance.getProcessDefinitionId());
    System.out.println("---->>"+processInstance.getBusinessKey());
}
```

使用POJO类

启动的流程时候的参数的value，不仅仅可以是String类型，还可以是Object对象（序列化的），Map,List,Array

我们这里采用对象做演示，执行步骤如下

1、设置流程的第一个节点动态的值为${user.userId}，他会默认查找变量name为user的对应的值的getUserId获取值，重新部署流程定义

2、启动流程时，设置这个user的javabean到流程的全局变量中

3、查看走到这个节点的当前任务的处理人是否是我们user的userId变量的值

4、设置节点UEL表达式：${user.userID}

代码演示

```
package com.sudojava.activiti.domain;

public class User {
    private String userID;
    private String name;
    private String pswd;

    public User() {
    }

    public User(String userID, String name, String pswd) {
        this.userID = userID;
        this.name = name;
        this.pswd = pswd;
    }

    public String getUserID() {
        return userID;
    }

    public void setUserID(String userID) {
        this.userID = userID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPswd() {
        return pswd;
    }

    public void setPswd(String pswd) {
        this.pswd = pswd;
    }

    @Override
    public String toString() {
        return "User{" +
                "userID='" + userID + '\'' +
                ", name='" + name + '\'' +
                ", pswd='" + pswd + '\'' +
```

```
                '}';
    }
}
```

启动流程

```
public void startProcessInstance2(String processName) {
    RuntimeService runtimeService = manager.getRuntimeService();
    Map<String,Object> variable = new HashMap<>();
    User user = new User();
    user.setName("admin");
    user.setPswd("admin");
    user.setUserID("zhangsan");
    //指定当前节点的办理人是zhangsan
    variable.put("user",user);
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey(processName,variable);
    System.out.println("---->>"+processInstance.getId());
    System.out.println("----
>>"+processInstance.getProcessDefinitionName());
    System.out.println("---->>"+processInstance.getProcessDefinitionId());
    System.out.println("---->>"+processInstance.getBusinessKey());
}
```

使用UEL-METHOD

执行步骤

1、设置节点的执行人为${method.getUserNameByUserId(userId)}，其中method方法是我们注入到spring中的一个类，userId是我们设置的全局变量

2、将method方法注入到activiti的processEngineConfiguration的bean中（在我们的activiti.cfg.xml中）

3、启动一个流程设置全局变量userId作为启动参数,看看是否走到这个节点的处理人是我们method方法getUserNameByUserId返回的name

代码演示

```
public class CommandMethod{

    public String getUserIDById(int userId){
        //查询数据库表中的User表
        return "activiti"+userId;
    }
}
```

activiti-cfg.xml配置

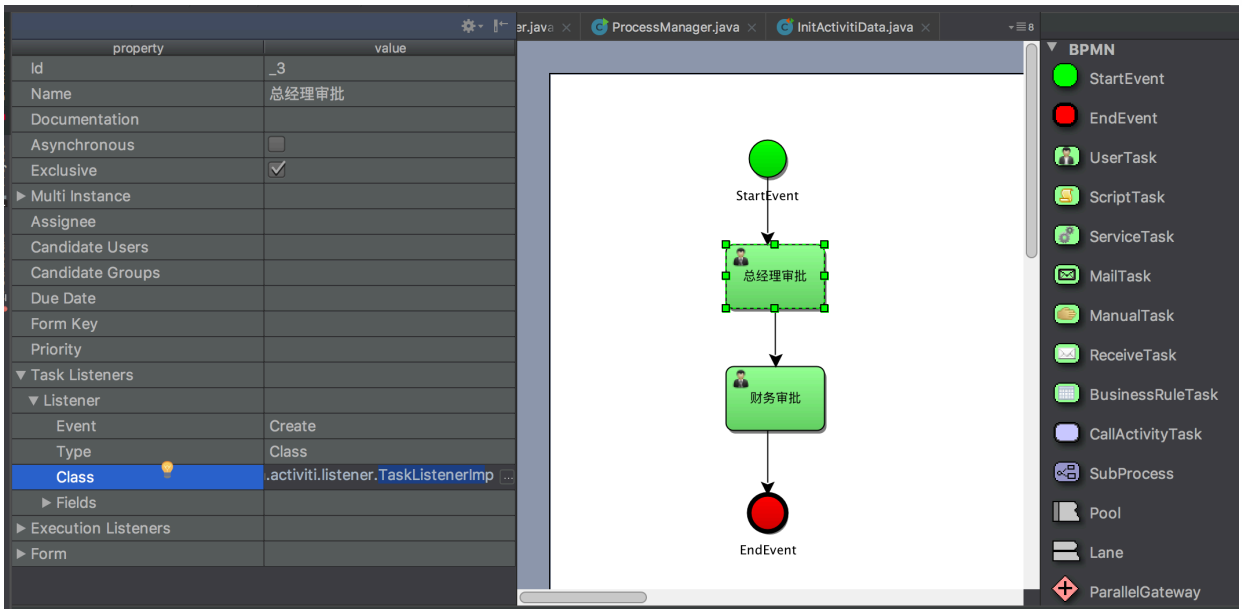
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="maxActive" value="3"/>
        <property name="maxIdle" value="1"/>
        <property name="username" value="root"/>
        <property name="password" value="root"/>
        <property name="url" value="jdbc:mysql://localhost:3306/activiti2?
useUnicode=true&amp;characterEncoding=UTF-8"/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    </bean>
    <bean id="commandMethod"
class="com.sudojava.activiti.bean.CommandMethod"/>
    <bean id="processEngineConfiguration"
          class="org.activiti.spring.SpringProcessEngineConfiguration">
        <property name="dataSource" ref="dataSource"/>
        <property name="transactionManager" ref="transactionManager"/>
        <property name="databaseSchemaUpdate" value="true"/>
        <property name="beans">
            <map>
                <entry key="commandMethod" value="commandMethod"></entry>
            </map>
        </property>
    </bean>
    <!-- (事务管理)transaction manager, use JtaTransactionManager for global
tx -->
    <bean id="transactionManager"

 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
</beans>
```

第三种：使用监听器TaskListenerImpl 来设置任务办理人

BPMN图



核心代码

```
package com.sudojava.activiti.listener;

import org.activiti.engine.delegate.DelegateTask;
import org.activiti.engine.delegate.TaskListener;

public class TaskListenerImp implements TaskListener {
    //设置待办事宜的处理人
    @Override
    public void notify(DelegateTask delegateTask) {
        delegateTask.setAssignee("zhangsan");
    }
}
```

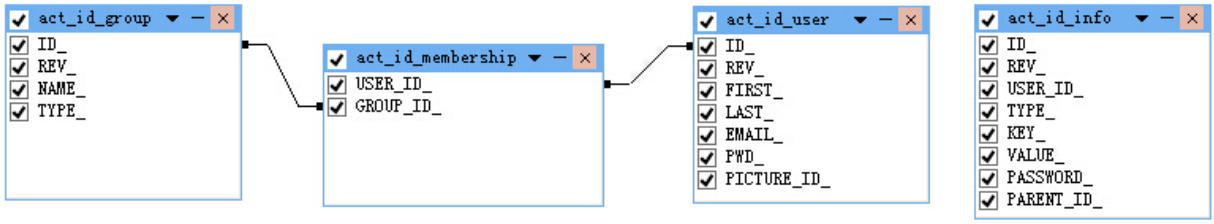
```
public void findPersonalTask() {
    String userID = "zhangsan";

 manager.getTaskService().createTaskQuery().taskAssignee(userID).list().for
Each(new Consumer<Task>() {
        @Override
        public void accept(Task task) {
            System.out.println("id=" + task.getId());
            System.out.println("name=" + task.getName());
            System.out.println("assinee=" + task.getAssignee());
            System.out.println("createTime=" + task.getCreateTime());
            System.out.println("executionId=" + task.getExecutionId());
        }
    });
}
```

第九章 Activiti用户与用户组的关系

在Activiti中默认建立了用户表，通过用户表可以构建简单的的用户管理功能。下面我们来看看Activiti给我们提供的四张关于用户和组的表。

9.1 表结构



1、从表的名字可以看出来，activiti其实提供了一个简单的用户表结构，用户组与用户之间通过映射表进行关联，用户信息是一个单独的表。

2、如果要扩展成权限表，只需要增加一个权限表，然后和用户表进行映射，或者和用户组进行映射，即可成为一个简单的权限管理结构表。

3、用户体系表和activiti的其他表没有外键关联，说明acitivi允许你使用自己创建的用户体系表，这样可以和spring security或者shiro容易的结合在一起。

9.2 操作用户组和用户表

1、创建用户组

```
    private ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();

    @Test
    public void addGroup() {
        IdentityService identityService = engine.getIdentityService();
        Group groupEntity = identityService.newGroup("1001");
        groupEntity.setName("超级管理员");
        groupEntity.setType("administrator");
        identityService.saveGroup(groupEntity);
    }
```

2、保存删除用户组

```
identityService.saveGroup(group);
identityService.deleteGroup("1");
```

3、查询用户组

```
//调用listPage方法，从索引为2的记录开始，查询3条记录
List<Group> datas = identityService.createGroupQuery().list();
  for (Group data : datas) {
      System.out.println(data.getId() + "---" + data.getName() + " ");}
//其他类似查询方法
identityService.createGroupQuery().listPage();//分页返回查询结果
identityService.createGroupQuery().count();//结果总数
identityService.createGroupQuery().groupName("managerA").singleResult();//查
询到多个时会抛出异常
identityService.createGroupQuery().groupNameLike("%managerA%").singleResult
();//
identityService.createGroupQuery().groupId("1").singleResult();//
identityService.createGroupQuery().groupType("typeA").singleResult();//
identityService.createGroupQuery().orderByGroupId().asc().list();//结果升序
identityService.createGroupQuery().orderByGroupName().desc().list();//结果降
序
```

4、添加、保存、删除用户

```
@Test
   public void addUser() {
       IdentityService identityService = engine.getIdentityService();

       User user = identityService.newUser("10001");
       user.setEmail("luo@sina.com");
       user.setFirstName("luo");
       user.setLastName("liwen");
       user.setPassword("admin");
       identityService.saveUser(user);//保存用户
       identityService.deleteUser("10001");//删除用户

   }
```

5、验证用户登录信息

```
boolean flag =   identityService.checkPassword("admin","admin");
```

6、设置用户信息表以及相关查询

```
//设置用户info信息，设置后type字段变成userinfo
identityService.setUserInfo("INFO表USER_ID","INFO表KEY","INFO表VALUE");
//设置账号信息，设置后TYPE字段变成account,一个用户可以用多个账号
identityService.setUserAccount("INFO表USER_ID","INFO表KEY","INFO表VALUE");
//查询相关信息
identityService.getUserInfo("INFO表USER_ID","INFO表KEY");
```

9.3 用户与用户组关系

1、设置用户和用户组关系

```
identityService.createMembership("用户ID","组ID");//绑定
identityService.deleteMembership("用户ID","组ID");//删除
```

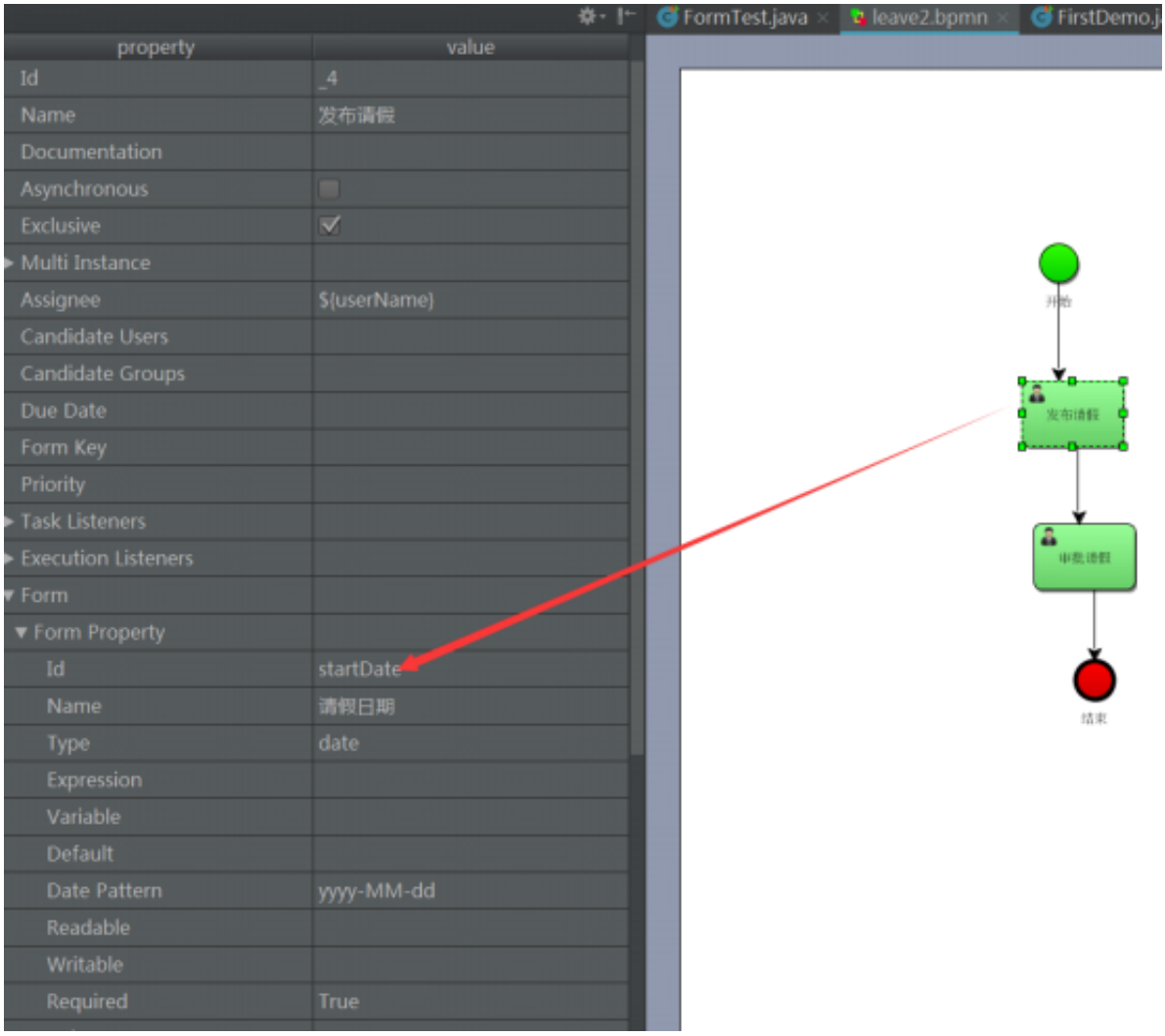
2、查询用户与用户组关系

```
List<Group> datas =
identityService.createGroupQuery().groupMember(user.getId()).list();
List<User> datas =
identityService.createUserQuery().MemberOfGroup(group.getId()).list();
```

第十章 表单

除了使用Addignee保存任务操作者，每个任务节点可以保存一些简单的信息。表单能保存的数据较少，可以自己创建表保存数据。

10.1 内置表单



```
<userTask activiti:assignee="${userName}" activiti:exclusive="true" id="_4"
name="发布请假">
  <extensionElements>
    <activiti:formProperty datePattern="yyyy-MM-dd" id="startDate" name="请
假日期" required="true" type="date"/>
    <activiti:formProperty id="reason" name="请假原因" required="true"
type="string"/>
  </extensionElements>
</userTask>
<userTask activiti:assignee="${adminName}" activiti:exclusive="true"
id="_5" name="审批请假">
  <extensionElements>
    <activiti:formProperty id="advise" name="意见" type="enum">
      <activiti:value id="true" name="同意"/>
      <activiti:value id="false" name="拒绝"/>
    </activiti:formProperty>
  </extensionElements>
</userTask>
```

操作表单：

```
ProcessDefinition definition = processEngine.getRepositoryService()

.createProcessDefinitionQuery().processDefinitionKey("leave_mybatis").lates
tVersion().singleResult();
FormService formService = processEngine.getFormService();
TaskService taskService = processEngine.getTaskService();
RuntimeService runtimeService = processEngine.getRuntimeService();
Map<String, String> param1 = new HashMap<>();
param1.put("userName", "user1");
//模拟当前请假条的id,保证task能查出唯一的任务。当然也可以查出一个task list批量处理
String businessKey = System.currentTimeMillis() + "";
//启动任务
ProcessInstance processInstance =
formService.submitStartFormData(definition.getId(), businessKey, param1);
//查询任务
Task task =
taskService.createTaskQuery().taskAssignee("user1").processDefinitionKey("l
eave_mybatis").processInstanceBusinessKey(businessKey).singleResult();
//查询任务中的表单信息
TaskFormData formData = formService.getTaskFormData(task.getId());
for (FormProperty property : formData.getFormProperties()) {
    System.out.println(property.getName() + "=" + property.getType());
}
Map<String, String> variable = new HashMap<>();
variable.put("reason", "有事");
```

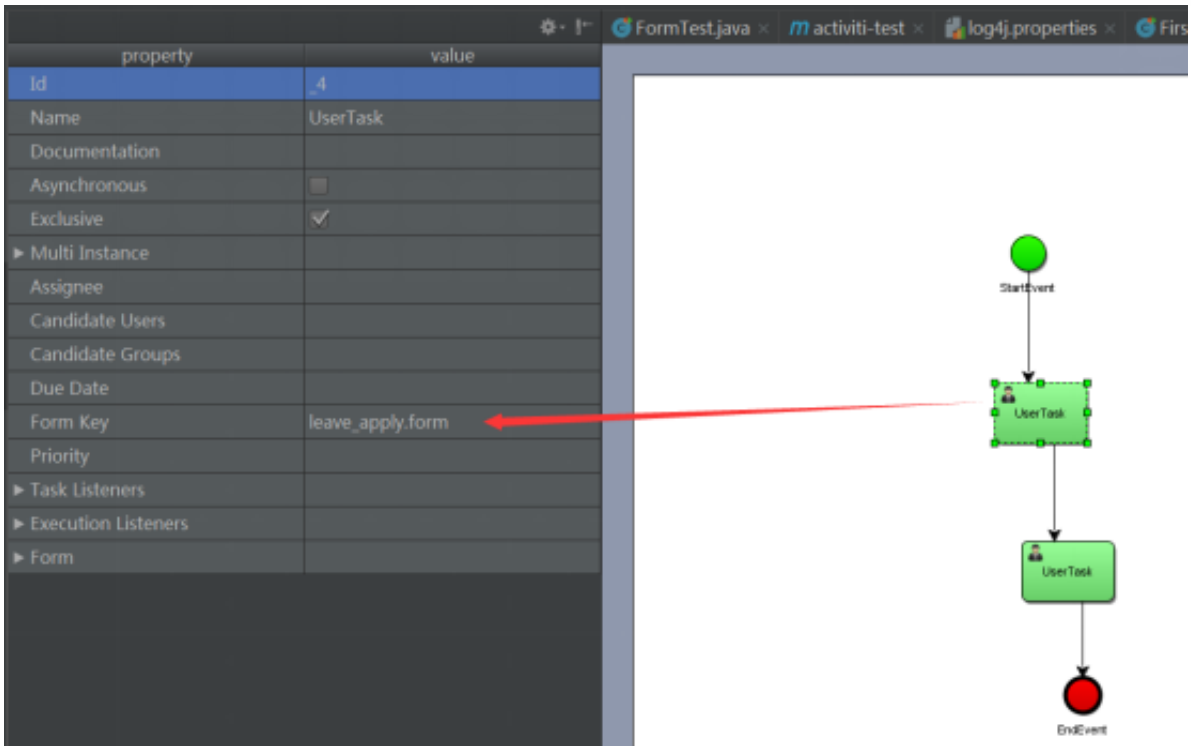
```
variable.put("startDate", "2018-01-01");
variable.put("adminName", "admin1");
//提交表单(完成当前节点的任务)
formService.submitTaskFormData(task.getId(), variable);

//查询出下一节点的任务
Task task2 =
taskService.createTaskQuery().taskAssignee("admin1").processDefinitionKey("
leave_mybatis").processInstanceBusinessKey(businessKey).singleResult();
//查询出任务中的参数
Map<String, Object> variables =
runtimeService.getVariables(processInstance.getId());
for (String key : variables.keySet()) {
    System.out.println(key + "=" + variables.get(key));
}
//完成任务
Map<String, String> variable2 = new HashMap<>();
variable2.put("advise", "true");
formService.submitTaskFormData(task2.getId(), variable2);
```

10.2 外置表单

外置表单可以自定义一个代码片段，如html片段，另存为leave_apply.form

```
<div class="control-group">
    <label class="control-label" for="startDate">开始时间：</label>
    <div class="controls">
        <input type="text" id="startDate" name="startDate"
class="datepicker" data-date-format="yyyy-mm-dd" required />
    </div>
</div>
<div class="control-group">
   <label class="control-label" for="reason">请假原因：</label>
   <div class="controls">
       <textarea id="reason" name="reason" required></textarea>
   </div>
</div>
```




```
Map<String, String> param1 = new HashMap<>();
param1.put("userName", "user1");
//模拟当前请假条的id,保证task能查出唯一的任务。当然也可以查出一个task list批量处理
String businessKey = System.currentTimeMillis() + "";
//启动任务
ProcessInstance processInstance =
formService.submitStartFormData(definition.getId(), businessKey, param1);
//查询任务
Task task =
taskService.createTaskQuery().taskAssignee("user1").processDefinitionKey("m
yProcess_1").processInstanceBusinessKey(businessKey).singleResult();
//查询出html片段，可以显示在页面上
Object renderedStartForm = formService.getRenderedTaskForm(task.getId());
System.out.println(renderedStartForm);
```

第十一章 BPMN规范

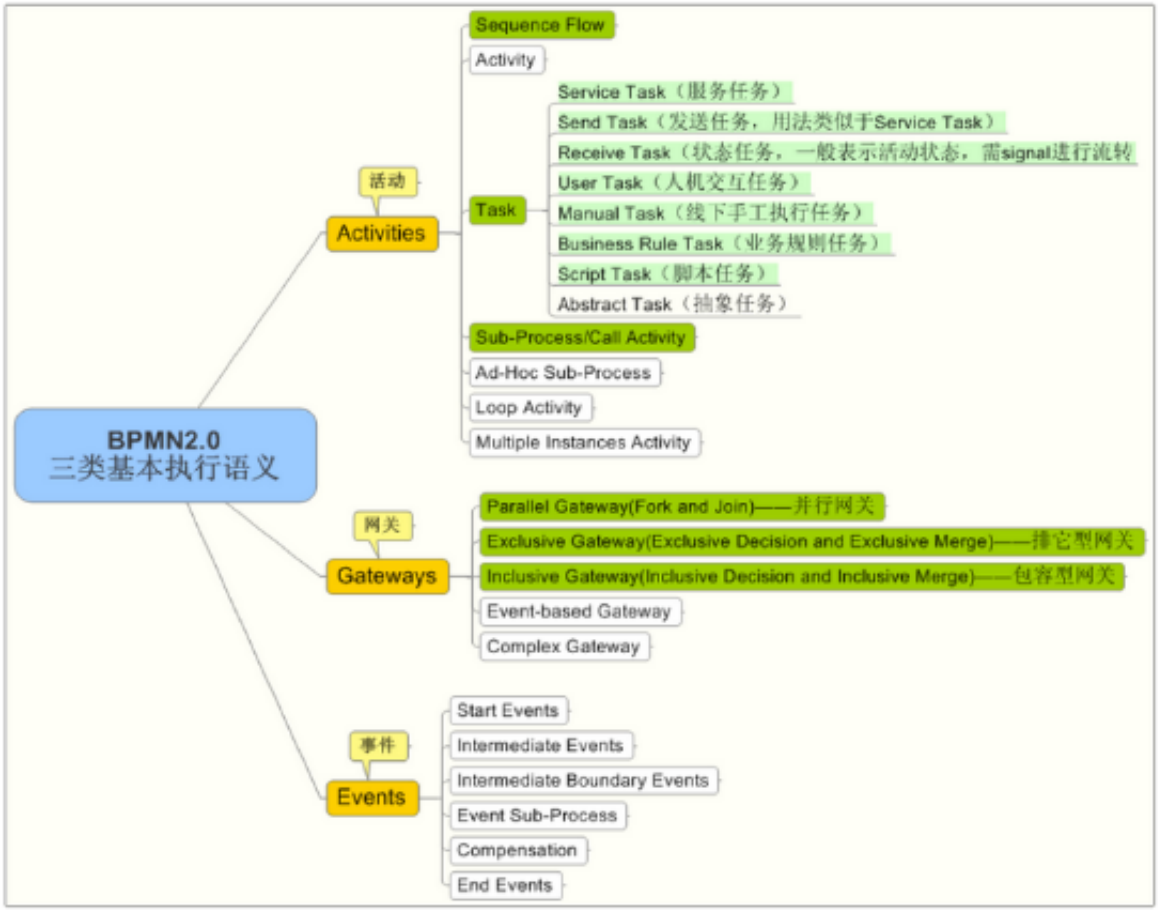
业务流程模型注解（Business Process Modeling Notation - BPMN）是业务流程模型的一种标准图形注解。这个标准 是由对象管理组（Object Management Group - OMG）维护的。包含以下部分：

流对象：事件、活动、网关

连接对象：序列流、消息流、关联

泳道：池、道

人工制品：数据对象、组、注释



总结

作业

面试题

1. 工作流的基本理论
2. Activiti简介及BPMN
3. 使用IDEA搭建Activiti开发环境
4. Activiti的整体结构