1. 工作流基础

1.1. 工作流相关概念

工作流（Workflow），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预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"。

工作流管理系统（WfMS，Workflow Management System）的主要功能是通过计算机技术的支持去定义、执行和管理工作流，协调工作流执行过程中工作之间以及群体成员之间的信息交互。工作流需要依靠工作流管理系统来实现。工作流管理系统是定义、创建、执行工作流的系统，应能提供以下三个方面的功能支持：

1. 定义工作流：包括具体的活动、规则等
2. 运行控制功能：在运行环境中管理工作流过程，对工作流过程中的活动进行调度
3. 运行交互功能：指在工作流运行中，WfMS 与用户（活动的参与者）及外部应用程序工具交互的功能。

一、定义工作流

二、执行工作流

采用工作流管理系统的优点

1. 提高系统的柔性，适应业务流程的变化
2. 实现更好的业务过程控制，提高顾客服务质量
3. 降低系统开发和维护成本

工作流框架有：Jbpm、OSWorkflow、ActiveBPEL、YAWL 等

OA（办公自动化）主要技术之一就是工作流。

1.2. 开源工作流 jBPM4.4 介绍

jBPM 即 java Business Process Management，是基于 java 的业务流程管理系统。jBPM 是市面上相当流行的一款开源工作流引擎，引擎底层基于 Active Diagram

模型。jBPM4.4 使用了 hibernate（3.3.1 版），因此可以很好的支持主流数据库。jBPM4.4 共有 18 张表。

jBPM 官方主页：http://www.jboss.org/jbpm

2. 准备 jBPM4.4 环境

2.1. jBPM4.4 所需环境

jBPM requires a JDK (standard java) version 5 or higher.

http://java.sun.com/javase/downloads/index.jsp

To execute the ant scripts, you'll need apache ant version **1.7.0** or higher: http://ant.apache.org/bindownload.cgi

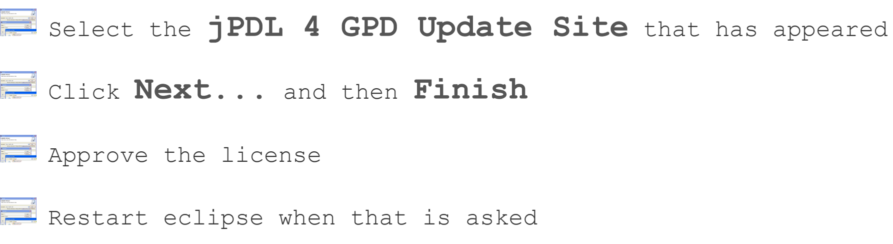
2.2. 下载相关资源

1，jBPM 下载地址：http://sourceforge.net/projects/jbpm/files/
2，Eclipse 下载地址（ Eclipse IDE for Java EE Developers (163 MB)，Version: 3.5)：
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo

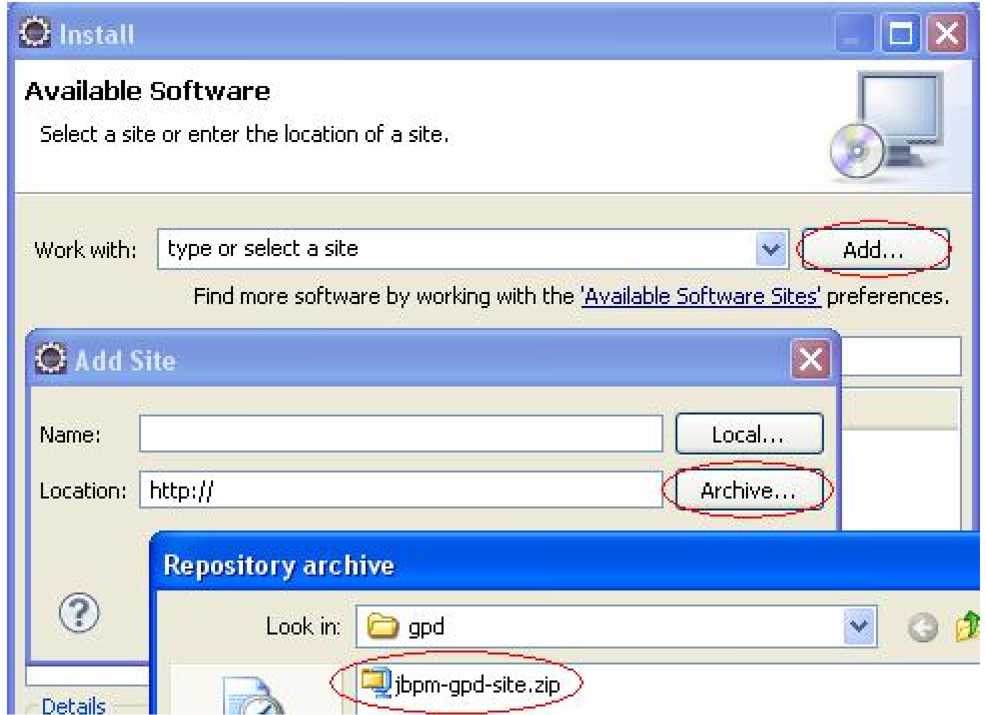
2.3. 安装流程设计器（GPD，Eclipse 插件）

GPD（Graphical Process Designer）是一个 Eclipse 插件。
安装方法说明（jBPM4.4User Guide，2.11.2. Install the GPD plugin into eclipse）：

- **Help** --> **Install New Software...**
- Click **Add...**
- In dialog **Add Site** dialog, click **Archive...**
- Navigate to **install/src/gpd/jbpm-gpd-site.zip** and click 'Open'
- Clicking **OK** in the **Add Site** dialog will bring you back to the dialog 'Install'



- Select the **jPDL 4 GPD Update Site** that has appeared
- Click **Next...** and then **Finish**
- Approve the license
- Restart eclipse when that is asked

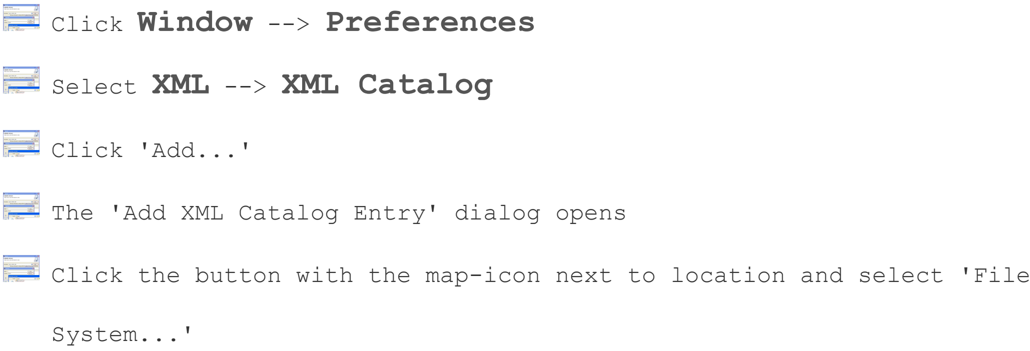


查看是否成功安装了插件：Window→Preference 中是否有 Jboss jBPM 项。

2.4. 在 Eclipse 中添加 jPDL4.4 Schema 校验

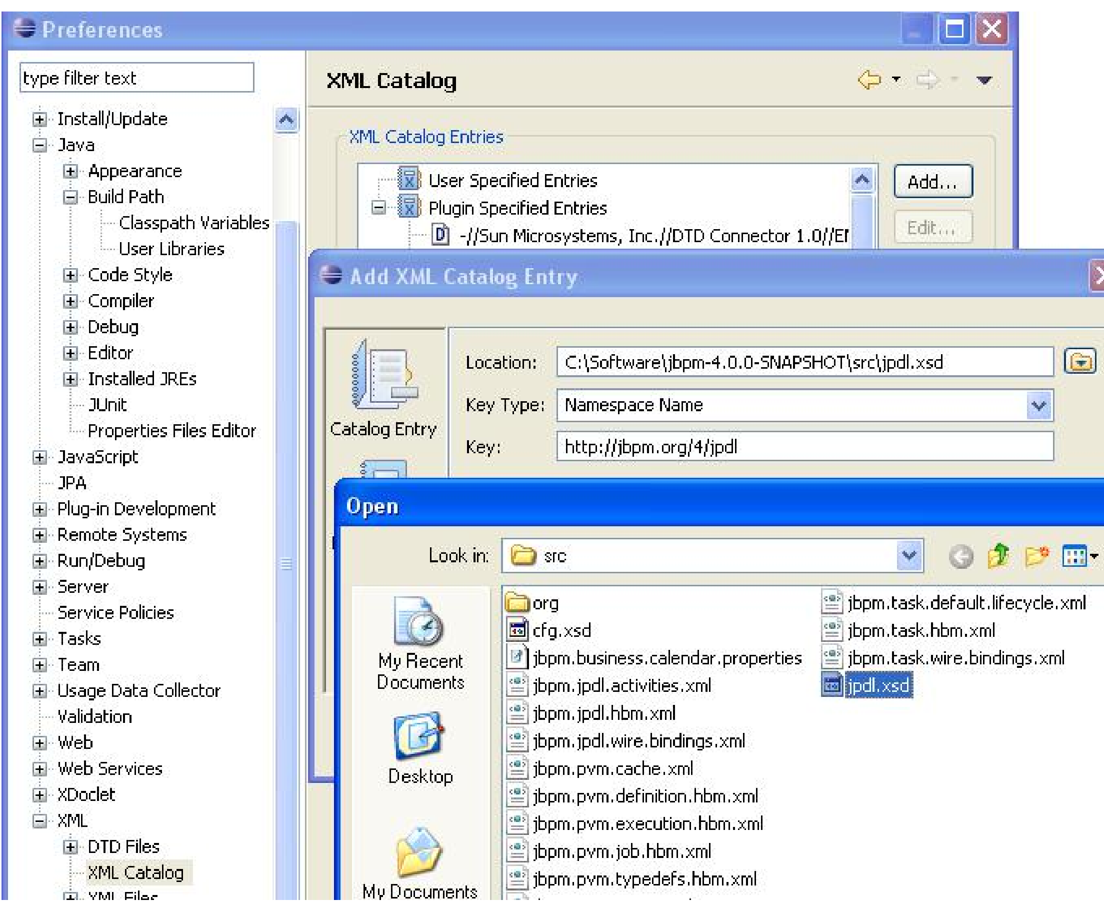
流程定义文件的 xsd 文件的路径为：JBPM_HOME/src/jpdl-4.4.xsd。

添加到 Eclipse 中的方法为(jBPM4.4 User Guide, 2.11.5. Adding jPDL 4 schema to the catalog)：



- Click **Window --> Preferences**
- Select **XML --> XML Catalog**
- Click 'Add...'
- The 'Add XML Catalog Entry' dialog opens
- Click the button with the map-icon next to location and select 'File System...'

- In the dialog that opens, select file **jpdl-4.4.xsd** in the src directory of the jBPM installation root.
- Click 'Open' and close all the dialogs



2.5. 准备 jBPM4.4 的开发环境

2.5.1. 添加 jBPM4.4 的 jar 包

1. ${JBPM_HOME}/jbpm.jar（核心包）
2. JBPM_HOME/lib/*.jar，不添加以下 jar 包：servlet-api.jar，junit.jar。其中 junit.jar 一定不要添加，因为是 3.8.2 版本，与我们使用的 junit4 有冲突。
3. 所使用的数据库对应的驱动的 jar 包(第 2 步所添加的 jar 包中已包含 mysql 的 jdbc 驱动 jar 包)。

2.5.2. 添加并定制配置文件

1. 配置文件可以从 JBPM_HOME/examples/src/中拷贝：

jbpm.cfg.xml、
logging.properties、
jbpm.hibernate.cfg.xml。

2. 修改 logging.properties 中的日志输出级别为 WARNING:
   java.util.logging.ConsoleHandler.level=WARNING
3. 修改 jbpm.hibernate.cfg.xml 中的数据库连接信息。如果使用 MySql，使用的方言一定要是 org.hibernate.dialect.**MySQL5InnoDBDialect**。
4. 数据库连接编码一定要是 UTF-8。否则可能会在部署含有中文字符的流程定义时会抛异常，说 sql 语法错误。

说明：如果要改变 jbpm.hibernate.cfg.xml 的文件名称，需要做：

1、从 JBPM_HOME/src/中拷贝 jbpm.tx.hibernate.cfg.xml 放到工程的 src/下，然后进行修改。

2、修改 jbpm.tx.hibernate.cfg.xml 中的 hibernate 主配置文件的路径配置(指定的是相对于 classpath 的相对路径)。

2.5.3. 初始化数据库

1，方法一：执行 sql 脚本文件
   ${JBPM4.4_HOME}/install/src/db/create/jbpm.*.create.sql
2，方法二：使用 Hibernate 的自动建表，在 jbpm.hibernate.cfg.xml 中配置：
   hibernate.hbm2ddl.auto=update。

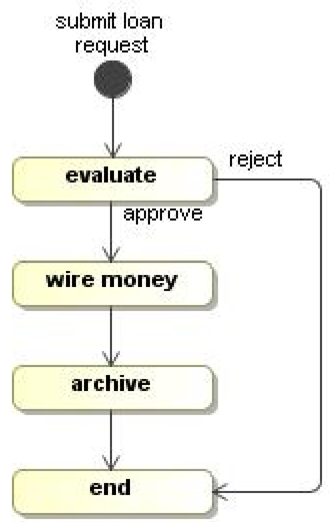
3. 核心概念与相关 API（Service API）

3.1. 概 念 ： Process definition, process instance , execution

3.1.1. Process definition

ProcessDefinition，流程定义：

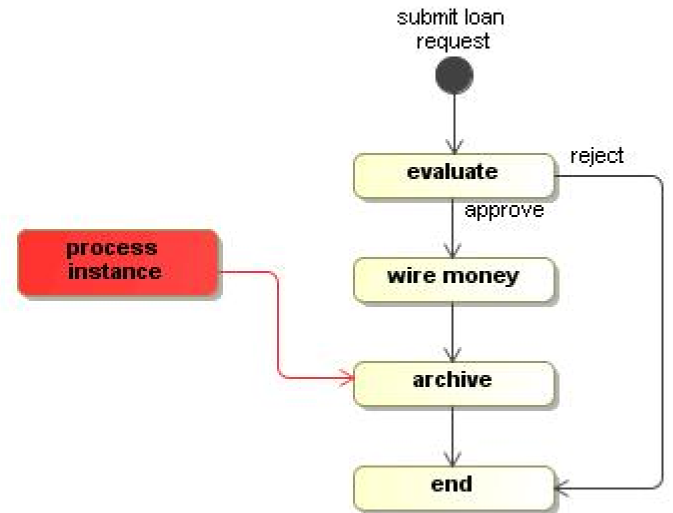
一个流程的步骤说明。如一个请假流程、报销流程、借款流程等，是一个规则。
例：



3.1.2. Process instance

ProcessInstance，流程实例：

代表流程定义的一次执行。如张三昨天按请假流程请了一次假。一个流程实例包括了所有运行阶段， 其中最典型的属性就是跟踪当前节点的指针。



3.1.3. Execution

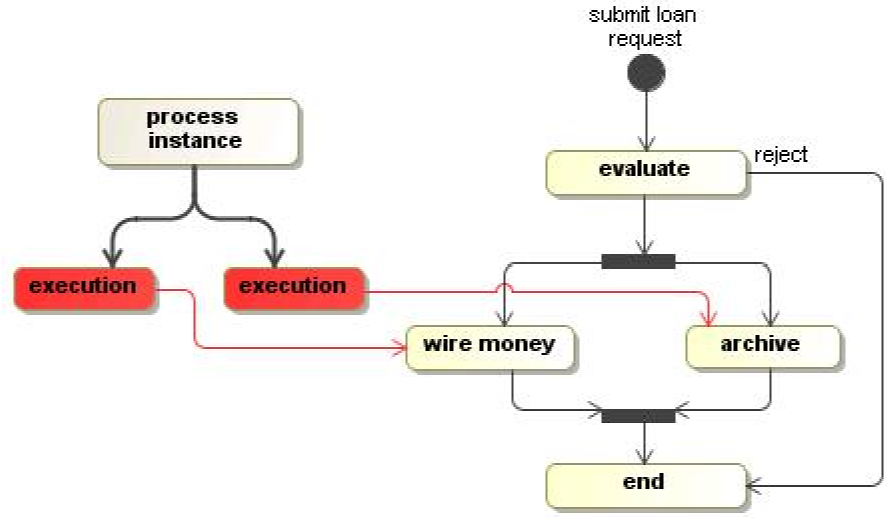
Execution，执行：

一般情况下，一个流程实例是一个执行树的根节点。

使用树状结构的原因在于， 这一概念只有一条执行路径， 使用起来更简单。 业务 API 不需要了解流程实例和执行之间功能的区别。 因此， API 里只有一个执行类型来引用流程

实例和执行。

假设汇款和存档可以同时执行，那么主流程实例就包含了 2 个用来跟踪状态的子节点：



4.1. ProcessEngine 与 Service API

4.1.1. Configuration 与 ProcessEngine

Interacting with jBPM occurs through services. The service interfaces can be obtained from the ProcessEngine which is build from a Configuration. A ProcessEngine is thread safe and can be stored in a static member field.

使用默认的配置文件（jbpm.cfg.xml）生成 Configuration 并构建 ProcessEngine：

```
ProcessEngine processEngine = new Configuration()
        .buildProcessEngine();
```

或是使用如下代码获取使用默认配置文件的、单例的 ProcessEngine 对象：

```
ProcessEngine processEngine = Configuration.getProcessEngine();
```

或是使用指定的配置文件（要放到 classPath 下）：

```
ProcessEngine processEngine = new Configuration()
      .setResource("my-own-configuration-file.xml")
```

```
        .buildProcessEngine();
```

4.1.2. jBPM Service API

jBPM 所有的操作都是通过 Service 完成的，以下是获取 Service 的方式：

```
RepositoryService repositoryService = processEngine
        .getRepositoryService();
ExecutionService executionService = processEngine
        .getExecutionService();
TaskService taskService = processEngine
        .getTaskService();
HistoryService historyService = processEngine
        .getHistoryService();
ManagementService managementService = processEngine
        .getManagementService();
```

各个 **Service** 的作用：

RepositoryService	**管理流程定义**
ExecutionService	**管理执行的，包括启动、推进、删除 Execution 等操作**
TaskService	**管理任务的**
HistoryService	历史管理（执行完的数据管理，主要是查询）
IdentityService	jBPM 的用户、组管理
ManagementService	

4.1.3. API 风格

方法调用链

每一个方法都是流程有关的一个业务操作，默认是一个独立的事务。

4.1.4. 查询的有关 API（风格）

功能说明	相应的查询 API
查询“流程定义”	`ProcessDefinitionQuery processDefinitionQuery = processEngine.getRepositoryService() .createProcessDefinitionQuery();`
查询“执行对象”	`ProcessInstanceQuery processInstanceQuery =`

（流程实例）	processEngine.getExecutionService() // .createProcessInstanceQuery();
查询“任务”	**TaskQuery** taskQuery = // processEngine.getTaskService()// .createTaskQuery();
查询“执行历史”（流程实例历史）	**HistoryProcessInstanceQuery** historyProcessInstanceQuery = processEngine.getHistoryService() .createHistoryProcessInstanceQuery();
查询“任务历史”	**HistoryTaskQuery** historyTaskQuery = processEngine.getHistoryService() .createHistoryTaskQuery();

以上列出的 Query 对象有：

1. ProcessDefinitionQuery
2. ProcessInstanceQuery
3. TaskQuery
4. HistoryProcessInstanceQuery
5. HistoryTaskQuery

这些 Query 对象的使用方法都是一致的，如下所示：

1，添加过滤条件：调用其中的有关方法指定条件即可。如：

a) processDefinitionQuery.processDefinitionKey("请假")是指定查询 key 为"请假"的流程定义；
b) taskQuery.assignee("张三")是指定办理人为"张三"的任务。

2，添加排序条件：

a) 调用 xxQuery.orderAsc(property)，表示按某属性升序排列
b) 调用 xxQuery.orderDesc(property)，表示按某属性降序排列
c) 可指定多个排序条件，就是代表第 1 顺序，第 2 顺序…等。
d) 属性名在各自的 Query 对象（接口）中有常量定义，如：
   i. ProcessDefinitionQuery.PROPERTY_ID
   ii. ProcessDefinitionQuery.PROPERTY_KEY
   iii. TaskQuery.PROPERTY_NAME
   iv. TaskQuery.PROPERTY_ASSIGNEE

3，指定分页有关信息：

a) 调用方法 xxQuery.page(firstResult, maxResults);
b) 这是指定 first 与 max 的值（就是 Hibernate 中的 Query.setFirstResult() 与 Query.setMaxResults()）
c) 如果没有调用这个方法，代表要查询出符合条件的所有记录。

4，查询得到结果：

a) 调用方法 xxQuery.list(); 表示查询列表
b) 调用方法 xxQuery.uniqueResult(); 表示查询唯一的结果
c) 调用方法 xxQuery.count(); 表示查询符合条件的记录数量

5. 管理流程定义

没有更新功能

5.1. 部署流程定义

注意区分 Deployment 与 ProcessDefinition

5.1.1. 示例代码 1：流程定义有关文件在 classpath 中

```
String deploymentId = processEngine.getRepositoryService()
        .createDeployment()
        .addResourceFromClasspath("process/test.jpdl.xml")
        .addResourceFromClasspath("process/test.png")
        .deploy();
```

5.1.2. 示例代码 2：一次添加多个流程定义有关文件（要先打成 zip 包）

```
String deploymentId = processEngine.getRepositoryService()
        .createDeployment()
        .addResourcesFromZipInputStream(zipInputStream)
        .deploy();
```

5.1.3. 说明

1， .addResourceFromClasspath(resource); 可以调用多次以添加多个文件。文件重复添加也不会报错。
2， .addResourceFromInputStream(resourceName, inputStream)添加一个文件（使用 InputStream）
3， .addResourcesFromZipInputStream(zipInputStream)添加多个文件，里面也可以有文件夹。
4， 以上方法可以在一起调用。

5.2. 删除流程定义

5.2.1. 示例代码 1：删除流程定义，如果有关联的流程实例信息则报错

```
repositoryService.deleteDeployment(deploymentId);
```

5.2.2. 示例代码 2：删除流程定义，并删除关联的流程实例与历史信息

```
repositoryService.deleteDeploymentCascade(deploymentId);
```

5.3. 查询流程定义

5.3.1. 相关查询 API 说明：ProcessDefinitionQuery

```
RepositoryService.createProcessDefinitionQuery()
```

5.3.2. 示例代码 1：查询所有流程定义

```
// 1，构建查询
ProcessDefinitionQuery pdQuery = processEngine.getRepositoryService()
        .createProcessDefinitionQuery()//
        .orderAsc(ProcessDefinitionQuery.PROPERTY_NAME)//
        .orderDesc(ProcessDefinitionQuery.PROPERTY_VERSION);

// 2，查询出总数量与数据列表
long count = pdQuery.count();
List<ProcessDefinition> list = pdQuery.page(0, 100).list();// 分页：取
出前100条记录

// 3，显示结果
System.out.println(count);
for (ProcessDefinition pd : list) {
    System.out.println("id=" + pd.getId()//
            + ",deploymentId=" + pd.getDeploymentId()//
            + ",name=" + pd.getName()//
```

```
            + ",version=" + pd.getVersion()//
            + ",key=" + pd.getKey()); //
}
```

5.3.3. 示例代码 2：查询所有最新版本的流程定义列表

```
// 1，查询，按version升序排序，则最大版本排在最后面
List<ProcessDefinition> all = processEngine.getRepositoryService()//
        .createProcessDefinitionQuery()//
        .orderAsc(ProcessDefinitionQuery.PROPERTY_VERSION)
        .list();
// 2，过滤出所有不同Key的最新版本（因为最大版本在最后面）
Map<String, ProcessDefinition> map = new HashMap<String,
ProcessDefinition>(); // map的key是流程定义的key，map的vlaue是流程定义对象
for (ProcessDefinition pd : all) {
    map.put(pd.getKey(), pd);
}
Collection<ProcessDefinition> result = map.values();
// 3，显示结果
for (ProcessDefinition pd : result) {
    System.out.println("deploymentId=" + pd.getDeploymentId()//
            + ",\t id=" + pd.getId()// 流程定义的id，格式：{key}-{version}
            + ",\t name=" + pd.getName()
            + ",\t key=" + pd.getKey()
            + ",\t version=" + pd.getVersion());
}
```

5.4. 获取部署对象中的文件资源内容

```
// 资源的名称，就是 jbpm4_lob 表中的 NAME_ 列表值
String deploymentId = "90001";
String resourceName = "test.png";
InputStream in = processEngine.getRepositoryService()
    .getResourceAsStream(deploymentId, resourceName);
```

5.5. 获取流程图中某活动的坐标

```
String processDefinitionId = "test-1"; // 流程定义的id
String activityName = "start1"; // 活动的名称

ActivityCoordinates c = processEngine.getRepositoryService()
            .getActivityCoordinates(processDefinitionId, activityName);
```

```
System.out.println("x=" + c.getX()
    + ",y=" + c.getY()
    + ",width=" + c.getWidth()
    + ",height=" + c.getHeight());
```

6. 执行流程实例

6.1. 启动流程实例

说明：流程实例创建后，直接就到开始活动后的第一个活动，不会在开始活动停留。

6.1.1. 示例代码 1：使用指定 key 的最新版本的流程定义启动流程实例

```
ProcessInstance pi = processEngine.getExecutionService()
          .startProcessInstanceByKey(processDefinitionKey);
```

6.1.2. 示例代码 2：使用指定 key 的最新版本的流程定义启动流程实例，并设置一些流程变量

```
// 准备流程变量
Map<String, Object> variables = new HashMap<String, Object>();
variables.put("申请人", "张三");
variables.put("报销金额", 1000.00);

// 启动流程实例，并设置一些流程变量
ProcessInstance pi = processEngine.getExecutionService()
       .startProcessInstanceByKey(processDefinitionKey, variables);
```

6.2. 向后执行一步（Signal）

6.2.1. 示例代码 1：向后执行一步，使用唯一的 outcome 离开活动

```
processEngine.getExecutionService().signalExecutionById(executionId);
```

6.2.2. 示例代码 2：向后执行一步，使用唯一的 outcome 离开活动，并设置一些流程变量

```
Map<String, Object> variables = new HashMap<String, Object>();
variables.put("审批结果", "同意");

processEngine.getExecutionService()
    .signalExecutionById(executionId, variables);
```

6.2.3. 示例代码 3：向后执行一步，使用指定的 outcome 离开活动

```
String outcome= "to end1";
processEngine.getExecutionService()
    .signalExecutionById(executionId, outcome);
```

6.2.4. 示例代码 4：向后执行一步，使用指定的 outcome 离开活动，并设置一些流程变量

```
String outcome= "to end1";
Map<String, Object> variables = new HashMap<String, Object>();
variables.put("审批结果", "同意");

processEngine.getExecutionService()
    .signalExecutionById(executionId, outcome, variables);
```

6.3. 查询任务

6.3.1. 查询个人任务列表

```
方式1：TaskService.findPersonalTasks(userId);

方式2：List<Task> list = taskService.createTaskQuery()
              .assignee(userId)
              .list();
```

```
// 显示任务信息
for (Task task : taskList) {
    System.out.println("id=" + task.getId()// 任务的id
        + ",name=" + task.getName()// 任务的名称
        + ",assignee=" + task.getAssignee()// 任务的办理人
        + ",createTime=" + task.getCreateTime() // 任务的创建（生成）的时间
        + ",executionId=" + task.getExecutionId());// 任务所属流程实例的id
}
```

6.3.2. 查询组任务列表

方式1：

```
taskService.findGroupTasks(userId);
```

方式2：

```
List<Task> list = processEngine.getTaskService()//
        .createTaskQuery()//
        .candidate(userId)//
        .list();
```

6.4. 完成任务

6.4.1. 正常完成任务（也可以同时设置一些流程变量）

```
String taskId = "420001";
processEngine.getTaskService().completeTask(taskId);

processEngine.getTaskService().completeTask(taskId, outcome);

processEngine.getTaskService().completeTask(taskId, outcome,
variables);
```

6.4.2. 自行控制任务完成后是否可向后流转

```
String taskId = "420001";

// 1，设置为false代表：办理完任务后不向后移动（默认为true）
TaskImpl taskImpl = (TaskImpl) processEngine
    .getTaskService().getTask(taskId);
taskImpl.setSignalling(false);

// 2，办理完任务
processEngine.getTaskService().completeTask(taskId);
```

6.5. 拾取任务

1, TaskService.takeTask(taskId, userId)，拾取组任务到个人任务列表中，如果任务有 assignee，则会抛异常。
2, processEngine.getTaskService().assignTask(taskId, userId)，转交任务给其他人，（如果任务有 assignee，则执行这个方法代表重新分配。也可以把 assignee 设为 null 表示组任务没有人办理了）

6.6. 设置与获取流程变量

6.6.1. 设置流程变量

6.6.1.1. 方式 1：根据 executionId 设置或获取流程变量

```
ExecutionService.setVariable(executionId, name, value);

Object obj = executionService.getVariable(executionId, "请假人");
```

6.6.1.2. 方式 2：根据 taskId 设置或获取流程变量

```
TaskService.setVariables(taskId, variables); // 一次设置多个变量

Object obj = executionService.getVariable(executionId, "请假人");
```

6.6.1.3. 流程变量所支持的值的类型（jBPM User Guide，7.2. Variable types）

7.2. Variable types

jBPM supports following Java types as process variables:

- java.lang.String
- java.lang.Long
- java.lang.Double
- java.util.Date

- java.lang.Boolean
- java.lang.Character
- java.lang.Byte
- java.lang.Short
- java.lang.Integer
- java.lang.Float
- byte[] (byte array)
- char[] (char array)
- hibernate entity with a long id
- hibernate entity with a string id
- serializable

For persistence of these variable, the type of the variable is checked in the order of this list. The first match will determine how the variable is stored.

6.7. 直接结束流程实例（自己手工结束）

```
String processInstanceId =  "test.10001";
processEngine.getExecutionService()
    .endProcessInstance(processInstanceId, ProcessInstance.STATE_ENDED);
```

7. jBPM4.4 的流程定义语言（设计流程）

7.1. process（流程）

是.jpdl.xml 的根元素，可以指定的属性有：

属性名	作用说明
name	流程定义的名称，用于显示。
key	流程定义的 key，用于查询。 如未指定，则默认为 name 的值。
version	版本，如果指定，则不能与已有的流程定义的版本重复；如未指定，则此 key 的流程定义的第 1 个为版本 1，以后的是版本递增（每

	次加 1）

7.2. Transition（连线、转移、流转）

1， 一个活动中可以指定一个或多个 Transition（Start 中只能有一个，End 中没有）。
   a) 开始活动中只能有一个 Transition。
   b) 结束活动中没有 Transition。
   c) 其他活动中有 1 条或多条 Transition
2， 如果只有一个，则可以不指定名称（名称是 null）；如果有多个，则要分别指定唯一的名称。

7.3. 流转控制活动

7.3.1. start（开始活动）

代表流程的开始边界，一个流程有且只能有一个 Start 活动。开始活动只能指定一个 Transition。在流程实例启动后，会自动的使用这个唯一的 Transition 离开开始活动，到一下个活动。

7.3.2. end、end-error、end-cancel（结束活动）

代表流程的结束边界，可以有多个，也可以没有。如果有多个，则到达任一个结束活动，整个流程就都结束了；如果没有，则到达最后那个没有 Transition 的活动，流程就结束了。

7.3.3. state（状态活动）

功能：等待。

7.3.4. task（任务活动）

分配任务：

1， actor=#{String 型的变量}
2， AssignmentHandler，需要在 <task> 元素中写 <assignment-handler class="AssignmentHandlerImpl"/>子元素。
   a) 指定的类要实现 AssignmentHandler 接口
   b) 在其中可以使用 Assignable.setAssignee(String)，分配个人任务。

7.3.5. decision（判断活动）

1，使用 expression，如：expr="#{'to state2'}"
2，使用 Handler，要实现 DecisionHandler 接口
3，如果同时配置了 expression 与 Handler，则 expression 有效，忽略 Handler。

7.3.6. fork、join（分支／聚合活动）

这是多个分支并行（同时）执行的，并且所有的分支 Execution 都到 Join 活动后才离向后执行。

7.4. 自定义活动（Custom）

1，在<custom>元素中指定 class 属性为指定的类。
2，这个类要实现 ExternalActivityBehaviour 接口，其中有两个方法：
  1，execute(ActivityExecution)，节点的功能代码
  2，signal(ActivityExecution, String, Map)，在当前节点等待时，外部发信号时的行为
  3，在 execute()方法中，可以调用以下方法对流程进行控制
    1，ActivityExecution.waitForSignal()，在当前节点等待。
    2，ActivityExecution.takeDefaultTransition()，使用默认的 Transition 离开，当前节点中定义的第一个为默认的。
    3，ActivityExecution.take(String transitionName)，使用指定的 Transition 离开
    4，ActivityExecution.end()，结束流程实例
  4，也可以实现 ActivityBehaviour 接口，只有一个方法 execute(ActivityExecution)，这样就不能等待，否则 signal 时会有类转换异常。

7.5. 事件

1， 在根元素中，或在节点元素中，使用<on event="">元素指定事件，其中 event 属性代表事件的类型。
2， 在<on>中用子元素<event-listener class="EventListenerImpl" />，指定处理的类，要求指定的类要实现 EventListener 接口
3， 事件类型：
   a) <on>元素放在根元素（<process>）中，可以指定 event 为 start 或 end，表示流程的开始与结束。
   b) <on>元素放在节点元素中，可以指定 event 为 start 或 end，表示节点的进入与离开
   c) 在 Start 节点中只有 end 事件，在 End 节点中只有 start 事件。
   d) 在<transition>元素中直接写<event-listener class="">，就是配置事件。（因为在这里只有一个事件，所以不用写 on 与类型）

e) 在<task>元素中还可以配置 assign 事件，是在分配任务时触发的。

8. jBPM4.4 应用

8.1. 与 Spring 集成（jBPM4.4 Developers Guide, Chapter 17. Spring Integration）

8.1.1. 在 jbpm.cfg.xml 中

1，删除配置：<import resource="jbpm.tx.hibernate.cfg.xml" />
2，增加配置：<import resource="jbpm.tx.spring.cfg.xml" />

8.1.2. 在 applicationContext.xml 中配置

```
<!-- 配置 ProcessEngine（整合 jBPM4） -->
<!-- jbpmCfg 是相对于 classpath 的相对路径，默认值为 jbpm.cfg.xml -->
<bean id="springHelper"
    class="org.jbpm.pvm.internal.processengine.SpringHelper">
    <property name="jbpmCfg" value="jbpm.cfg.xml"></property>
</bean>
<bean id="processEngine" factory-bean="springHelper"
    factory-method="createProcessEngine" />
```

8.1.3. 测试

```
@Test // 测试 ProcessEngine
public void testProcessEngine() {
    ProcessEngine processEngine = (ProcessEngine)ac.getBean("processEngine");
    Assert.assertNotNull(processEngine);
}
```

8.1.4. 注意事项

如果做了 JBPM4.4 与 Spring 整合（使用了 jbpm.tx.spring.cfg.xml），则在程序中就一定要使用 Spring 注入 ProcessEngine，千万不能使用 Configuration.getProcessEngine()生成 ProcessEngine，因为这时内部的代码

有以下逻辑：如果整合了 Spring 但没有 ApplicationContext，就默认读取 applicationContext.xml 创建 ApplicationContext 实例并从中获取名为 "ProcessEngine" 的对象。而这时如果把 pe = Configuration.getProcessEngine()写成某 Spring 中管理的 bean 的初始化代码，就会有无限循环，不停的创建 ApplicationContext 了！

8.2. 自行控制事务

1，修改 jbpm.tx.hibernate.cfg.xml
   a) 不让 jBPM 自行管理事务：去掉<standard-transaction-interceptor />
   b) 让 Jbpm 使用 SessionFactory.getCurrentSession()：修改为 <hibernate-session current="true" />
2，配置可以使用 SessionFactory.getCurrentSession()，在 jbpm.hibernate.cfg.xml 中配置：<property name="hibernate.current_session_context_class">thread</property>
3，要使用同一个 SessionFactory，且都要使用 SessionFactory.getCurrentSession() 获取 Session：
   a) 同一个 SessionFactory：SessionFactory sf = processEngine.get(SessionFactory.class)
   b) 在 BaseDaoImpl 中增加：
      i. getSession() { return HibernateUtils.getSessionFactory().getCurrentSession(); }
      ii. getProcessEngine(){ return org.jbpm.api.Configuration.getProcessEngine(); }
4，统一的打开与提交或回滚事务：使用 OpenSessionInViewFilter 控制事务。

8.3. 启动 Tomcat 后，访问 JSP 时（使用的是 MyEclipse 自带的 Tomcat，是 6.0 的版本），报错： Caused by: java.lang.LinkageError: loader constraints violated when linking javax/el/ExpressionFactory class

```
at org.apache.jsp.WEB_002dINF.jsp.UserAction.loginUI_jsp._jspInit(loginUI_jsp.java:39)
at org.apache.jasper.runtime.HttpJspBase.init(HttpJspBase.java:52)
at org.apache.jasper.servlet.JspServletWrapper.getServlet(JspServletWrapper.java:159)
at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:329)
at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:320)
at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:266)
... 40 more
```

说明：原因是 Jbpm 的 juel.jar, juel-engine.jar, juel-impl.jar 包和 Tomcat6.0 中的 el-api.jar 包冲突了。

有三个解决办法：

1，方法一：在 MyEclipse 的 Preferences -> MyEclipse -> Application Servers -> Tomcat -> .. -> Paths 中配置 Append to classpath，选中 juel.jar, juel-engine.jar, juel-impl.jar 这三个 jar 包

就可以了。

2，方法二：将 juel.jar, juel-engine.jar, juel-impl.jar 这三个包复制到 tomcat6 下 lib/ 中，并删除原来的 el-api.jar，

切记还要把工程中 WEB-INF\lib 下的 juel.jar, juel-engine.jar, juel-impl.jar 删除，不然还是要冲突。

3，方法三：换成 tomcat5.5，就没有问题了。

8.4.完成流程实例中的最后一个任务时报错（任务实例结束时），或删除流程定义级联删除流程实例时，报错如下：

com.mysql.jdbc.exceptions.MySQLIntegrityConstraintViolationException: Cannot delete or update a parent row: a foreign key constraint fails (`itcastoa_20100909/jbpm4_execution`, CONSTRAINT `FK_EXEC_INSTANCE` FOREIGN KEY (`INSTANCE_`) REFERENCES `jbpm4_execution` (`DBID_`))

解决办法：把方言设为 MySQL5InnoDBDialect，不能是 MySQLDialect。