基于 jBPM4 的临时动态性需求研究*

苏 展,刘 锋

(安徽大学 计算机科学与技术学院,安徽 合肥 230039)

摘 要:介绍了国内目前的工作流领域特点,尤其对临时动态性需求(会签、撤销、任意回退等)的各种场景进行了分析,提出了基于 jBPM4 的解决思路及一种动态路由的方法,以解决临时回退的问题。

关键词:jBPM;临时动态性需求;动态路由

中图分类号:TP319 **文献标识码**:A

Research on temporary dynamic demand based on jBPM4

SU Zhan, LIU Feng

(School of Computer Science and Technology, Anhui University, Hefei 230039, China)

Abstract: This paper introduced the domestic present workflow field, especially for temporary dynamic characteristics of various needs are analyzed, and the scene is proposed based on jPBM4 solution. Finally puts forward a dynamic routing method to solve the problem of back temporarily.

Key words: jBPM; temporary dynamic demand; dynamic routing

工作流管理技术[1-2]作为一种过程建模和过程管理的核心技术,是新兴于 20 世纪 90 年代的一种信息化技术。随着近些年国内企业、政府信息化的建设,工作流技术被越来越广泛地应用于业务流程管理(BPM)领域。jBPM 是一种轻量级的 J2EE 开源框架,具有良好的可扩展性。其最新版本较 jBPM3.2 版本,在核心引擎等方面都有较大改动,本文总结了新版本的特点。结合国内工作流领域的应用特点,本文特别针对临时动态性需求,分析了各种场景的情况,给出了应用 jBPM4 的解决思路,为解决临时回退的问题,提出了一种动态路由的方法,该方法亦可以用于解决各种临时动态性需求的场景。

1 jBPM4 特点

jBPM 是 JBoss 众多开源项目中的一个工作流开源项目,也是目前应用最广泛的工作流项目。在 2009 年 7 月 10 日,JBoss jBPM 团队发布了 jBPM4 的正式版本。jBPM4 完全基于流程虚拟机(PVM)的机制,对核心引擎进行了重新设计,而 PVM 的引入也使 jBPM4 可以支持多流程语言[3]。jBPM4 特点如下:

(1)在流程定义的对象上,节点类型划分更清晰。

(2)基于观察者模式的事件监听(Event-Listener)机制。在 jBPM4 中活动节点对象(ActivityImpl)、转移对象(TransitionImpl)、流程定义对象(ProcessDefinitionImpl),全都继承了 ObservableElementImpl 对象,因此,组成流程定义的 3 个主要元素都可作为观察者模式中的被观察者来观察,而这些对象本身就直接支持注册各种事件(Event),然后由监听器(Listener)来监听这些 Event。

(3)基于 ExecutionImpl、Command 模式和 AtomicOperation 实现的内核引擎在 jBPM4 中用 ExcutionImpl 代替 jBPM3 中 Token 机制。流程的流转依赖于 ExcutionImpl 调用各个 AtomicOperation 原子操作,如 ExecuteActivity、MoveToParentActivity、TransitionTake、TransitionStartActivity、ExecuteEventListener、TransitionEndActivity 进行推进,而 ExecutionImpl 在各个实例对象之间进行转移。

(4)加入了历史库,但还不完善。

2 国内工作流领域的特点

目前国内在工作流领域[4]主要的应用是人工工作流,也就是以人工任务密集型的工作流为主。随着国内企业和公共组织的信息化发展越来越快,IT 系统的积累和建设经验也越来越丰富,以自动任务密集型为主的应

* 基金项目:安徽省高等学校省级自然科学研究重点项目(KJ2007A43)

用将会逐渐增多。人工任务密集型工作流的应用主要集中在以下领域：电子政务，行政审批，企业协同办公，电信、电力的工单，企业采购、合同、销售等。

从功能需求上，这些人工任务密集型流程的典型特点主要有：(1)用户友好的流程定义工具。(2)表单能够自定义。(3)灵活的临时动态性需求，例如：任意回退、会签、撤销等。这些需求，存在很强的人为性因素。

国内专注于工作流产品方面的公司起步较国外晚，其产品大多只专注于某个行业的具体需求，如：有生博大的工作流产品主要定位于“电子政务系统中的审批流”；西安协同的工作流产品偏重于“电信行业”；信雅达的工作流产品偏重于“金融行业”；上海东兰的工作流产品则偏重于“协同领域”等等。

3 各种临时动态性需求情况的解决思路

jBPM4 是由国外开源小组领导的项目，流程定义工具和表单设计并不是其项目的特点，但其在流程处理方面的思想有其特色，特别是流程核心引擎的架构很值得学习研究。笔者在学习研究的过程中，对国内灵活的各种临时动态性需求的场景进行了模拟分析，提出了如何应用 jBPM4 来解决问题的思路。

3.1 会签

会签在政府或企业都是必不可少的功能，尤其是审批流中。会签可以分为单步会签(只有 1 个审批环节)及多步会签(每 1 个子审批流有多个审批环节)。单步会签：在流程的某个环节需要由多个办理人(多个不同部门的领导)共同办理或者签署意见，这是企业或政府的内部最为常见的情况。多步会签(也称为并联审批)：其实质就是一个单步的审批环节变为了在部门内部一个比较复杂的审批流程，这个审批流程有多个审批环节。

多步会签的情况较为复杂，本文只提供应用 jBPM4 解决单步会签的思路：即对 TaskService 进行扩展开发，实现动态任务实例的创建，参照 TaskActivity 类中的方法进行扩展，扩展后再调用 addTaskParticipatingUser( )或 addTaskParticipatingGroup( )方法实现动态增加任务办理人，此时即实现了单步会签功能。

3.2 撤销

撤销是指任务在发送给下一个办理人之后，发现任务发送错误，此时在下一个办理人还没有办理之前，可以撤回当前任务，而重新选择其他人进行办理。串行流程如图 1 所示。

图 1 串行流程示意图

节点 2 的处理人(假设是王二)办理完毕之后，将任务提交，此时任务到达了节点 3(假设李三办理)，这时李三就会收到一个待办任务，在李三还没有办理之前，王二突然发现，有 1 个业务数据填写错误，或者粘贴的附件错误，这时王二需要将发送给李三的任务撤销，重新更正数据后或修改粘贴的附件后再发送给李三审批。此外，假设节点 3 的办理人有 2 个人(李三和赵五)，那么王二需要在运行期间根据业务特性手动地选择任务是提交给李三还是赵五，但由于王二的误操作，把本来应该发给赵五的办理任务错发送给了李三，此时，在李三办理之前，王二也可以将发送给李三的任务撤销，然后重新发送给赵五。

对于上述模拟的撤销，jBPM4 解决思路如下：

(1)删除需要撤销的任务实例及其与此任务实例相关的所有工作流实例。在文件 TaskServiceImpl.java 中，jBPM4 提供了级联删除任务实例的相关方法：

```
public void deleteTaskCascade(String taskId) {
    commandService.execute (new DeleteTaskCmd(task-
Id , true));
}
```

(2)修改当前任务实例的状态。即将张三的已经办理完毕的节点 2 对应的 TaskInstance 的状态更改为待办状态：(Task.STATE_OPEN)

```
task.setState(ask.STATE_OPEN);
taskService.saveTask(task);
```

3.3 任意回退

回退作为审批流最常见的需求，每 1 个审批环节都有可能会有审批通过、不通过 2 种情况。审批不通过时，一般是回退到上一个环节，但是在某些情况下，有可能跨环节回退，而到底回退到哪个环节，可以由用户根据业务需求进行自定义，并且在回退环节工作完成之后，其下一步的方向也可以由用户自定义。下面根据各种不同的回退分别模拟并给出解决思路。

3.3.1 串行流程

串行流程，最直接的方式就是在需要回退的各个节点之间建立回退线，如图 2 所示。如果需要从节点 4 回退到节点 2，则直接在节点 4 之后加 1 个分支决策节点，连接 2 个分支，1 个是流向节点 5，1 个是返回节点2。

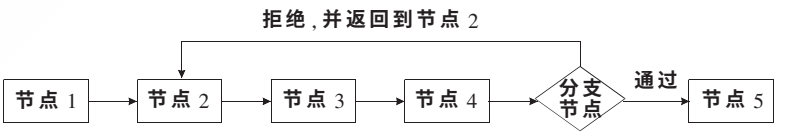

图 2 串行流程的回退实现

对于节点 2，在 jBPM4 中，其参与模式又分为 4 种：task－assignee(assignee user、assignee expression)、task candidates(candidate－groups、candidate－users)、task assignment handler、task swimlanes。

(1)task－assignee 任务的办理人的值可以直接指向 1 个特指的用户的 ID 或者 1 个关联到业务中某个特指用户的表达式(如 order.owner)。此时，如果 assignee 赋予了1 个特指用户的 ID，回退时不用做任何处理；如果 assignee

赋予了 1 个业务表达式，在回退时，需要保证业务表达式的运算逻辑能够正确执行。

(2)如果 task candidates 任务的办理人为几个特定的组的集合或者用户的集合，实际上就是俗称的“竞办”。这样的任务被称为 groupTask，必须被某一个组或者用户拾取才能进行办理，因此，如果回退到这样的节点时，需要把任务回退给当时的拾取人。而拾取人需要到 HistoryTask 和 HistoryDetailImpl 2 个实体对应的历史库中取得。

(3)task assignment handler 任务的办理人通过用户自己编写的代码来实现，当回退到这样的节点时，也不需要做特殊处理，只要保证自己的代码(AssignmentHandler)在执行回退逻辑时同样能够正确执行即可。

(4)task swimlanes 任务的办理人为“泳道”，回退到这样的节点时，任务的办理人可以自动取得，同样不需要做任何处理。

以上 4 种情况，在回退之后的处理又分为 2 种情况：一种是按照原来的路径重新执行，即节点 2→节点 3→节点 4；另一种是，节点 2 办理完毕后，直接返回给节点 4。对于第 1 种情况，不需要做处理。而对于第 2 种情况，由于在节点 2 和节点 4 之间并没有一个转移存在，因此 jBPM4 也没有提供支持。针对这种情况可以通过动态路由(即动态创建转移)的思路来实现(具体实现思路见第 4 节)，实际上回退也可以用动态创建转移来实现，但不用画回退线了。

3.3.2 *M* 选 *N* 分支流程

M 选 *N* 分支流程中，*N* 必须满足 $M>N\geq 1$，假设回退前流程的执行路径为节点 1→节点 2→节点 4→节点 5，如图 3 所示，此时回退有 2 种情况：

(1)由节点 5 回退到节点 1，而节点 1 在进行业务数据的修改后，直接返回给节点 5 的处理人进行办理或审批。

(2)由节点 5 回退到节点 1，而节点 1 在进行业务数据的修改后，重新按照节点 1→节点 2→节点 4→节点 5 的路径再执行 1 遍。

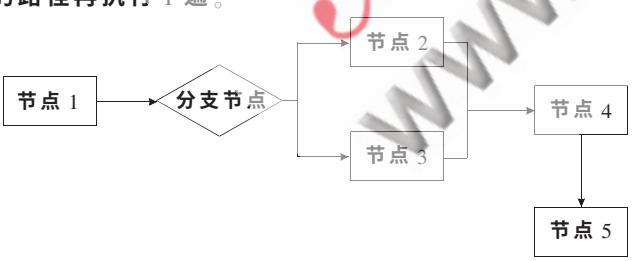

图 3 分支流程示意图

情况(2)与情况(1)不同的是：在回退之后继续审批时，需要考虑分支节点的决策条件，在自动决策时要保证决策逻辑正确执行；在人工决策时，需要记录原来的执行路径(保证重走节点 1→节点 2→节点 4→节点 5)。

3.3.3 同步分裂、“与”汇聚流程

同步分裂、“与”汇聚流程如图 4 所示，回退前执行的路径为节点 1→节点 2→(节点 3、节点 4)→节点 5→节点 6。此时回退有 4 种不同的情况：

(1)由节点 6(或节点 5)回退到节点 2(或节点 1)，节点 2 重新办理完毕后，直接返回给节点 6 的处理人进行办理或审批。

(2)由节点 6(或节点 5)回退到节点 2(或节点 1)，节点 2 重新办理完毕后，重新按照节点 2→(节点 3、节点 4)→节点 5→节点 6 的路径再次执行 1 遍。

(3)由节点 6(或节点 5)回退到节点 3(或节点 4)，节点 3(或节点 4)办理完毕后，直接返回给节点 6。

(4)由节点 6(或节点 5)回退到节点 3(或节点 4)，节点 3(或节点 4)办理完毕后，按照节点 3(或节点 4)→节点 5→节点 6 的执行路径，再次执行。

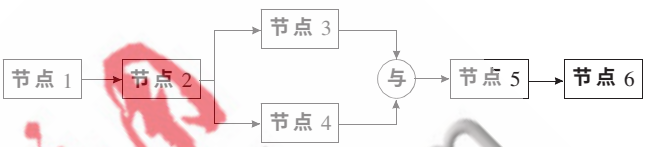

图 4 同步分裂，与汇聚流程示意图

情况(1)、(2)和(3)处理起来没有什么问题。但情况 4 中，当流程由节点 6 回退到节点 3，节点 3 办理完毕后，重走路径节点 3→节点 5→节点 6 时，在“与节点”的“与”运算逻辑就会无法执行，因为另 1 个“与”分支(节点 4)并没有新的实例产生，流程就会僵死此处。所以可以通过修改 JoinActivity.java 这个类文件中的方法：

protected boolean isComplete(List<executionimpl> joinedExecutions, Activity activity)

在此方法中加入对该情况的处理代码即可。

3.3.4 同步分裂、“或”汇聚流程

同步分裂、“或”汇聚流程如图 5 所示，回退前执行的路径为节点 1→节点 2→(节点 3、节点 4)→节点 5→节点 6。此时回退同样有 4 种不同的情况：

(1)由节点 6(或节点 5)回退到节点 2(或节点 1)，节点 2 重新办理完毕后，直接返回给节点 6 的处理人进行办理或审批。

(2)由节点 6(或节点 5)回退到节点 2(或节点 1)，节点 2 重新办理完毕后，重新按照节点 2→(节点 3、节点 4)→节点 5→节点 6 的路径再次执行 1 遍。

(3)由节点 6(或节点 5)回退到节点 3(或节点 4)，节点 3(或节点 4)办理完毕后，直接返回给节点 6。

(4)由节点 6(或节点 5)回退到节点 3(或节点 4)，节点 3(或节点 4)办理完毕后，按照节点 3→节点 5→节点 6 的执行路径，再次执行。

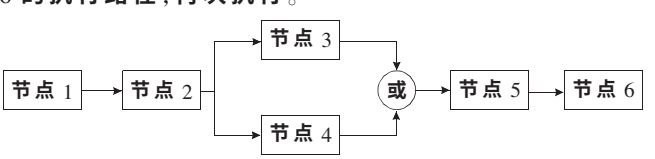

图 5 同步分裂，“或”汇聚流程示意图

由于是“或”汇聚，因此在实现上不需要做任何其他处理了。

综合以上情况,回退本身在理论上有各种各样的情况存在,再加上业务的回退(或者业务逻辑补偿,如果需要就必须在流程的每个环节进行业务快照的备份,在回退时调用这个快照进行补偿),此时回退可以说是整个流程体系中最复杂的。但是在真实的业务情况中,有些可能是根本不会出现或者很少出现,因此要考虑的是能不能通过变通的方式处理。

4 动态路由的思路

针对于特定的业务实例,有时需要在原本没有转移关系的环节之间进行特定的跳转,例如在图1中的串行流程中(1-2-3-4-5),节点4与节点2之间是没有任何转移存在的,但是针对于某个运行期的特定业务实例,临时要求环节直接从节点4回退跳转到节点2(而略过了节点3)。此时就需要在节点4和节点2之间由程序动态地创建一个转移(transition),并设定为优先级最高,如图6所示。

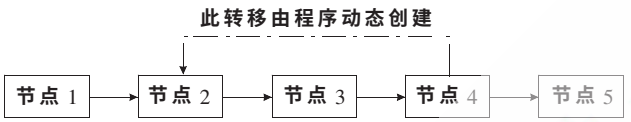

图6 动态路由实现示意图

因此在执行takeTransition( )方法时,按照优先级将优先执行动态创建的转移,然后对外暴露1个jumpTransition(String destinationActivityName)方法给客户端。应用jBPM4实现时可按照如下步骤进行扩展:

(1)参照文件CompleteTaskCmd.java扩展开发用于跳转的cmd:JumpTaskCmd。

(2)参照文件TransitionStartActivity.java的原子操作,定制开发用于跳转的原子操作JumpTransitionStartActivity。

由于jBPM4中的执行动作都是基于Command模式实现的,因此在扩展开发自己的跳转动作时,就可以做到对jBPM4本身的代码不做侵入修改而实现。这种临时建立动态路由的思路不仅可以用于回退,同样可以用于重新操作后的跳转的实现。

jBPM作为目前应用最广泛的开源工作流产品,其最新的第4版有着很好的架构及扩展性。但是由于国外的流程应用与国内的应用有所不同,因此要想让jBPM4更好地满足国内的流程应用的需求,还需要做一定的定制开发。本文针对国内的流程应用的典型特点进行了较详细的分析,特别针对临时动态性需求的各种情况,给出了基于jBPM4的解决思路(限于篇幅文中并没有给出很详细的可以运行的代码)。本文所做的工作只是工作流应用技术研究的一小部分,还有很多方面需要继续学习和实践,需要进行深入的研究。

参考文献

[1] 范玉顺.工作流管理技术基础[M].北京:清华大学出版社,2001.

[2] WfMC-TC00-1003v1. 1. The workflow reference model[S]. 1995.

[3] JBoss官方.jBPM Documentation[EB/OL].http://www.jboss.org/jbossjbpm.2009-09.

[4] 胡长城.工作流讲解[EB/OL].www.javafox.org.2009-09.

(收稿日期:2009-10-14)

作者简介:

刘锋,男,1962年生,教授,主要研究方向:软件工程、并行计算。

苏展,1985年生,男,硕士生,主要研究方向:工作流、软件工程。