
# 工作流模型分析

## Workflow Patterns

版本：1.1

作者　　：胡长城
网上游名：银狐 999 ；james999
创作时间：2003-11-27
联系信箱：james-fly@vip.sina.com
MSN　　：fcxiao2000@hotmail.com

引文：

本文是原“ 工作流模型分析 ”的补充和完善。其最初版本，可参看 http://www.huihoo.org/jfox/jfoxflow/workflow_model_fox999.pdf

谈论工作流模型，也不仅仅是为了谈论。理论需要实践去检验；而实践业需要理论去指引方向。任何有关 Workflow 的开发，都需要基于一些特定的流程模型去处理，所以首先就需要确定一个完善成熟可用的“ 流程模型 ”。希望此篇能够让大家明白一些基本的运转模型。写这篇文章，也算“ 以文会友 ”，希望与更多得研究工作流的朋友交流。

下面就以在“ 小议学习 Java 的浮躁心态 ”中所说的一句话，作为开篇。

> 知识是需要沉淀的；思想也是在不断的学习、磨练中走向成熟的；而技术也是在不断的创造中开拓的。
>
> ——银狐999

# 目 录

# 1. 概述：

有关“什么是工作流”和工作流的概念，就不在这里介绍了。大家有兴趣的可以到 WFMC 上看看。在 WFMC 的 xpdl 规范中，没有 Task（任务）概念，而是 Activity（活动）概念。不过大多数情况下，我们对“任务”的理解，可能更加容易。所以，在本篇中，都是采用 Task 概念。有关为什么选择 Task，在“任务与活动”一章中有讲解。

这里先说说个人的看法：一个工作流包括一组任务（Task）及它们的相互顺序关系，还包括流程及任务的启动和终止条件，以及对每个任务的描述。其实这是摘自 http://www.simflow.net/workflow/workflow.htm 上一段话有关工作流的描述。只是原文叫“活动”，我改为任务（Task），可能更好理解一些。

现实应用中，不光单工作流的应用较为广泛，多工作流之间的嵌套或整合也有广泛的应用市场。在本文后面的章节会简要的分析。

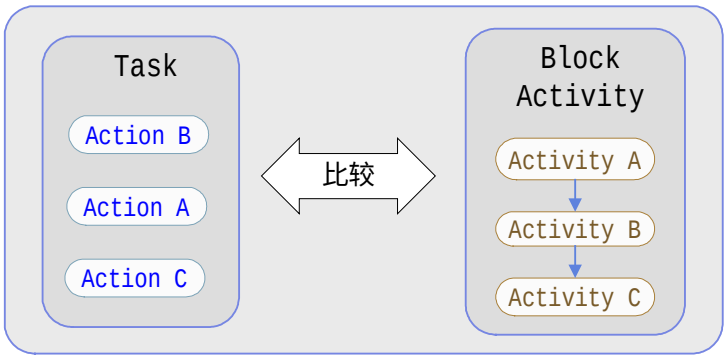
在后续的文章中，大家可能会发觉本文没有提及“事务”（Transaction）控制。在有些工作流模型文章中，在讲解模型的时候，会将事务也融合进来。本文没有讲“事务”融合进来，是因为“事务”，以及“角色权限控制”“组织模型分配”等等模块，虽然对工作流来说，是必不可少的控制模块，但是针对流程的运转模型，不是根本性的影响因素。

# 2. 任务与活动

活动（Activity）是 WMFC 的标准模型元素，描述的是工作流中的一个活动——“A description of a piece of work that forms one logical step within a process”。在 XPDL 的规范中，Activity 已经是描述流程远转的最小单元。

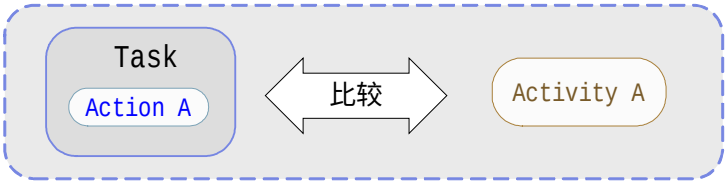
在本文中，我采用的是“任务”（Task）概念。在描述业务中，一个任务表示的是流程的所需要完成的某一项工作，这项工作可能是一次操作（Action）即可完成，也可能是几次操作的组合。其有些像 XPDL 中的 Block Activity。但请注意，不等简单的等同，这个在模型构建的时候，是个“划分粒度”问题。粒度的划分在应用中，会涉及到很多，最为直接的就是权限控制。

大家参考一下图（1-1）。列出了 Task 与 Block Activity 的相似比较。稍微注意以下，在左边的 Task 中，多个 Action 之间没有连接线。而在 Block Activity 中，各个 activity 却需要按顺序执行。




**图（2-1）**

下面的图（1-2）可能又让大家迷惑了。上面刚刚才说的 Task 与 Block Activity 有些相似，下图却拿 Task 与单个 Activity 比较。可以注意一下，左边的 Task 只有一个 Action 可以执行，和右边的 Activity 表达的意思就很相似了，都表示需要执行一个活动/动作。




**图（2-2）**

上面的比较，更多的是个人的一些看法和思考。以前从事过用 Task/Action 表示的工作流开发，而 XPDL 却采用了一些 Activity 表示。在考虑“到底用什么方式表示更好”的这个问题的时候，做了上面两种的类似比较。可能未必正确，也存在商榷的地方。

真正在应用中，采用什么“ 粒度划分 ”，采用哪个名词表示，还是根据应用来决定了。Xpdl 采用 Activity/Block Activity，但是在其他的一些工作流模型表述中，有相当一部分采用 Task 等概念。

**不过，本文后续将采用 TASK 概念表示一个任务。**

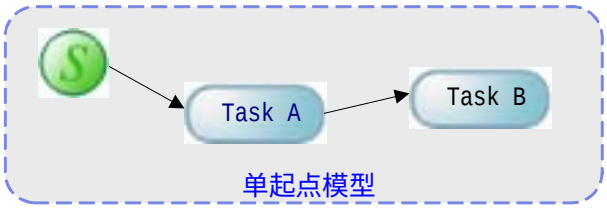
# 3. 流程起点模型

任何事物都有由头有尾，一个流程也不例外，那么现在我们就从流程的“ 头 ”——流程起点说起。一个工作流能够运行，就需要一系列条件来激活。而这个激活的位置，就是“ 起点 ”（Start Node）。

需要说明的是，起点也是一种任务节点（Task Node）。这个特殊的任务节点也许会执行一定的操作，也许仅仅只是一些数据状态的改变。但是，不论什么原因，最终会导致一个流程实例的产生——流程被激活了。

## 3.1 单起点

单起点（Single Start Node）估计大家都比较容易理解，现实中工作流应用的也是最为普遍。如下图（3-1）所示，其就是单起点的模型。




**图（3-1）**

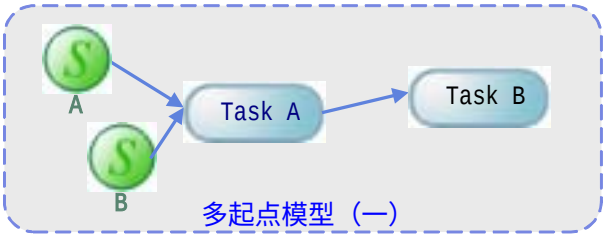
## 3.2 多起点

多起点（Multi Start Node ）的工作流，在现实应用不是太多。其主要表达的是：在同一流程中，存在多个起点。说到这里，有必要重新申明一下：起点也是一种任务节点，而不是独立于工作流任务特殊节点。

多起点的工作流模型，基本上有如下三种方式。

### 3.2.1. 多起点方式一

请参考图（3-2），起点 A 和起点 B，它们都可以激活流程的运行，而且激活后，流程都会共同指向 Task A。所以，对于 Task B 来说，其不关心流程是如何激活的，其只关系从 Task A 是否正确的传递来正确的流程数据。

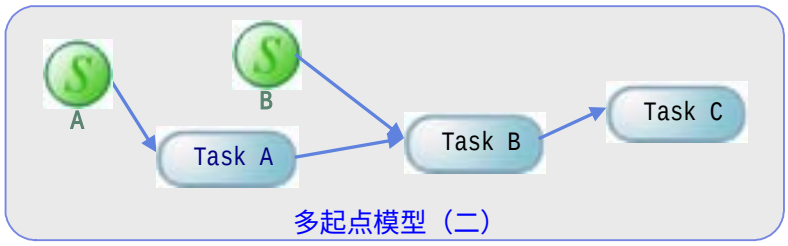



图（3-2）

## 3.2.2. 多起点方式二

在方式二（如下图（3-3））中，起点 A 激活工作流后，导致流程沿着 Task A——Task B——Task C 方向流转。而从起点 B 激活工作流后，Task A 则被跳过。

这种方式，在现实中是极为少见的。如果将 Start B——Task B 这条流程段 与 Start A——Task B 这条流程段，分开来看。则可以近似看作的两个“子流程”的选择性汇总（两选一，或多选一）的情况。

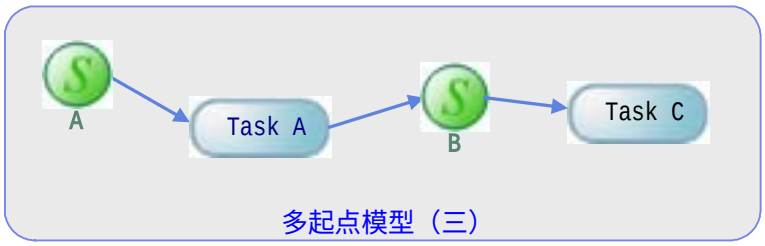



图（3-3）

## 3.2.3. 多起点方式三

方式三（如下图（3-4）），虽然也存在多个起点，但是基本是按照一个统一流程方向运行的。这是与方式二最大的区别所在。在此，须要再此申明：一个起点（Start Node）同时也是一个任务节点（Task Node）。参看图中的 Start B 节点。

此种方式，在现实中，还是有一定应用性的。特别是在多个流程之间信息交互的时候，流程 A 发送消息数据，激活流程 B 的运行。但是未必是从流程 B 的默认激活点激活，可能是从流程 B 的中途某个任务激活。比如图中的 Start B 任务节点。




图（3-4）

# 4. 流程激活模型

上面我们看了工作流的起点模型。也知道任何流程，都必须有起点，或者相对的起点。一个流程被激活后，会从起点开始沿着预定的流程路线，有序或无序的往下进行（注意，我这里提到了“ 无序 ”二字，我将在后续讲解“ 无序 ”状态）。

所以，起点就是这个流程被激活的源头。下面让我们来看看，一个流程被激活的方式，或者说一个起点，被激活的方式：

从起点的激活方式，有如下两种方式：

## 4.1 人工激活

大多数的流程激活，都是因为人为的信息数据输入或产生。比如一个订单处理流程，客户提交了订单信息（订单信息数据产生），则激活了订单处理流程的开始。

## 4.2 定时或限时激活

在一个特定的时间，因为特定的情况，符合特定的条件，激活某个特定的流程（或任务）。

这种激活方式，在现实中很少单独出现，大多数情况，都因为在某一个流程中，因为在限定的时间内，因某项任务未达到预期的状态，而激活另外的任务或新的处理流程。也就说，这种方式，是受外来因素影响的，而且大多与一些流程任务（或流程模式）一起出现。

举个定时激活的实例：比如，订单处理流程，限定 5 天内发货，那么定义在第三天的时候，如果没有接到发货通知，则激活一个催办信息（催办任务）。这样流程系统，会在第三天的时候自动发出催办信息。

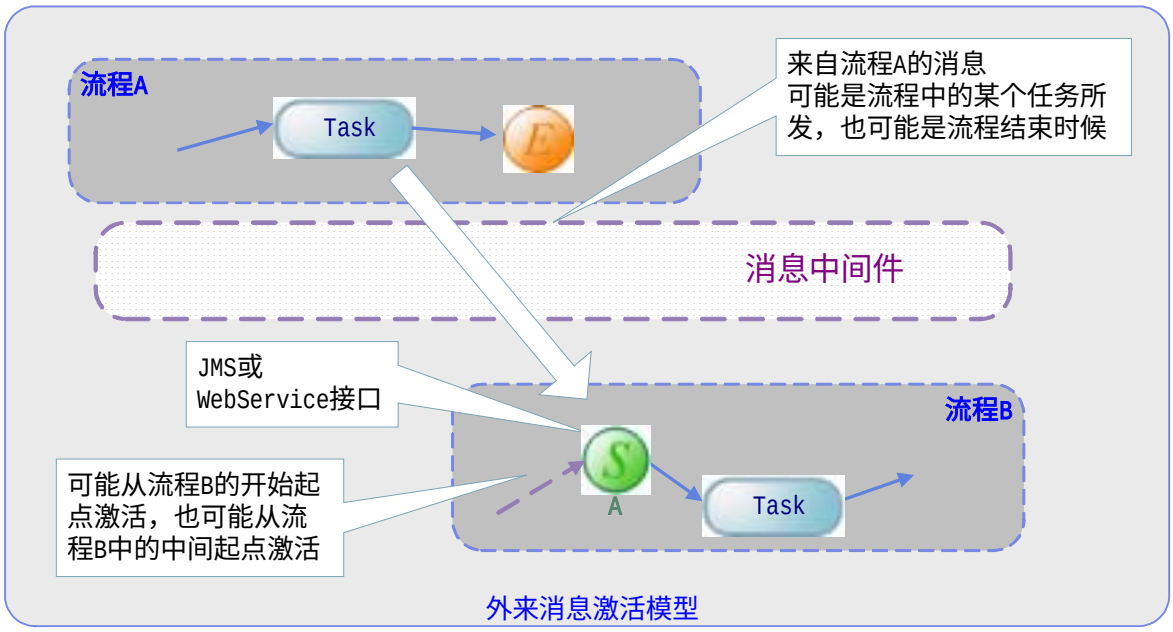
## 4.3 外消息激活

这种方式，大多是在多流程信息交互（或大小流程嵌套）应用中。现在比较流行的业务流程整合/管理（BPM），基本上都涉及到这方面内容。

如下图所示，流程 A，在结束的时候（在以下的所有图中，将采用红色框图，表示结束节点），会向流程 B 发送 Message，以激活流程 B 的运行。至于这个消息是 Soap 消息，还是通过消息中间件转发的 Message，这就是不同的应用方式了。

一般现实应用中，都需要考虑 JMS 或 WebService 的应用接口。从个人目前所实施过的工作流应用来说，大多还是采用 Message Query 方式居多。虽然软件的发展，逐渐 SOA（面向服务）化，但是 WebService 的安全性或数据正确性，还有待进一步的发展，从这一方面说，比起消息中间件的高度安全性和消息正确性，WebService 目前还是稍逊一筹。其实，安全性和信息正确性，是很多应用客户非常关心的焦点。

但是，SOA 化的发展是未来的趋势。所以现在大多的应用都会提供 JMS 和 WebService 接口，或其他类似接口。




**图（4-1）**

上面说了那么多，但是我想很多工作流的文档，都少有提及。什么起点模型，什么激活模型，似乎都是一个陌生的概念。但在现实中，如何去处理好一个流程的起点，特别是存在多个起点情况，是件至关重要的事情。因为流程必须从起点激活（这里的起点可以是开始起点，也可以是中间起点），而且激活的方式又是什么？是人工操作激活，还是 Timer，或者外界信息的激活，都有可能。

当然，至于如何激活，Timer 如何设置，外界信息如何正确的处理等等问题，就不是本篇的讨论主题。但在将来的文章，我会专门说明。

# 5. 流程运转模型

下面就让我们进入“ 工作流模型 ”（workflow patterns）的核心地带哦。在先前的《工作流模型分析》中，这一章更多讲的是个人对工作流模型一些理解，有些不完善。所以，在当前 1.1 版中，就参考了 workflow patterns 模型，以及 WFMC 的 XPDL 规范。

本章也将是本文最为核心的地方了，什么是工作流，也将在其运转模型中体现。一个流程是一系列的任务（Task）按照某种预定的方式，组合起来的。

说到这里，可能对 XPDL 规范深研的同仁，就要提出疑问了。在 WFMC 的 XPDL 规范中，仅有 Activity（活动）概念，而没有 Task 概念。这两者之间有什么关系吗？

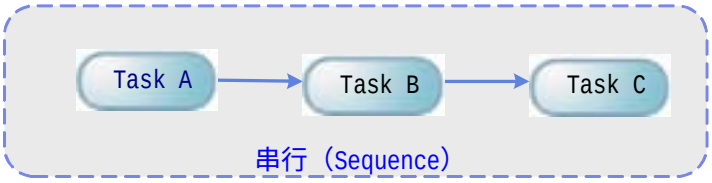
首先，他们的概念范畴是不一样的（或者说粒度不一样），虽然在图形上看，也是一样的连接，一样的框图。但是，在 XPDL 规范中 Activity 已经是最小的动作单元，而这里所说的 Task 则不是，其实际上还包含多个 Action（动作）。

有关这些 Task 和 Action，与 Activity,Block Activity。之间到底存在什么样的粒度差异，不是本篇的重点。这个也会在后续的文章中，单独详细讲解。

# 5.1 简单运转模型

## 5.1.1. 串行

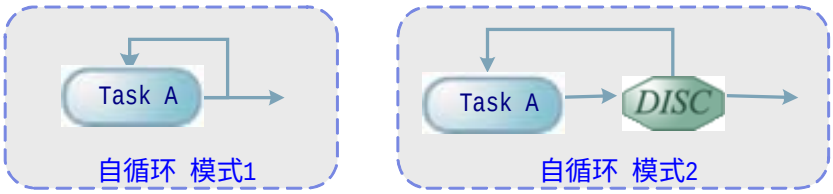
串行（Sequence）是最为简单，也最为容易理解的模型。按照预定的任务列表，有序的执行，如下图（5-1）所示。




**图（5-1）**

## 5.1.2. 自循环

自循环（Self-Cycle）的模型，主要用于表示：同一个任务节点，重复的执行多次。




**图（5-2）**

如图中所显示。“ 模式 2 ” 比 “ 模式 1 ” 多了一个鉴别节点（Discriminator Node）。这两种模式，在现实中应用的都较为广泛，其中 “ 模式 1 ” 更多的偏向人为的选择，也就是说，在任务执行后，由人为的决定是否继续重复的执行这次任务；而 “ 模式 2 ” 则更多的倾向于一个既定的规则，按照原有的规则，决定是否重复执行。

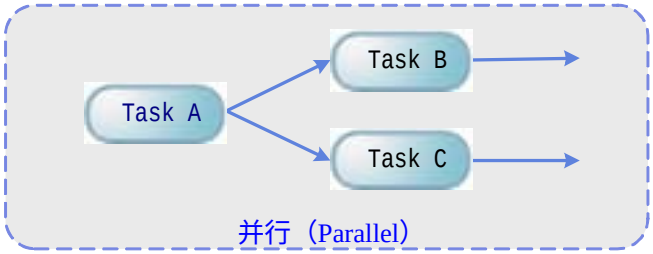
举例子：在电子政务办公系统中，经常出现的 “ 多处长联合审批 ” 过程。多个处长（个人）属于同一个处长角色（角色单元）。针对同一个审批过程，采用自循环（审批这个过程重复执行）就可以基本解决问题。

## 5.2 发散运转模型

### 5.2.1. 并行

并行（Parallel），在有的某型中叫“ And 模式 ”。

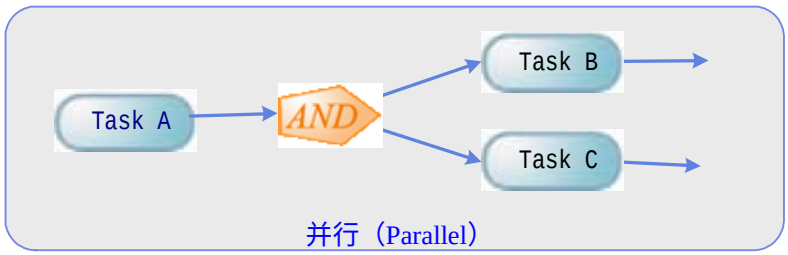
是说在流程运行过程中，因为不同的条件或情况，或者处理的业务需要多部门（多任务）分开处理，而产生了流程分支。如下图所示




**图（5-3）**

流程在执行完任务 A 后，因为需要，产生了两个并发执行的分支（A——B 和 A——C）。这两个分支之间是对等的，也是并行执行的。

有关上面的流程图，可能在以后的一些文章/文档中，大家会看到下面类似的图形



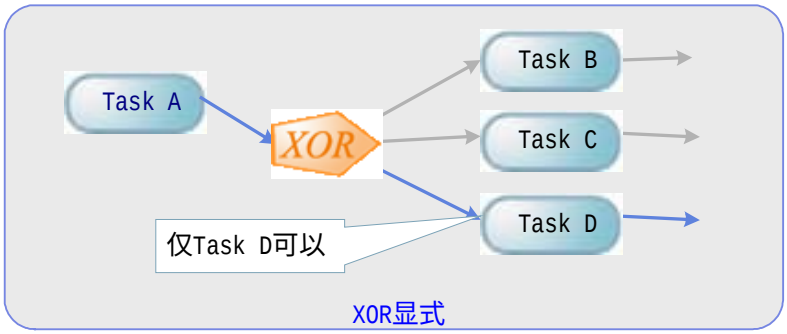

**图（5-4）**

虽然比上图多了一个 And 选择器，但实际上，两图，表示的是同一个含义或模型。所以大家在应用或读书的时候，可以长个心眼哦，自己学会实质性的分析。

### 5.2.2. 异或模型（显式）

异或（XOR）显式模型，又叫 Exclusive Choice（独占式选择）

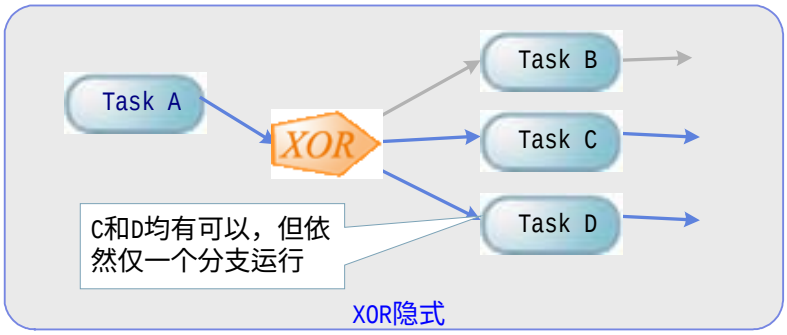
当一个任务处理完后，发现其后面可允许走多个分支流程，但只允许选择其中某一个分支运行。如下图所示，虽然在任务 Task A 后预订了三个不同的任务，但是仅 Task D 满足条件（一般多为人工操作选择），造成后续的流程中，走了 A——D 分支，而另外的分支被抛弃。




图（5-5）

## 5.2.3. 异或模型（隐式）

同为 XOR 模型，隐式和显式的区别不是太大。唯一的不同点就是，隐式 XOR 模型存在多种可选择的分支，但最终，依然仅有一个分支运行。如下图所示，存在分支 A—C 和分支 A—D 都满足条件，但最终也依然只能有一个分支被激活。至于哪一个分支被激活，这可能是人为的操作，也可能是某种随即的自动选择。不论哪种方式，人须保证一个分支被激活后，其它分支被抛弃。




图（5-6）

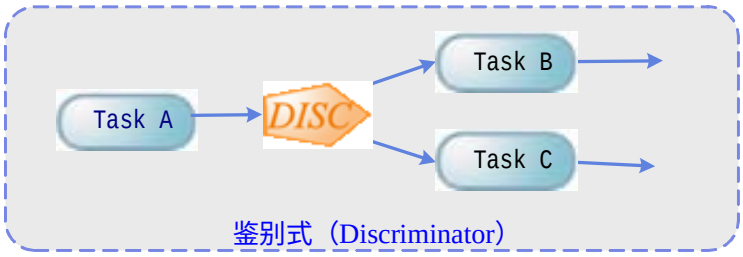
现实中，隐式 XOR 模型的应用非常少，而且大多数的工作流引擎不支持这种模型，仅支持显式 XOR 模型。在 XPDL 中的 XOR 模型中，并没有提出“显式”和“隐式”的概念，默认还是以显式 XOR 模型为操作方式。

大家在设计的时候，我想不考虑这种情况，应该不会有太大的应用麻烦。

## 5.2.4. 鉴别模型

鉴别模型，在标准的 Workflow Patterns 表述中，又叫 Discriminator Choice。

这同前面的“独占式选择”很相似，唯一不同点，就是多了一个鉴别器（Discriminator）。当任务达到这个鉴别器的时候，鉴别器会根据当前流程所处的状态，对比预先设定的一些选择规则，自动判别接下来流程的流向，也就是自动根据条件，选择一个满足条件的分支运行。




**图（5-7）**

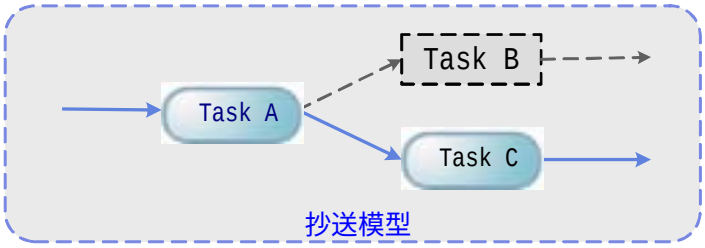
鉴别器模式（有的可能叫选择器等等名字，表达的意思基本相同），在现实应用中较为广泛。比如在订单申请流程中，设定一个依据数额判别流向的鉴别器，如果数额大于等于 5000 就走分支流程 A，如果数额小于 5000 就走分支流程 B。

## 5.2.5. 抄送模型

抄送模型，本身不是一个标准的工作流运转模型，但是在现实应用中，比比皆是。

它表达的意思是（请参考下图），存在主流程（A——C），在一个任务（A）执行完毕后，会继续执行主流程上下一个预定任务（C），但是同时也会激活另一任务（B）（或另外的流程）的执行，但是任务 B 以及任务 B 的后续流程，不会对主流程运转造成影响。

请注意图中的 A——B 流程沿线，用的是灰色虚线表示，而且任务 B 也同样采用灰色表示。




**图（5-8）**

来个举个电子办公系统中，经常遇到得例子说明一下：比如一个发文，在交司局会签的时候，可能会抄送一份给另外的司局备案，这个过程就或额外的激活一个不影响主会签流程的“ 抄送任务 ”，比如图中 Task B。

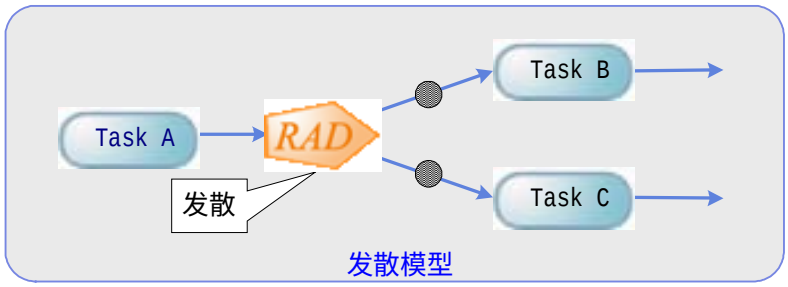
## 5.2.6. 发散模型

说到这里，大家可再回过头参看一下并行模型（5.2.1 节）。发散和并行最大的区别就是，各个分支（branch）的流程状态（或流程数据）：

在并行模型中，分支状态（A-B）与分支状态（A-C）是大多数情况下是不相等的。由任务 A 执行后的状态进行一定条件下的“ 拆分 ”，形成了两个分支（或多个分支）流程。这多个分支流程，在最终需要重新聚合成一个主流程，以确保流程信息的完整性（当然，实际运行中，可能存在因为超时等特定原因而最终抛弃某个子流程）。

而在发散模型中，分支状态（A-B）与分支状态（A-C）是绝对相等的。因发散而产生的多个分支流程，在最终未必聚合（可能因为种种原因，聚合的时候会抛弃一个和多个分支流程）。

这里面说到了“ 聚合 ”概念，在后续的介绍上，将加以详细叙述




图（5-7）

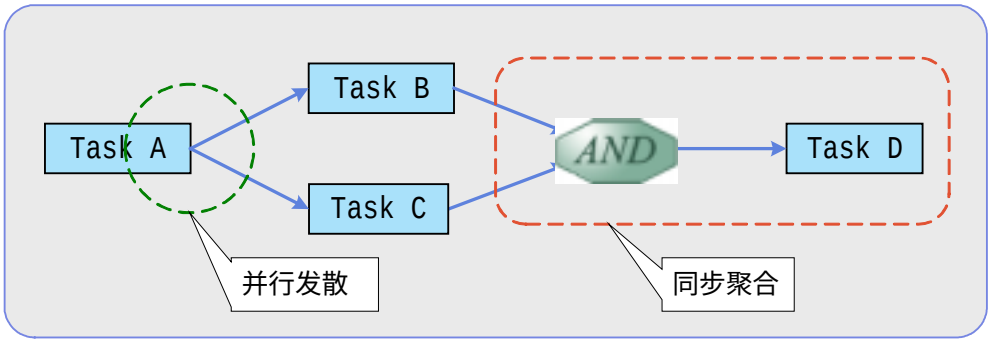
## 5.3 聚合运转模型

下面我们就将进入聚合模型的介绍。因为有了“ 发散 ”，在一个流程的后续运转中，才会出现“ 聚合 ”这个问题。所以在后续讨论聚合模型的时候，大多情况下都会结合上面的发散运转模型。

### 5.3.1. 同步聚合

由必要说明一下，同步聚合（Synchronize merge），可不是“ 同时聚合 ”噢。

同步聚合表示的是：在聚合点，会等待所有分支的到来，如果不考虑超时（一般流程回设定任务执行期限）和异常等情况下，流程必须等待所有的分支（Task B 和 Task C）都执行完（到达 And 汇聚点）后，才能激活后续的任务，也就说流程才能正确的往下运行。




图（5-10）

这里面会涉及到如何聚合的问题，是人工聚合，还是自动聚合。一般这里会引入规则引擎（Rule Engine）来负责分支的聚合，按照预定的规则，将流程数据（状态）汇聚。

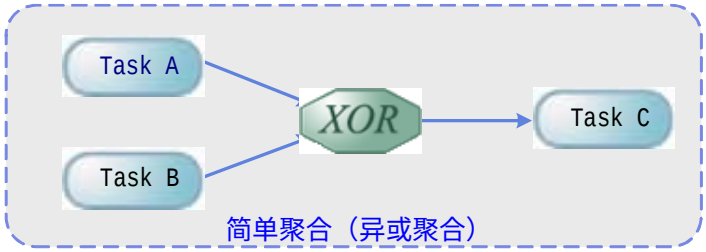
当然，现实中，分支聚合（或者分支发散）还会涉及到类似“ 触发器（trigger）”，“ 定时器（Timer）”，或者各式各样的“ 监听器（listener）”等等。这些在真正模型应用实施的中必须详细设计，考虑完善。这些内容超出了本文的范围，但大家在实际中需要多加关心。

## 5.3.2. 简单聚合

简单聚合（Simple Merge）模型，大多的模型文章都采用“异或聚合（XOR Merge）”这个名称。

虽然名为简单聚合，不过在现实应用中，其实并不简单。

这种聚合一般采用“多选一”的原则，或采用类似“先进先出”法则（人工干预除外）。在聚合的时候也有可能涉及到流程数据（状态）校验等问题。不够，如够涉及到过于复杂的筛选规则或数据校验规则，似乎采用下面的“鉴别聚合”更好。



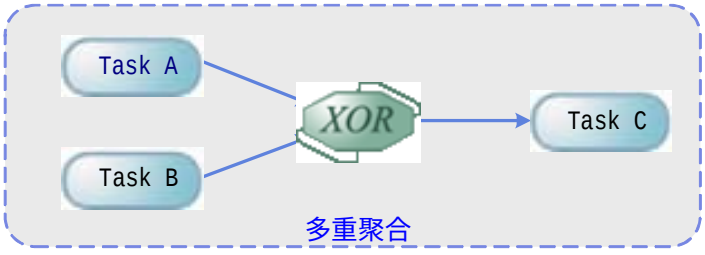

**图（5-11）**

当然，一旦某一个分支被通过。则余下的分支则被终止，或者运行到聚合点就结束。这里可能也需要详细分析一下。不过看来要等到下一个版本再说了。

## 5.3.3. 多重聚合

多重聚合（Multiple Merge），与上面的简单聚合有些相似。但是比起 Simple Merge 可就复杂多了。到目前为止，在现实中，我还没有碰到过这样的流程实施。

多分支在聚合的时候，采用类似于“先进先出”法则，但是不同于简单聚合的是，任何一个分支，在到达这个聚会点的时候，均会激活后续流程的运转。(这种情况，是参考 Workflow Patterns 写的，既然存在，想必现实中也有应用。但是后续流程被多次激活，就需要解决这些产生的多实例问题，就将问题过于复杂化。很多工作流产品不支持这种模型，也是比较明知的)



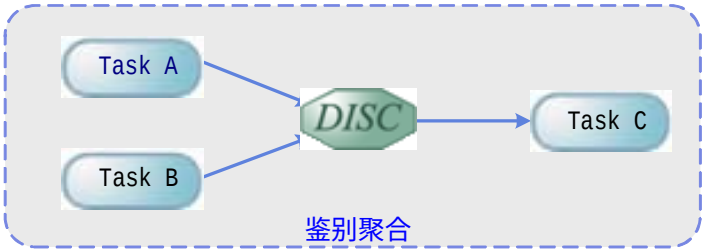

**图（5-12）**

还会涉及到一个问题：如果一个后续流程实例刚刚被激活（还没有运行完），又一个分支到达，那么这个分支是否激活后续流程实例呢？在不同的工作流引擎中（workflow enginner）中会有不同的解决方案，有的选择立即激活，有的选择等待延迟激活。就这一点来说，不是本文的讨论主题，有兴趣的朋友，可以在自己的引擎中实现不同的方式。

### 5.3.4. 鉴别聚合

鉴别聚合（Discriminator Merge）较容易理解，应用的也较为广泛。

一般情况下，流程中不大会独立存在。通常会结合“同步聚合”或“简单聚合”之类的存在。鉴别的目的，就是更准确的聚合，让那些符合特定条件的分支聚合。




**图（5-13）**

但在现实应用中，考虑的就比较多。在“同步聚合”中我也提到过，需要负载一些辅助性功能模块。通常都会存在一个“规则引擎”，来定义或处理聚合规则

### 5.3.5. 优先聚合

这个是在与 colin 兄讨论的时候，他给的建议。当时他提到了 priority merge（优先聚合），另一个是 counter merge。有关 Counter Merge 我确实没有多少概念，所作的工作流开发中，也没有应用过。colin 兄也仅仅是提到，也没有做详细的解释。如果哪为仁兄能够对 counter merge 探讨一下，不慎感激。

有关优先聚合，可能就需要对流程分支，赋予一定的优先级别。打破通常的“先进先出”的规则。

## 5.4 特殊运转模型

既然是特殊，当然是普通流程模型中所少见的。在 Workflow Patterns 中，以下的几种模型似乎都没有被提及过，也许老外的流程应用较为规矩。

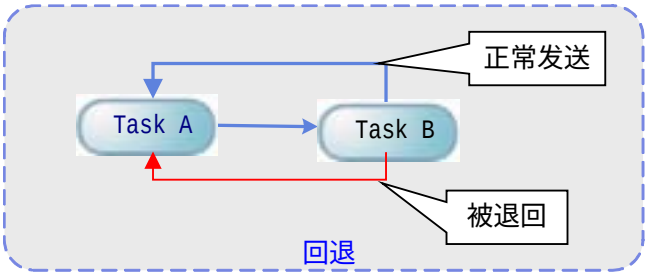
以前我是从事电子政务流程开发的，下面的几种模型，在电子政务的工作流应用中，较

为普遍。说出来，也算参考一下。虽然标准没有提及，但并不代表应用没有。Wfmc 的 xpdl 标准更多的是基于工业化流程处理，偏向自动化处理事务，所以人为的参与就比较少。但是国内目前的工作流应用，大多就基于人的参与，所以人为的操作（包括权限）就很重要。

### 5.4.1. 回退

回退，在有的应用中叫“ 退回 ”。

如下图所示，有任务 A 到任务 B 属于正常发送，但从任务 B 到任务 A，则出现两种情况：（1）正常发送，如图中 B—A 蓝色线；（2）可能因为某些特殊原因，被任务 B 退回，要求任务 A 重新办理，如图中 B—A 红色线。虽然都是从 B 到 A，代表的意义却完全不同。

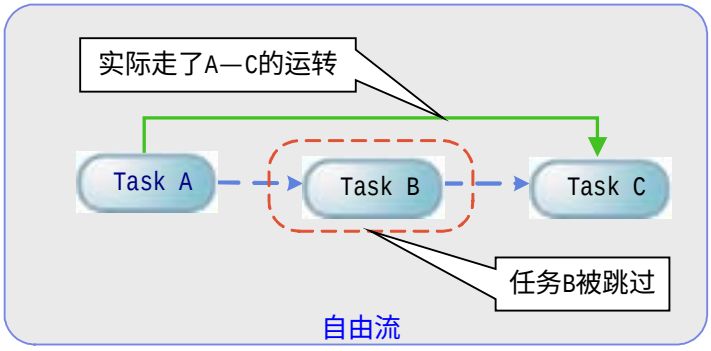



**图（5-14）**

这种应用，在现实中应用比较多，特别是电子政务中的公文流转或审批。

### 5.4.2. 自由流

自由流，表示的是一个任务执行完后，但其后续的运转不按照预定的顺序进行。而是人为地动态选择。




**图（5-15）**

如上图所示，原先的是流程是按照 A—B—C 顺序执行了。但是执行任务 A 之后，人为地让流程沿 A—C 方向运转，直接跳过了原先的任务 B。

流程的打乱，会造成很多问题：数据的完整性，流程的可控性等等，这些问题都回随之而来。所以很多工作流产品并不支持自由流的存在。不过这一现象在电子政务系统中，较为普遍。在工业化流程中，可能就较为规范，出现的可能性较低。

### 5.4.3. 委托代办

委托代办，从理论上说，不能算是一种模型。确切的说，应该是应用中一个需要处理的问题。由于在现实应用中，较为普遍的存在。就在这里顺便简要的提一下。

一个任务交给了员工 A（角色 A）处理，但是员工 A 最近出差，无法正常办理，就可以委托给员工 B（可能是角色 A，也可能是角色 B ）处理，以保证流程能够正常的进行下去。

### 5.4.4. 催办

催办，和上面的“ 委托代办 ”，以及下面即将介绍的“ 取回 ”，都属于应用中，需要解决一类问题：在执行完任务 A 到任务 B 的运转后，任务 A 设定一个催办日期，在催办日期到来的时候，向任务 B 发送催办请求，以催促任务 B 的执行。其前提是，任务 A 已经执行过，任务正在执行（有可能已经执行完）。

### 5.4.5. 取回

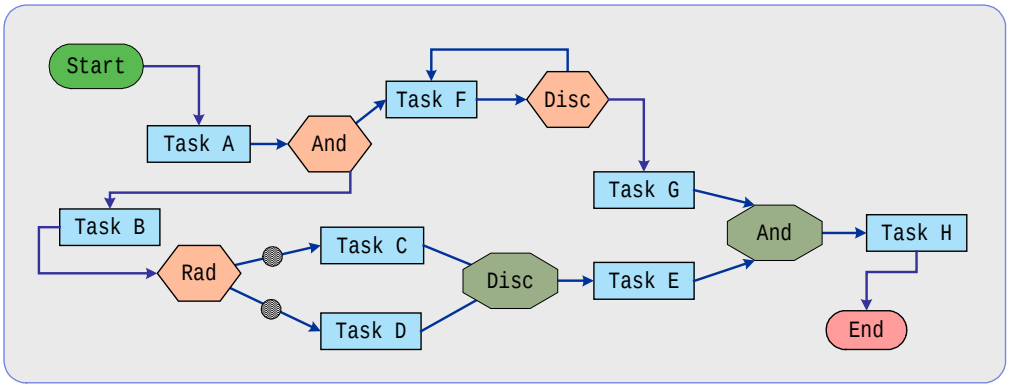
取回，在流程中也是较为通用的动作：流程由任务 A 运转到任务 B，任务 B 虽然接受了 A 所发送的请求或数据，但还没有确认执行的情况下，任务 A 有权取回，重新执行。

举例：科员起草了一份文件，交给处长审批。但处长还没有察看的情况，科员有权取回文件，重新修改。

# 6. 流程组合嵌套模型

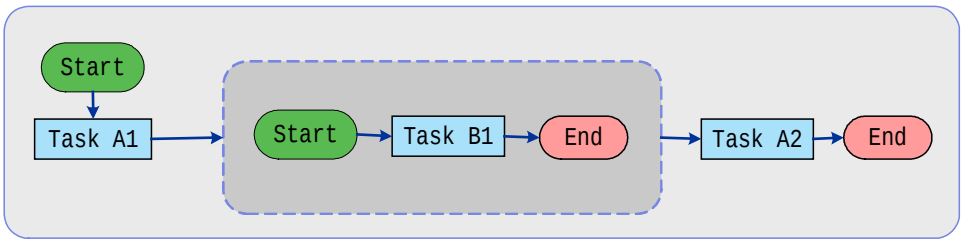
一直到现在，所说的模型，都是定位在“ 任务之间的关系 ”。不论前面的发散运转模型，还是聚合运转模型，都只是流程内部的任务关系，而不涉及到流程与流程之间的关系。

请参看下图，虽然任务很复杂，但是所有的任务都限定在同一个流程中，而且为了巩固前面的一些运转模型概念，我特意在里面包含了并行，发散，自循环，鉴别聚合，同步聚合等模型。




**图（6-1）**

让我们再来看看下面的流程，看起来比上面的流程简单，其实不是。仔细的看，其实这里面有两个流程在运行，一个主流程内嵌一个子流程。




**图（6-2）**

接下来，就来看看流程与流程之间会存在什么样的关系，存在什么可能的模型。

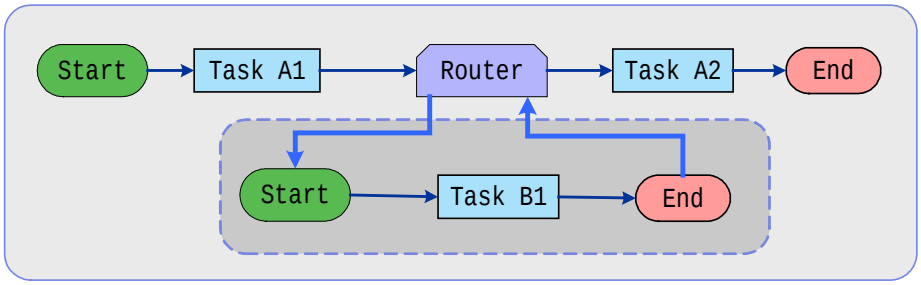
## 6.1 内嵌模型

内嵌模型刚刚已经提到了，就是在一个主流程中，内嵌了一个或多个子流程。每个子流程自身可能是可独立运转的；也有可能是主流程的辅助性子流程，不可独立运行。

由提要比较一下 xpdl 的 block 模型

### 6.1.1. 主流程等待方式

请参考下图，在主流程运行到“ Router ”位置的时候，会激活一个子流程的运行；在子流程运行完后，会重新运行到主流程的“ Router ”位置，继续主流程的运行。




图（6-3）

在这种方式下，当子流程运行的时候，主流程会暂停，等待子流程的完结。

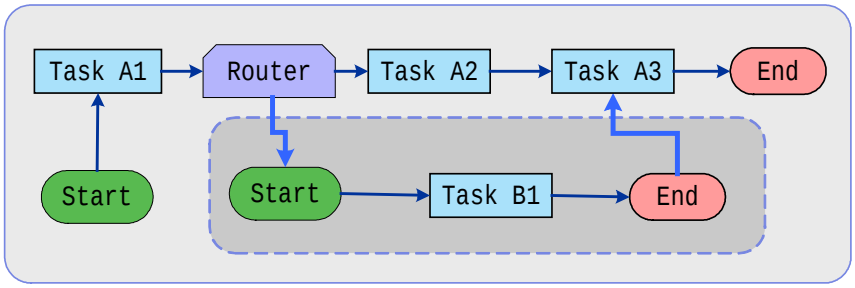
### 6.1.2. 主流程也运行方式

比较下面的图与“主流程等待方式”的图，就会发现很大的不同。

与“主流程”相同的是，当主流程运行到“Router”位置的时候，会激活一个子流程的运行。但是，激活子流程后，主流程并没有停止，而是基于按照预定的流程方向运行；同时，激活后的子流程也同样处于软转状态。

说到这里，估计很多让都会询问，那么子流程的信息什么时候返回呢？虽然在下图中，表示为子流程的信息返回到主流程的“任务 A3”。但是，依然涉及到很多问题，比如：什么时候聚合，怎么聚合的问题了；而且主流程和子流程的运行时间未必搭配恰当，有可能存在主流程首先运行到 Task A3 点，而这时候子流程还没有运行结束情况，反之亦然。

这种情况，大多数采用“同步聚合”的方式：如果有一方未到达的情况下，另一方会等待。当然，这其中可能涉及到等待超时等不良因素，这时候主流程时选择继续等待，还是发催办消息，还是继续运行，就是工作流引擎的设计问题了。

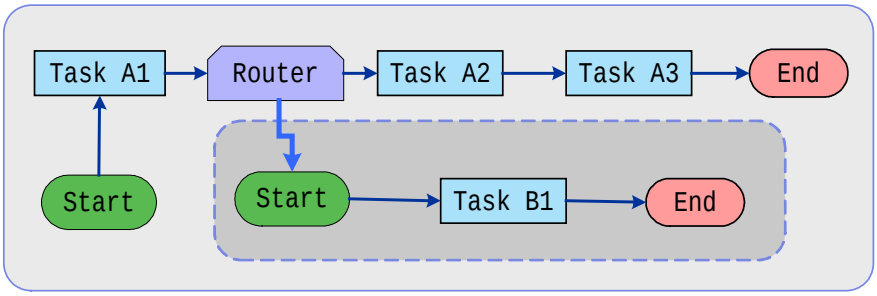



图（6-4）

## 6.2 外嵌模型

与内嵌模型不同的地方，就是外嵌的子流程，不返回主流程。在主流程激活子流程后，主流程继续运行，且不关心子流程的运行状态或运行结果。

参考下图，你会发现其与内嵌模型中的“主流程也运行方式”非常的相似，区别就是，子流程最终没有返回到主流程。




**图（6-5）**

# 7. 流程整合模型

（26 日与 colin 讨论的一些流程整合模型内容。本想在这一版加上的。不过发觉讨论的内容，还没有消化完，看来要等待下一版了。这一章与先前的版本没有大改动。）

流程整合的模型，已经超越了“ 流程运转模型 ”的概念范畴。但是作为目前“ 系统整合 ”的一个比较流行的趋势，拿到这里顺便提一下。

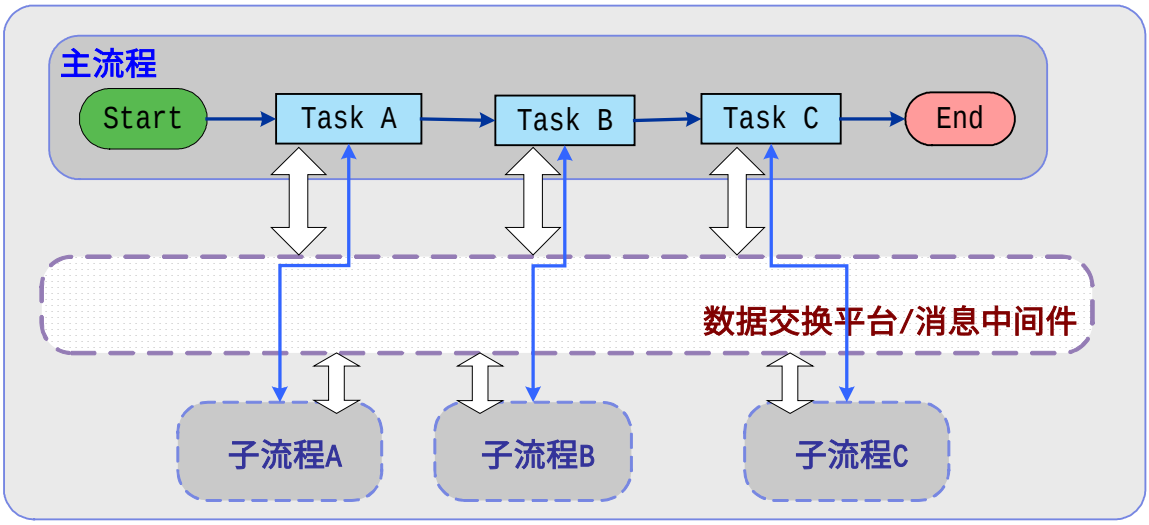
现在的业务越来越复杂，跨区域，跨部门之间信息交互方式的需要越来越明显，而且跨区域，跨部门之间业务配合也越来越多。从信息整合的发展来看，“ 面向应用的数据层整合 ”和“ 面向服务的接口层整合 ”都逐渐走向“ BMP ”模式：由中央主流程控制多个子流程（分布在不同地域或不同部门，各自独立的流程）协同运行，以达到整个业务逻辑的运行。

其实在第四章“ 流程的激活模型 ”的“ 外界消息激活 ”模型中，我已经简单提到了一些，只是不太明确。那么现在让我们来看看一个普通的“ 流程整合 ”大概是什么样子的，请参看下图。

实际的整合要比这张图上的复杂很多，也许还会有一些 JMS/WebService 等的信息交换接口，可能用到不同厂家的数据交换平台，或消息中间件等等；当然那些安全措施也必不可少了。

简单的整合模型，基本上都是采用“ 主流程控制 ”的方式：由一个主流程控制整个流程的运行，由各个子流程具体完成某项任务，并向主流程返回处理结果。主流程在确定子流程正确运行/处理完后，并得到处理完的信息后，会继续按照预定的流程路线，激活另一个子流程。

在有的流程整合设计中，主流程本身不完成任何任务（只负责运转控制）；而有的设计中，主流程本身自己也需要完成一些任务。




图（7-1）

# 8. 流程终止模型

## 8.1 按分布分

找不到什么好的文字来形容了，就用“ 分布 ”二字吧。主要是从节点的数量和形态上分析

### 8.1.1. 单结束点

这个就不要解释了吧。表明流程仅允许一个“ 结束点 ”存在。

### 8.1.2. 多结束点

参考一下前面的“ 多起点 ”，我想这个也不难理解哦

### 8.1.3. 非标准结束点

既然是一个非标准结束节点，就是说流程结束的地方，不是原先定义的流程节点。有可能在流程中间因为异常，或人为的强制终止。

## 8.2 按行为分

从行为上来看，流程的结束，基本上存在下面的三种情况：“正常”“异常”“激活新任务”。

### 8.2.1. 正常终止

按照预定的流程运转，到达结束节点后正常结束。当然上面说到的“人为强制终止”也应该算是一种正常情况。

### 8.2.2. 异常终止

因为非正常因素，系统在运行过程中产生严重的异常，造成流程非正常终止。一般针对这种可能存在的情况，流程引擎都会制定一整套处理机制，而且系统监控模块也需要报告异常的来源和起因。

### 8.2.3. 激活新任务

一个流程虽然终止了，但正因为其终止，而引发其它系统的某些流程或应用实例运行。比如在一个订单流程结束后，有可能会激活一个发货流程。再比如在电子政务应用中，涉及到的归档问题：收发文归档，以及发文转收文等等，都算这一类问题。


至此，工作流模型分析算最后完结了。

本文没有说到许多有关 XPDL 的内容。大部分的内容和模型，是根据个人经验阐述。

现实中，可能存在的模型要比这些“图形”要复杂很多，也会考虑很多因素（组织模型，安全，信息文档等等）。考虑的因素越多，涉及的流程复杂度越高，对工作流引擎的要求就越高。实际上，一个非常通用的工作流引擎是很难存在的。因为一个工作流引擎不仅需要解析预定的流程，而且还需要控制维护流程运转中的数据信息（很多业务数据是有很强的领域性），所以大多的工作流引擎都是定位在某一方向上，以解决某一类问题为主。

希望以上的文字，能够让大家了解通用的一些流程运转模型。真正在使用中，还需要大家自己去摸索，去积累了。

毕竟一家之言，难免有遗漏之出，望斧正。有兴趣一起讨论工作流的朋友，可以发邮件至 james-fly@vip.sina.com。也可以到 MSN 上，大家一起讨论，fcxiao2000@hotmail.com

# 9. 文档日志：

2003-11-20

今天决定修改/完善原先的流程模型分析。初步设想：补上“ 自由流 ”“ 委托代办模型 ”，“ 结束模型 ”。如果有可能，再加上“ 权限控制模型 ”

2003-11-24

周末竟然都没有好好抓紧。不过找 btliver 要了他曾经作的 Desige，调试通过，也算一个参考收获。

2003-11-26

zuoba 给俺发了分“ 工作流模式 ” 翻译稿。以前研究过英文版 workflow patterns，但是个别理解的有些缺陷，今天又参考了翻译稿，弥补了一些。遗憾的是，翻译稿作者没有加入自己一些看法。

2003-11-26 晚

与 colin 讨论的流程整合模型。并提到了两种额外的聚合模式。

2003-11-27

（白天）补上了特殊运转模型系列

（晚上）补上终止模型。总算完成，从头到尾又修改一遍。

# 10. 参考文档

【Workflow Patterns】
http://tmitwww.tm.tue.nl/research/patterns/

【Workflow Process Definition Interface】 WFMC
http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf

【XPDL 中文版】 张敬波（踏冰）； 韩伟（浆糊）
http://www.eworkflowing.com

【工作流模式中文版】 翻译者不详细

知识是需要沉淀的；思想也是在不断的学习、磨练中走向成熟的；而技术也是在不断的创造中开拓的。

——银狐999