第 5 章

活动图

活动图是 UML 用于对系统的动态行为建模的另一种常用工具，它描述活动的顺序，展现从一个活动到另一个活动的控制流，从而指明了系统将如何实现它的目标。活动图在本质上是一种流程图。活动图着重表现从一个活动到另一个活动的控制流，是内部处理驱动的流程。使用活动图能够演示出系统中哪些地方存在功能，以及这些功能和系统中其他组件的功能如何共同来满足前面使用用例图建模的商务需求。本章将详细介绍活动图的相关知识，并对活动图的各种符号表示及相应的语义进行逐一讨论。

本章学习内容：

- 活动图概述
- 活动图的组成元素
- 控制结点

5.1 活动图概述

在 UML 中的活动图本质上就是流程图，它显示链接在一起的高级动作，代表系统中发生的操作流程。活动图的主要作用就是用来描述工作流，其中每个活动都代表工作流中一组动作的执行。

5.1.1 定义活动图

活动图（Activity Diagram）可以用于描述系统的工作流程和并发行为，它用于展现参与行为的类所进行的各种活动的顺序关系。活动图可看作状态图的特殊形式，即把活动图中的活动看作活动状态，活动图中从一个活动到另一个活动，相当于状态图中从一个状态到另一个状态。活动图中活动的改变不需要事件触发，源活动执行完毕后自动触发转移，转到下一个活动。

活动图是一种特殊形式的状态机，用于对计算流程和工作流程建模。活动图中的状态表示计算过程中所处的各种状态，而不是普通对象的状态。活动图包含活动状态，活动状态表示过程中命令的执行或者工作流程中活动的进行。活动图也可以包含动作状态，它与活动状态类似，但是它们是原子活动，并且当它们处于活动状态时不允许发生转换。活动图还可以包含并发线程的分叉控制，并发线程表示能被系统中的不同对象和人并发执行的活动。

活动图在用例图之后提供了系统分析中对系统的进一步充分描述。活动图允许读者了解系统的执行，以及如何根据不同的条件和刺激改变执行方向。因此，活动图可以用来为用例建模工作流，更可以理解为用例图具体的细化。

在使用活动图为一个工作流建模时，一般需要经过如下步骤。

（1）识别该工作流的目标。也就是说该工作流结束时触发什么，应该实现什么目标。

（2）利用一个开始状态和一个终止状态分别描述该工作流的前置状态和后置状态。

（3）定义和识别出实现该工作流的目录所需的所有活动和状态，并按逻辑顺序将它们放置在活动图中。

（4）定义并画出活动图创建或修改的所有对象，并用对象流将这些对象连接起来。

（5）通过泳道定义谁负责执行活动图中相应的活动和状态，命名泳道，并将合适的活动和状态置于每个泳道中。

（6）用转移将活动图上的所有元素连接起来。

（7）在需要将某个工作流划分为可选流的地方放置判定框。

（8）查看活动图是否有并行的工作流。如果有，就用同步表示分叉和连接。

上述步骤中使用了活动图的各种组成元素，如活动、状态、泳道、分叉和连接等，它们将会在后面的章节中详细讲解，这里读者只需要了解即可。

活动图的优点在于它是最适合支持并行行为的，而且也是支持多线程编程的有力工具。当出现下列情况时可以使用活动图。

- **分析用例**　能直观清晰地分析用例，了解应当采用哪些动作，以及这些动作之间的依赖关系。一张完整的活动图是所有用例的集成图。
- **理解牵涉多个用例的工作流**　在不容易区分不同用例，而对整个系统的工作过程又十分清晰时，可以先构造活动图，然后用拆分技术派生用例图。
- **使用多线程应用**　采用“分层抽象，逐步细化”的原则描述多线程。

活动图的缺点也很明显，即很难清晰地描述动作与对象之间的关系，虽然可以在活动图中标识对象名，或者使用泳道定义这种关系，但仍然没有使用交互图简单直接。当出现下列情况时不适合使用活动图。

- **显示对象间的合作**　用交互图显示对象间的合作更简单、直观。
- **显示对象在生命周期内的执行情况**　活动图可以表示活动的激活条件，但不能表示一个对象的状态变换条件。因此，当要描述一个对象整个生命周期的执行情况时，应当使用状态图。

5.1.2　活动图的主要元素

在构造一个活动图时，大部分工作在于确定动作之间的控制流和对象流。除此之外，

活动图还包含了很多其他元素，本节将简要介绍其中主要元素的概念，在本章后面做详细介绍。

- **对象流** 由一个结点产生的数据，由其他结点使用。
- **控制流** 表示结点间执行的序列。
- **控制结点** 用于构建控制流和对象流，包括表示流的开始和终止结点、判断和合并，以及分叉和汇合等。
- **对象结点** 对象结点流入和流出被调用的行为，表示对象或者数据。
- **结合化的控制流结构** 如循环和分支等。
- **分区和泳道** 依照各种协作方式来组织较低层次的活动，如同现实世界中各个机构或角色各司其职。
- **可中断区间和异常** 表示控制流偏离正常执行的轨道。

活动图的核心元素是活动，两个活动的图标之间用带箭头的直线连接。在 UML 中活动表示成圆角矩形；如果一个活动引发一个活动，两个活动的图标之间用带箭头的直线连接；活动图也有起点和终点，表示法和状态图中相同；活动图中还包括分支与合并、分叉与汇合等模型元素。分支与合并的图标和状态图中判定的图标相同，而分叉与汇合则用一条加粗的线段表示。图 5-1 所示是一个人找饮料喝的活动图。

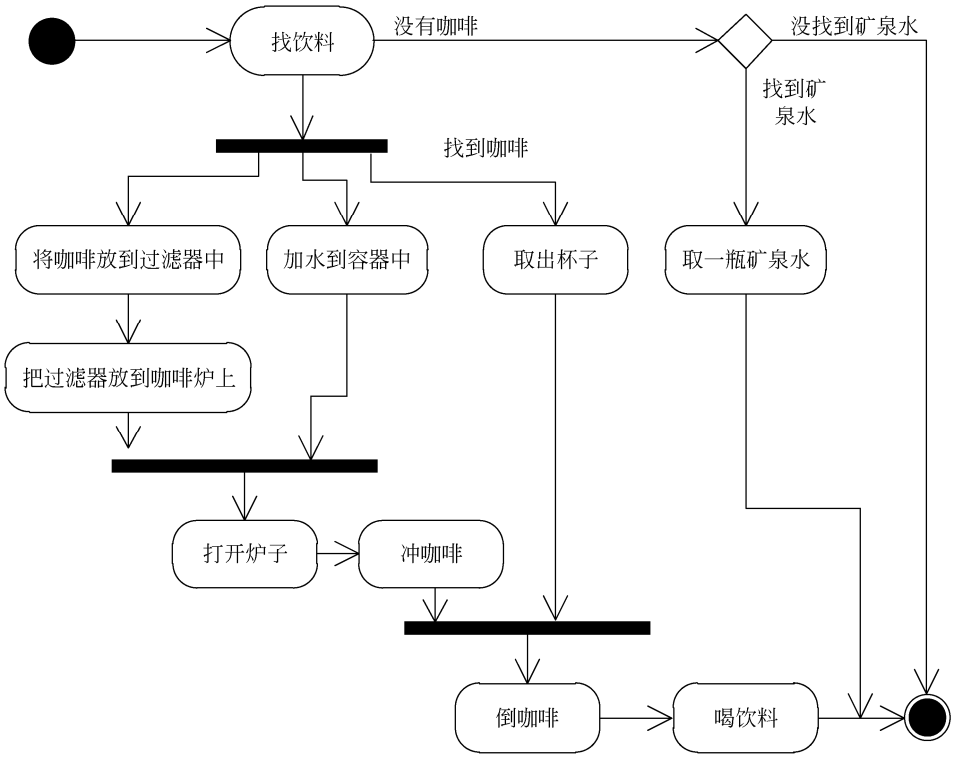

图 5-1 活动图示例

5.1.3 了解活动和动作

在构造活动图时，活动和动作是两个最重要的概念，其具体描述如下所述。

1. 活动

在活动图中每次执行活动时，都包含一系列内部动作的执行，其中每个动作可能执行 0 次或者多次。这些动作往往需要访问数据、转换或者测试数据。这些动作需按一定次序执行。

一个活动规范允许多个控制线程的并发执行和同步，以确保活动能按指定的次序执行。这种并发执行的语义容易映射到一个分布式的实现。在两个或者多个动作之间的执行次序有严格限制，所有这些限制都明确地约束了流的关系。如果两个动作之间不能直接或者间接地按确定次序执行，它们就可以并发执行。但在具体实现时并不强制并行执行，一个特定的执行引擎可能选择顺序执行或并行执行，只要能满足所有的次序约束。

一个活动通过控制流和对象流来协调其内部行为的执行。当出现如下原因时，一个行为开始执行。

- 前一个行为已执行完毕。
- 等待的对象或数据在此时变为可用。
- 流外部发生了特定事件。

一个活动图中，一组活动结点用一系列活动边连接起来。活动结点包含如下几种。

- **动作结点** 可执行算术计算、调用操作、管理对象内部数据等。
- **控制结点** 包含开始和终止结点、判断与合并等。
- **对象结点** 表示活动中所处理的一个或者一组对象，也包括活动形参结点和引脚。

活动边是一种有方向的流，可说明条件、权重等内容。活动边可根据所连接的结点种类分为如下两类。

- **控制流** 连接可执行结点和控制结点的边，简称控制边。
- **对象流** 连接对象结点的边，简称对象边。

流意味着一个结点的执行可能影响其他结点的执行，而其他结点的执行也可能影响当前结点的执行，这样的依赖关系可以表示在活动图中。

提 示

在一个活动中可以调用其他活动，就像一个操作可调用另一个，形成一个调用层次，最后到单个简单动作。在面向对象模型中，活动通常是被间接调用的，而不是直接调用的，而且方法被绑定到操作上。

2. 动作

一个活动中可以包含各种不同种类的动作，常见的动作分类如下。

- **基本功能** 如算术运算等。
- **行为调用** 如调用另一个活动或者操作。
- **通信动作** 如发送一个信号，或者等待接收某一个信号。
- **对象处理** 如对属性值或者关联值的读写。

活动中一个动作表示一个单步执行，即一个动作不能再分解，但一个动作的执行可能导致许多其他动作的执行。例如，一个动作调用一个活动，而此活动又包含了多个动作。这样，在调用动作完成之前，被调用的多个动作都要按次序执行完成。

一个动作可以有一组进入边和一组退出边，这些边可以是控制流，也可以是对象流。只有所有输入条件都满足时，动作才开始执行。动作执行完成之后，按控制流的方向启动下一个结点和动作，同时按对象流的方向输出对象表示结果，下一个结点和动作可将这些对象作为自己的输入，再启动自己的执行。

5.2 活动图的组成元素

活动图是状态图的一种特殊形式，其所有或多数状态都处于活动状态，它既可手动执行任务，也可自动执行任务。一个活动图主要包括最常用的基本组成元素和一些其他组成元素。

5.2.1 基本组成元素

除了标记符略微不同之外，活动图保留了许多传统的流程图特征，而活动图的基本元素包括活动状态、动作状态、转移、判定、开始和结束状态等。

1. 活动状态

活动也称为动作状态（Action State）是活动图的核心符号，它表示工作流过程中命令的执行或活动的进行。与等待事件发生的一般等待状态不同，活动状态用于等待计算处理工作的完成。当活动完成后，执行流程转入到活动图的下一个活动。活动状态具有以下特点。

- **原子性** 活动是原子的，它是构造活动图中的最小单位，已经无法分解为更小的部分。
- **不可中断性** 活动是不可中断的，它一旦开始运行就不能中断，一直运行到结束。
- **瞬时行为性** 活动是瞬时的行为，它所占用的处理时间极短，有时甚至可以忽略。
- **存在入转换** 活动可以有入转换，入转换可以是动作流，也可以是对象流。动作状态至少有一条出转换，这条转换以内部动作的完成为起点，与外部事件无关。
- **多出现性** 在一张活动图中，活动允许多处出现。

在 UML 中活动状态使用一个带有圆角的矩形表示，这与状态标记符相似，图 5-2 显示了活动状态。活动指示动作，因此在确定活动的名称时应该恰当地命名，选择准确描述所发生动作的词，如保存文件、打开文件，或者关闭系统等。

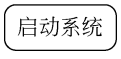

图 5-2 活动状态示例

UML 中的一个活动又可以由多个子活动构成，来完成某个复杂的功能，此时各个子活动之间的关系相同。在进行分解子活动时，有如下两种描述方法。

- **子活动图位于父活动的内部** 该方法是将子活动图放置在父活动的内部，它的优点在于建模人员可以很方便地在一个图中看出工作流的所有细节，但嵌套层次太多时，阅读该图会有一定困难。图 5-3 演示了该描述方法。

❑ **单独绘制子活动图** 使用一个活动表示子活动图的内容，在活动外重新绘制子活动图的详细内容。该方法的好处在于可简化工作流图的表示。图 5-4 演示了该描述方法。

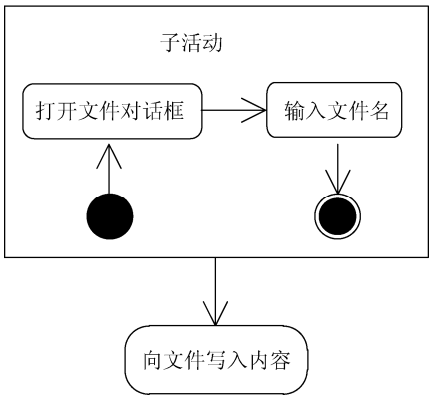

图 5-3 子活动图表示法 1

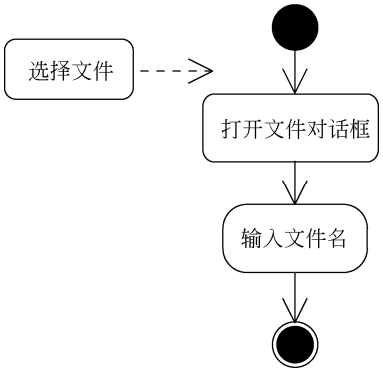

图 5-4 子活动图表示法 2

2．动作状态

活动表示某个流程中任务的执行，活动图中的活动也叫活动状态。活动图中有活动状态和动作状态，动作状态是活动状态的特例。

对象的动作状态是活动图的最小单位的构造块，是指执行原子的、不可中断的动作，并在此动作完成后通过完成转换转向另一个状态。在 UML 中动作状态使用平滑的圆角矩形表示，动作状态所表示的动作写在矩形内部。图 5-5 为一个动作状态的示例。

3．转移

一个活动图有很多动作或者活动状态，活动图通常开始于初始状态，然后自动转换到活动图的第一个动作状态，一旦该状态的动作完成后，控制就会不加延迟地转换到下一个动作状态或者活动状态。所有活动之间的转换称为转移。转移不断重复进行，直到碰到一个分支或者终止状态为止。

本章前面的活动图中已经多次用到了转移。转移是状态图中的重要组成部分，是活动图中不可缺少的内容，它指定了活动之间、状态之间或活动与状态之间的关系。转移用来显示从某种活动到另一活动或状态的控制流，它连接状态与活动、活动之间或者状态之间。转移的标记符是执行控制流方向的开放的箭头。图 5-6 显示了转移的可使用对象。

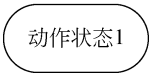

图 5-5 动作状态示例

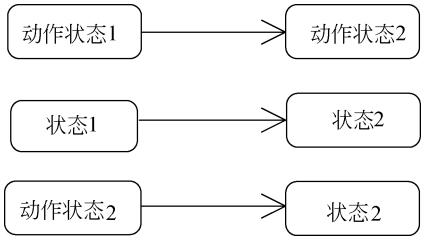

图 5-6 转移示意

有时候仅当某件确定的事情已经发生时才能使用转移，这种情况下可以将转移条件赋予转移来限制其使用。转移条件位于方括号中，放在转移箭头的附近，只有转移条件为“真”时才能到达下一个活动。图 5-7 为带有条件的转移示意图。

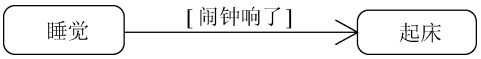

图 5-7 转移上的条件

在图 5-7 中如果要实现从活动“睡觉”转移到活动“起床”，就必须满足转移条件“闹钟响了”。只要转移条件为真时，转移才发生。在实际应用中，带有条件的转移使用非常广泛，后面的章节中将详细介绍转移条件的相关知识。

4．判定

一个活动最终总是要到达某一点，如果一个活动可能引发两个以上不同的路径，并且这些路径是互斥的，此时就需要使用判定来实现。

在 UML 中判定有两种表示方式：一种是从一个活动直接引出可能的多条路径。另一种方式是将活动转移引到一个菱形图标，然后从这个菱形的图标中再引出可能的路径。

无论用哪种方式，都必须在相关的路径附近指明标识执行该路径的条件，并且条件表达式要用中括号括起来。图 5-8 所示是判定的两种表示示例图。

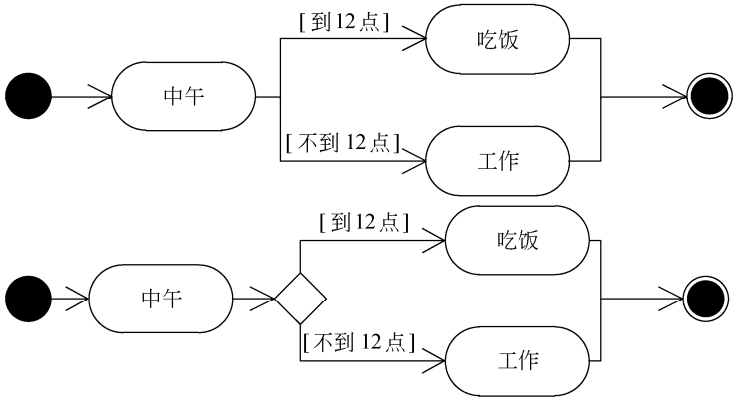

图 5-8 判定示例

5．开始和结束状态

状态通常使用一个表示系统当前状态的词或短语来标识。状态在活动图中为用户说明转折点的转移，或者用来标记工作流中以后的条件。

前面学习了活动状态和动作状态，除了它们，UML 还提供了两种特殊的状态，即开始状态和结束状态。开始状态以实心黑点表示，结束状态以带有圆圈的黑点表示，如图 5-9 所示。

图 5-9 UML 开始和结束状态

在一个活动图中只能有一个开始状态，但可以有多个结束状态。图 5-10 演示了开始状态和结束状态一对多的关系。

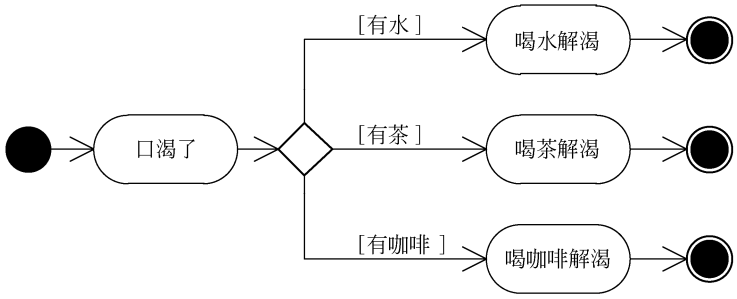

图 5–10　包含多个结束状态的活动

从图 5-10 可以看出，该活动仅包含一个开始状态，但是对应了 3 个结束状态。从开始状态进入到“口渴了”状态之后，无论转移到哪个活动都将结束控制流。

5.2.2　其他元素

除了前面讲到的活动图元素标识符外，活动图还具有其他一些元素，如事件和触发器、泳道、对象流、发送信号动作、接收事件动作，以及可中断区间等，它们也是活动图中不可缺少的标记符。这些元素与基本元素一起构建了活动图的丰富内容，综合使用它们能提高绘图技术，丰富活动图表达能力。

1．事件和触发器

事件（Event）和触发器（Trigger）的用法和控制点相似，区别是它们不是通过表达式控制工作流，而是被触发来把控制流移到对应的方向。事件非常类似于对方法的调用，它是动作发生的指示符，可以包含一个或多个参数，参数放在事件名后的括号中。图 5-11 演示了事件的使用方法。

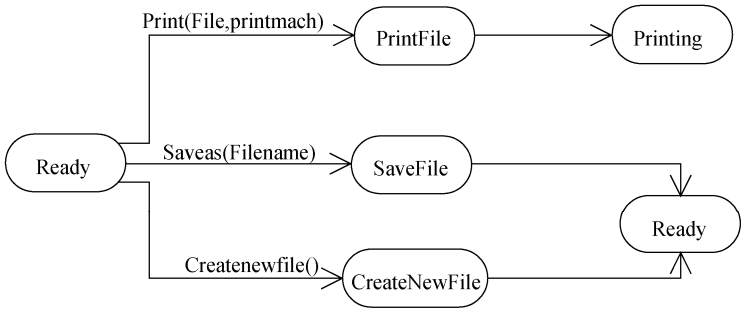

图 5–11　事件使用图

在图 5-11 中控制流根据事件进入 3 个方向，事件触发控制流离开“准备”进入相应的活动。第一个事件“Print()”具有两个参数（File 和 printmach），进行打印文件的活动；

第二个事件“Saveas()”只有一个参数（Filename），进行保存文件的活动；第三个事件“Createnewfile()”没有任何参数，进行创建新文件的活动。

2．泳道

活动图指定了某个操作时活动和动作状态的发生顺序，但是不能指定该活动或者状态属于谁，因而在概念层无法描述每个活动由谁来负责，在说明层和实现层无法描述每个活动由哪个类来完成。虽然可以在每个活动上标记出其所负责的类或者部门，但难免带出诸多麻烦。泳道的引用解决了这些问题。

泳道将活动图划分为若干组，每一组指定给负责这组活动的业务组织，即对象。在活动图中，泳道区分了负责活动的对象，它明确地表示了哪些活动是由哪些对象进行的。在包含泳道的活动图中，每个活动只能明确地属于一个泳道。每个泳道具有一个与其他泳道不同的名字。泳道间的排列次序在语义上没有重要的意义，但可能会表现现实系统里的某种关系。图 5-12 显示了泳道的标记符。

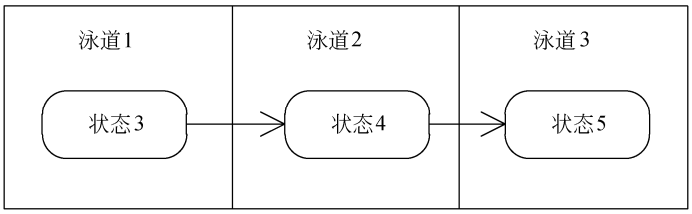

图 5–12 泳道示意图

由图 5-12 可以看出，泳道使用矩形框表示，矩形框顶部是对象名或域名，该对象或域负责泳道内的全部活动。从这里可以看出，泳道将活动图逻辑描述和交互图的职责描述结合在一起。图 5-13 演示了使用泳道的活动图。

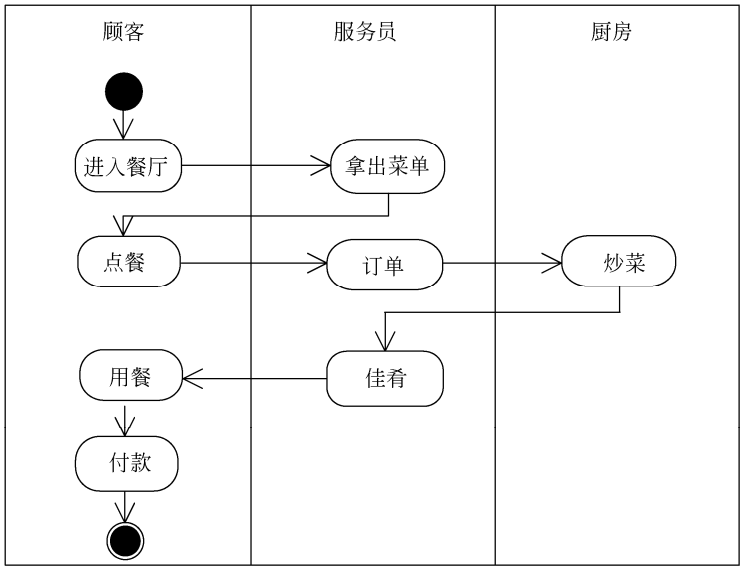

图 5–13 使用泳道的活动图

图 5-13 简单地描述了顾客用餐的活动图，其中涉及了顾客、服务员和厨房 3 个对象，各自负责自己的活动。由于在图中使用了泳道，因此读者能轻松地看出 3 个对象之间的

交互。泳道很清晰地划分出每个对象所负责的不同活动，以及泳道间活动的关系。

注　意

泳道和类不是一一对应的关系，泳道关心的是职责，一个泳道可以由一个或者多个类实现。

3．对象流

活动图描述某个对象时，可以将涉及到的对象放到活动图中，并用一个依赖将其连接到进行创建、修改和撤销的活动或状态上，对象的这种使用方法就构成了对象流。对象流是活动图中活动或状态与对象之间的依赖关系，表示活动使用对象或者活动或状态对对象的影响。

在活动图中，对象流标记符用带箭头的虚线表示。如果箭头从活动出发指向对象，则表示该活动对对象施加了一定的影响，施加的影响包括创建、修改和撤销等；如果箭头是从对象指向活动，则表示对象在执行该活动。图 5-14 所示为对象流，它连接了对象与活动。

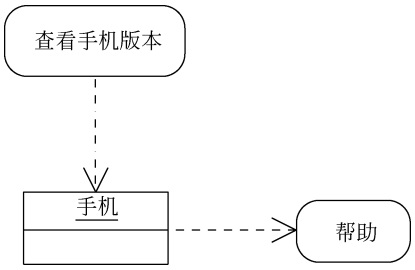

图 5–14　对象流与活动

对象流中的对象具有以下特点。

- 一个对象可以由多个活动操纵。
- 一个活动输出的对象可以作为另一个活动输入的对象。
- 在活动图中，同一个对象可以多次出现，它的每一次出现表明该对象正处于对象生存期的不同时间点。

图 5-15 是一个含有对象流的活动图，该图中对象表示图书的借阅状态，借阅者还书之前图书的状态为已借；当借阅者还了图书之后，图书的状态发生了变化，由已借状态变成了未借状态。

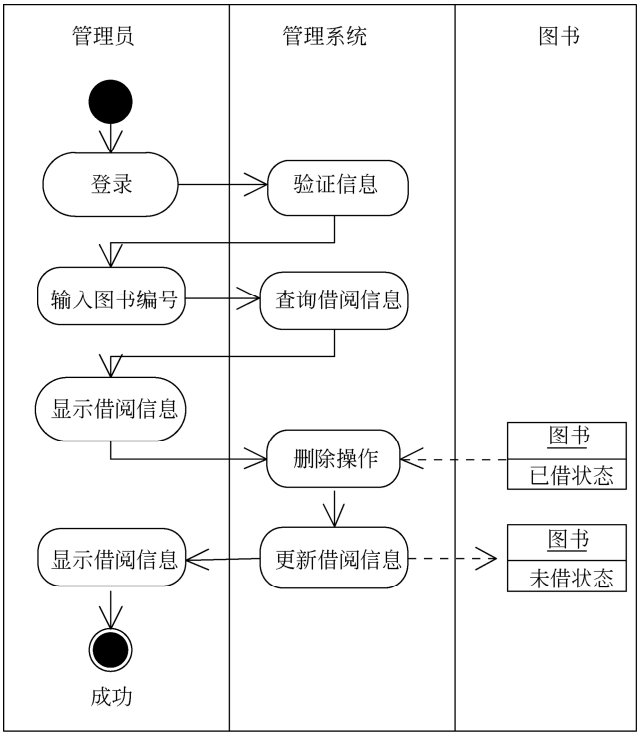

图 5–15　使用对象流的图书归还活动图

4．发送信号动作

发送信号动作是一种特殊的动作，它表示从输入信息创建一个信号实例，然后发送到目标对象。发送信号动作可能触发状态机的转换或者活动的执行。在发送信号动作时，可以

包含一组带有值的参数。由于信号是一种异步消息，所以发送方立即继续执行，所有的响应都将被忽略，并未返回给发送方。

发送信号动作表示为一个凸边矩形。图 5-16 所示为订单处理工作流中的一个片断，发送了两个信号。在创建订单之后向仓库发送一个信号，该动作是“接收订单请求”；然后创建发票，再向客户发送一个信号，该动作是“提示收货”。

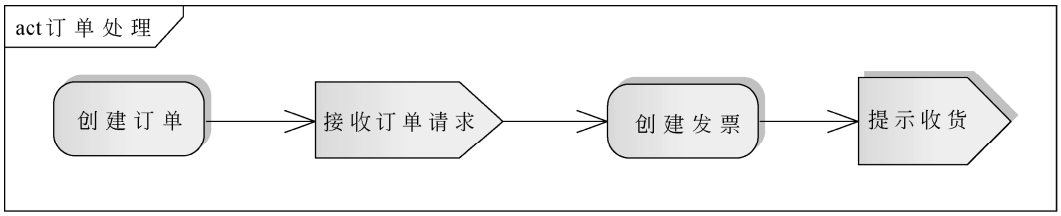

图 5-16 订单处理中的发送信号动作

在图 5-16 中仅描述了发送信号动作，而没有描述信号对象，也没有描述信号的接收方。如果需要的话，发送的一个信号对象可作为发送信号动作的一个输出对象。

5．接收事件动作

接收事件动作也是一个特殊的动作，表示等待满足特定条件的某个事件发生。

一个接收事件动作至少关联一个触发器，每个触发器都确定了一种接收的事件类型，事件的类型可以是异步调用事件、改变事件、信号事件和时间事件。一个接收事件的动作可以接收多种类型的事件。

对于调用事件，接收事件动作只能处理异步调用，而不能处理同步调用。而对于信号事件，一个触发器可确定一种信号的类型及其子类型。

接收事件动作对发生的事件进行接收和处理，所发生的事件是由拥有该动作的对象所检测的。在一个接收事件动作执行时，该对象将检查到一个事件发生，并与其中一个触发器的事件类型匹配。如果所发生的事件没有被其他动作接收，那么这个接收事件动作就执行完成了，而且输出一个值来表示这个发生的事件。如果所发生的事件没有匹配触发器所指定的任何事件类型，那么该动作就继续等待，直到匹配才能接收。

接收信号事件动作使用一个凹边矩形表示。例如，在图 5-17 中的“取消订单”就是一个接收信号事件动作，它表示等待一个信号（取消订单）发生。接收到这样一个信号之后，将调用一个取消订单的动作。图中接收事件动作没有描述输入，实际上它肯定接收到一个信号，可能来自当前活动之外，也可能来自客户。

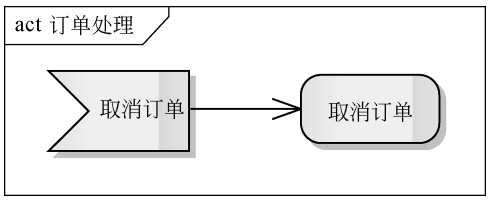

图 5-17 订单处理中的接收信号动作

一些事件接收动作可以没有输入，这也是动作的一个特点，此时当它的外层活动或者结点启动时，这个动作就启动了。该动作在接收到一个事件之后仍然保持有效。也就是说，在接收到事件而且输入一个值之后，它仍然继续等待另一个事件发生而不会终止。当外层活动或者结点终止时，此动作才终止。

例如，在图 5-18 中描述了一个发送信号和接收信息的示例。该图表示当一个订单处

理完成向客户发送一个请求支付的信息，然后等待接收来自客户的一个确认支付信号。只有当请求支付信号发送之后，才可能收到来自客户的确认支付的信号。当确认信号到达后，立即按订单发货。

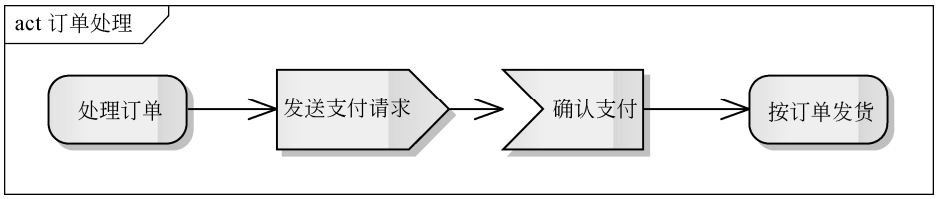

图 5-18 发送信号和接收信息

在图 5-18 中从发送信号动作到接收信号动作有一个控制流，它表示两个动作的前后顺序，但是并不能表示发送和接收的是同一个信号。

6．可中断区间

在对活动图建模时往往会出现这样的情形，即当一个活动执行在特定区间时，如果发生某种来自活动外部的事件，那么当前区间中的活动立即终止，然后转去处理所发生的事件，而且不能再回头继续执行。UML 2 中提供了可中断区间来支持这种建模。

可中断活动区间是一种特殊的活动分组，当发生某种事件时，在一个活动中把某一范围中的所有控制流都撤销。具体来说，一个可中断区间包含了多个活动结点，而且有一个或者多条流作为该区间的中断退出区间。当一个控制流沿着其中一条流退出时，该区间中的所有其他流和活动都终止。

中断流是一种特殊的活动流，对于可中断活动区间来说，每个中断流必须在区间内有一个源结点，而且中断流的目标结点必须在区间之外，且必须在同一个活动之中。

一个可中断区间往往包含有一个或者多个接收事件动作，它们表示可能导致中断的不同事件。当一个控制流在区间内退出时，该区间就中断了，此时控制流离开该区间，但是未被终止。另外，区间中的接收事件动作没有进入流，只有当一个控制流进入该区间时，该动作才被激活，以等待特定事件的发生。

一个可中断区间表示为一个虚线的圆角矩形，其中包含一组结点和控制流。一条中断流表示为一个“闪电”符号，从区间中接收事件动作指向区间外的某个结点，如图 5-19 所示。

在图 5-19 中可中断区间包含了“接收订单”“生成订单”和“按订单发货”。在“按订单发货”未完成之前，如果接收到一个“请求取消订单”事件，将离开该区间而执行“取消订单”动作，然后终止活动。这个事件来自当前活动的外部，例如来自客户的请求。实际上，当控制流进入执行可中断区间时，接收事件动作就已经激活，准备好接收特定事件了。当中断事件发生时，可能对同一个订单，一些可中断活动区间外的活动正在并发进行，但此时“按订单发货”动作不能完成导致不能同步进入“订单完成”。

7．异常

在行为建模中往往需要处理许多例外的情况。面向对象编程语言中都提供了异常处

理机制，而 UML 2 也提供了异常处理器来对异常进行建模。

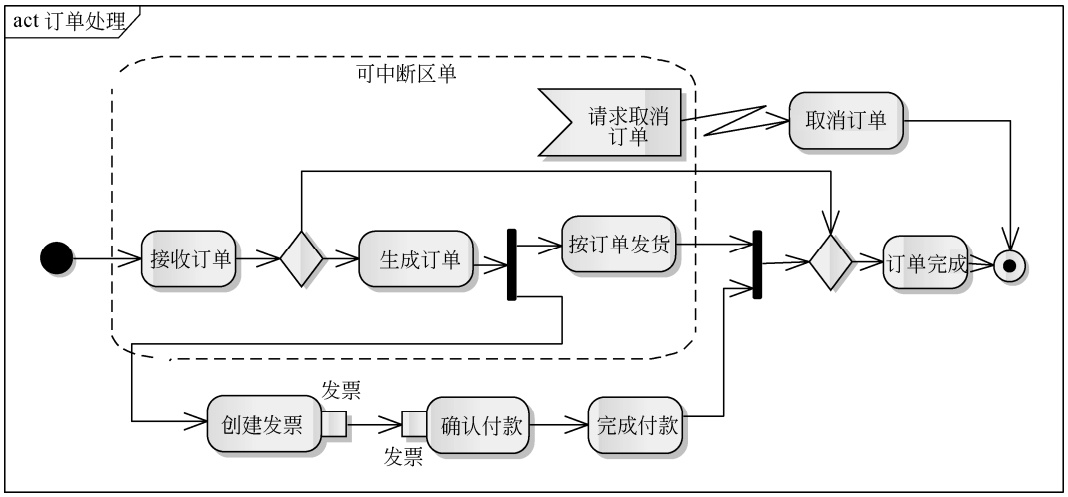

图 5–19 可中断区间示例

一个异常表示发生某种不正常的情况而停止了不正常的执行过程。在如下情况下可能发生异常。

- **可能是由于底层执行的行为错误而引起的** 例如，访问数组的下标超界，除数为零等情况。
- **可能是由一个引发异常的动作而显式引起的** UML 2 中有一种特殊的动作称为“引发异常”，它的执行将引发指定类型的异常，这类似于编程语言中使用 throw 语句抛出的异常。

为了使程序能正确地响应各种异常，就必须知道发生了哪一种异常，以及该异常的属性。将异常建模为对象就能很好地解决此问题。特定时间发生的一个异常看作一个对象，而一种异常具有相同的对象种类，反映了异常的本质特性。这就出现了专门表示异常的类型层次。例如，C++中的 Exception 类。UML 2 虽然没有提供专门的异常类型，但提供了异常处理器。

异常处理器是一种特殊的建模元素，它有一个保护结点，而且确定一个异常处理执行体和一个异常类型。当保护结点执行发生特定类型的异常时，该执行体就执行，主要包括如下几个方面。

- 一个异常处理器关联一个被保护结点，该结点可以是任何一种可执行结点。如果一个异常被传播到该结点之外，此处理器将检查是否匹配异常类型。
- 一个异常处理器有一个可执行结点作为执行体，如果该处理器与异常类型相匹配就执行。
- 一个异常处理器必须说明一种以上异常类型，表示该处理器所能捕捉的异常种类。如果所引发的异常类型是其中之一或者子类型，那么该处理器将捕捉该异常，而且开始执行其执行体中的行为。
- 一个异常处理器还需要一个对象结点作为异常输入，往往表示为该处理器的一个对象结点。当处理器捕获到一个异常时，该异常的控制流就放在此结点上，从而导致异常体的执行。

例如，图 5-20 所示为一个异常处理器的示例。其中一条异常使用“闪电”流从一个被保护结点指向一个异常器体的结点，该结点表示能捕获的一种异常类型。

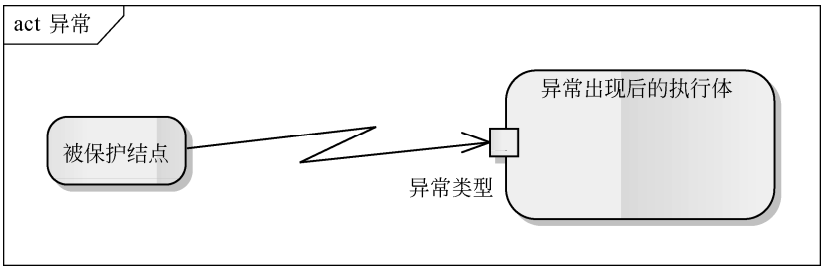

图 5-20 异常处理器示例

图 5-20 表示了被保护结点执行中如果出现异常，异常对象将沿着控制流传递给处理器。如果异常对象的类型与捕获的类型相同，则处理器的执行体就执行，而被保护结点的行为被终止。这种表示方式与编程语言中的 try catch 语言相似。

一个保护结点也可能引发多种类型的异常。例如，在图 5-21 中如果发生“异常类型 1”异常时，将被一个处理器捕获，并提供一个“结果 1”作为输出；当发生“异常类型 2”异常时，将被另一个处理器捕获，并提供一个“结果 2”作为输出。如果异常没有发生，被保护结点就正常结束，进入下面的“输出结果”结点。当发生以上两种异常之一时，“输出结果”结点将使用异常的输出作为结果执行。

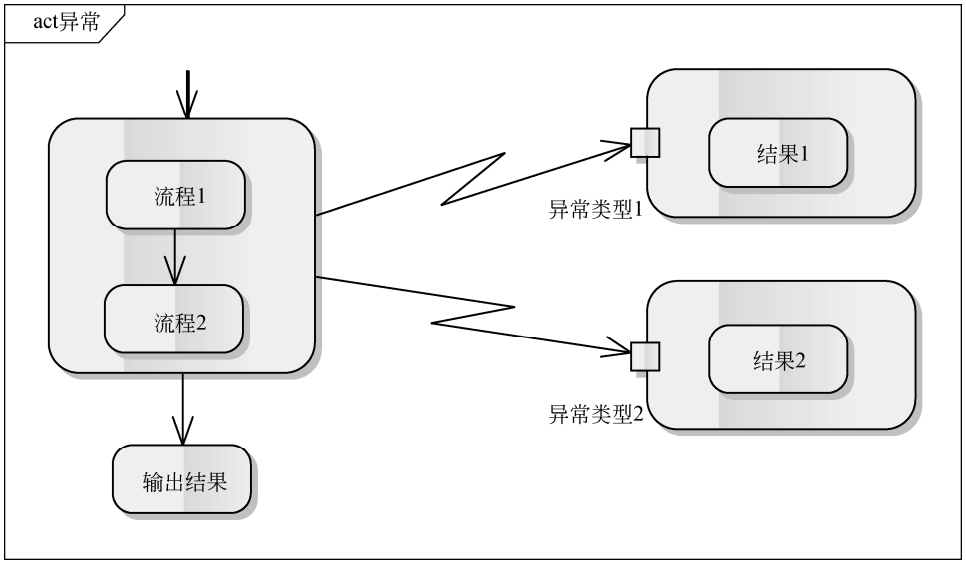

图 5-21 多异常类型示例

5.3 控制结点

控制结点是一种特殊的活动结点，用于在动作结点或对象之间协调流，包括分支、合并、分叉与汇合等。

5.3.1 分支与合并

当想根据不同条件执行不同分支的动作序列时，可以使用判定。UML 使用菱形作为判定的标记符，它除了标记判断外还能表示多条控制流的合并。本节将详细讲解有关判定进行分支和合并的相关知识。

1. 分支结点

分支可以进行简单的真/假测试，并根据测试条件使用转移到达不同的活动或状态。在活动图中可以使用判断来实现控制流的分支。图 5-22 演示了简单的两个分支测试真/假条件。

分支根据条件对控制流继续的方向做出决策，使用分支使得工作更加简洁，尤其是对于带有大量不同条件的大型活动图。所有条件控制点都从此分支，控制流转移到相应的活动或状态，这样用户就可以通过做出决策明确动作的完成。分支同样可以像判定一样完成判断条件不止一项的情况。图 5-23 是该情况下的图形表示。

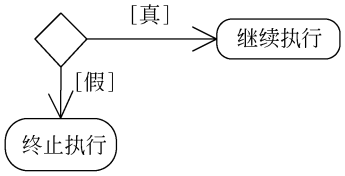

图 5-22 真/假测试条件

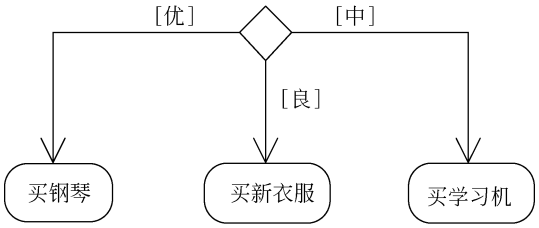

图 5-23 多分支

图 5-23 表示家长根据孩子考试成绩给予不同的奖励，条件选项分别有优、良和中，根据条件可能进入的状态有买钢琴、买新衣服或者买学习机。这种结构非常类似于大多数编程语言中的 switch 语句和 if else 组合语句的效果。

在布置易于阅读的活动图时，使用判定标记符增加了一些方便，因为它提供了彼此间的条件转移，起到节省空间的作用。图 5-24 演示了判定标记符在活动图中表示分支的使用。

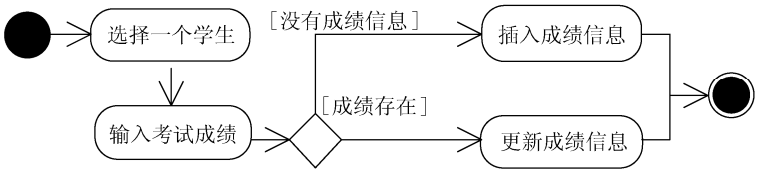

图 5-24 保存成绩活动图

图 5-24 是教师保存学生成绩的一个活动图，其中判定标记符的作用是根据条件分支控制流。在输入成绩时，根据成绩是否已经被记录来转移到不同的活动。如果成绩已经被记录，则转移到更新成绩的活动；如果没有成绩，那么将插入成绩。

除了使用判定来表示分支外，还可以使用活动来判断条件。根据活动结果可使用转移条件来建模，如图 5-25 所示。

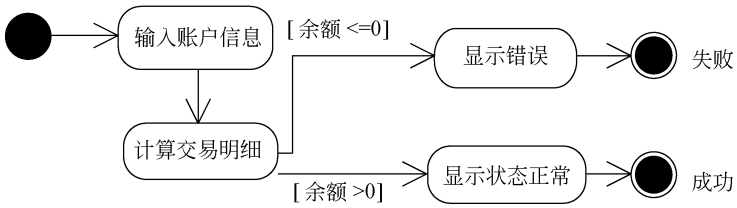

图 5-25　使用活动判断

在图 5-25 中计算账户余额的活动揭示该账户是否透支。做出判断所需的所有信息都是活动本身提供的，没有外部判断，也没有其他可用信息。为了显示由该活动导致的选择，这里仅建模离开该活动的转移，每个转移具有不同的转移条件。

2．合并结点

合并将两条路径连接到一起，合并成一条路径。前面使用判定用作分支判断，并根据条件转向不同的活动或状态。这里判定被用作合并点，用于合并不同的路径，它将多条路径重合部分建模为同一步骤序列。

在实际应用中，判定标记符不管是用作判断，还是作为合并控制流，在活动图中都使用得十分广泛，几乎每个活动图中都会用到。图 5-26 显示了活动图中使用判定标记符合并结点的情况。

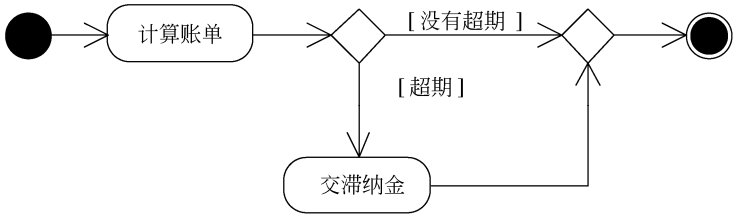

图 5-26　使用判定进行合并的活动图

图 5-26 所示的是计算信用卡账单的活动，如果交易超过规定的免息期末全额还款，将产生滞纳金。如果没有超期的交易金额，则直接进行下面的活动，直到结束状态。这里的第一个判定标记符用来表示判断，第二个判定标记符用来合并控制流。

5.3.2　分叉与汇合

前面多次使用了判定标记符，它能根据不同条件将控制流分为多个方向，也可以将多个控制流合并成一个路径。但对象在运行时可能会存在两个或多个并发运行的控制流，此时判定标记符不能完成这些功能。为了对并发的控制流建模，UML 中引入了分叉和汇合的概念。

分叉和汇合与转移密不可分。因为分叉是用于将一个控制流分为两个或多个并发运行的分支，它可以用来描述并发线程，每个分叉可以有一个输入转移和两个或多个输出转移，每个转移都可以是独立的控制流。图 5-27 是 UML 中分叉的标记符。

汇合与分叉相反，代表两个或多个并发控制流同步发生，它将两个或者多个控制流合并到一起形成一个单向控制流。每个连接可以有两个或多个输入转移和一个输出转移，如果一个控制流在其他控制流之前到达了连接，它将会等待，直到所有控制流都到达了才会向连接传递控制权。图 5-28 显示了连接标记符。这里需要说明的是：分叉和汇合的标记符都是黑粗横线，为了区分分叉和汇合，在图 5-27 和图 5-28 中分别为它们加入了转移。

在活动图中，使用分叉和连接来描述并行的行为。即每当在活动图上出现一个分叉时，就有一个对应的汇合将从该分叉分出去的分支合并在一起。图 5-29 是一个使用了分叉和汇合的活动图。

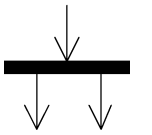

图 5-27 分叉标记符

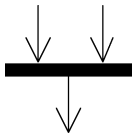

图 5-28 汇合标记符

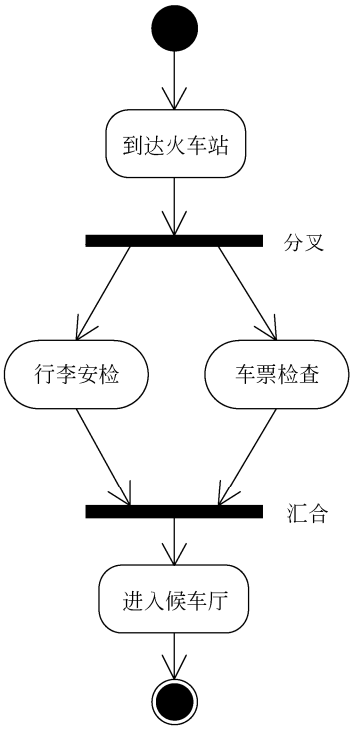

图 5-29 分叉和汇合

图 5-29 中用了一个分叉和一个汇合描述进入火车站候车厅前的活动图。首先到达火车站，此时要求分别检查随身携带的行李和乘车车票，这两项检查是同时进行的，当两个活动都完成时同时到达下一个状态后，才能进行进入候车厅动作。

注　意

动作同步发生并不意味着它们一定同时完成。事实上，一项任务很可能在另一项之前完成。不过，结合点会防止有任何流在所有进来的工作流完成以前继续通过结合点，使得只有所有的工作流完成以后，系统才会继续执行后续动作。

5.4 实例：创建 BBS 论坛活动图

在本节前面详细介绍了活动图中各个元素的表示方式，并给出了简单示例。本节将以 BBS 论坛系统为例进行分析，并逐步实现其活动图，让读者了解绘制活动图的基本步骤和技术要领。

5.4.1 建模步骤

在系统建模过程上，活动图能够被附加到任何建模元素以描述其行为，这些元素包含用例、类、接口、组件、结点和操作等。现实中的软件系统一般都包含很多类，以及复杂的业务过程，这里的业务过程指工作流。系统分析可以用活动图来对这些工作流建模，用以重点描述这些工作流；也可以用活动图对操作建模，用以重点描述系统的流程。

无论在建模过程中活动图的重点是什么，它都是用活动流来描述系统参与者和系统之间的关系。建模活动图也是一个反复的过程，活动图具有复杂的动作和工作流，检查修改活动图时也许会修改整个工程。所以有条理地建模会避免许多错误，从而提高建模效率。建模活动图时可以按照以下步骤来进行。

（1）为工作流建立一个焦点，确定活动图所关注的业务流程。由于系统较大，不可能在一张图中显示出系统中所有的控制流。通常，一个活动图只用于描述一个业务流程。

（2）确定该业务中的业务对象。选择结束全部工作流中的一部分有高层职责的业务对象，并为每个重要的业务对象创建一条泳道。

（3）确定该工作流的开始状态和结束状态。识别工作流初始结点的前置条件和活动结束的后置条件，确定该工作流的边界，这可有效地实现对工作流的边界进行建模。

（4）从该工作流的开始状态开始，说明随时间发生的动作和活动，并在活动图中把它们表示成活动状态或者动作状态。

（5）将复杂的活动或多次出现的活动集合归到一个活动状态结点，并对每个这样的活动状态提供一个可展开的单独的活动来表示它们。

（6）找出连接这些活动和动作状态结点的转换，从工作流的顺序开始，考虑分支，再考虑分叉和汇合。

（7）如果工作流中涉及重要的对象，则也可以将它们加入到活动图中。如果需要描述对象流的状态变化，则需要显示其变化的值和状态。

5.4.2 创建活动图

活动图能够显示出系统中哪些地方存在功能，以及这些功能和系统中的其他功能如何共同满足前面使用用例图建模的商务需求。前台功能根据用户的身份使用活动图分别建模，图 5-30 为会员用户的活动图。

从图 5-30 中可以看出，会员用户输入登录信息成功登录系统后，可以进入操作功能界面；登录失败则会重新登录。进入会员管理操作界面后会显示会员可以进行的操作，如发表帖子、回复或浏览帖子、添加好友等，这些操作是并列的，所以会员选择一项操

作完成后会退出系统。

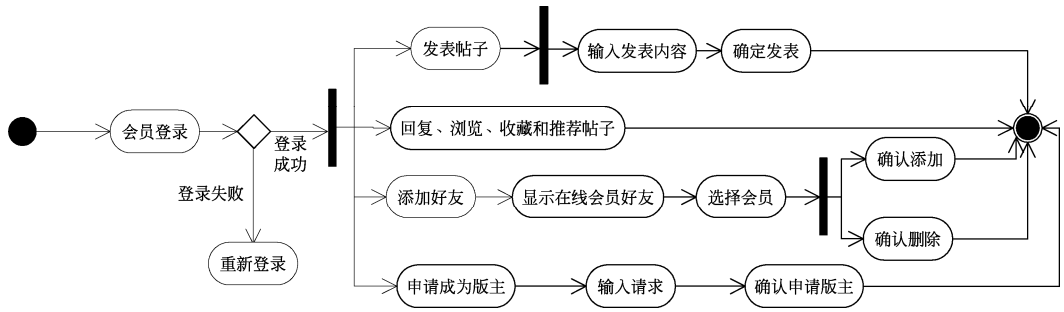

图 5-30 论坛系统会员用户的活动图

如图 5-31 为普通用户的活动图。

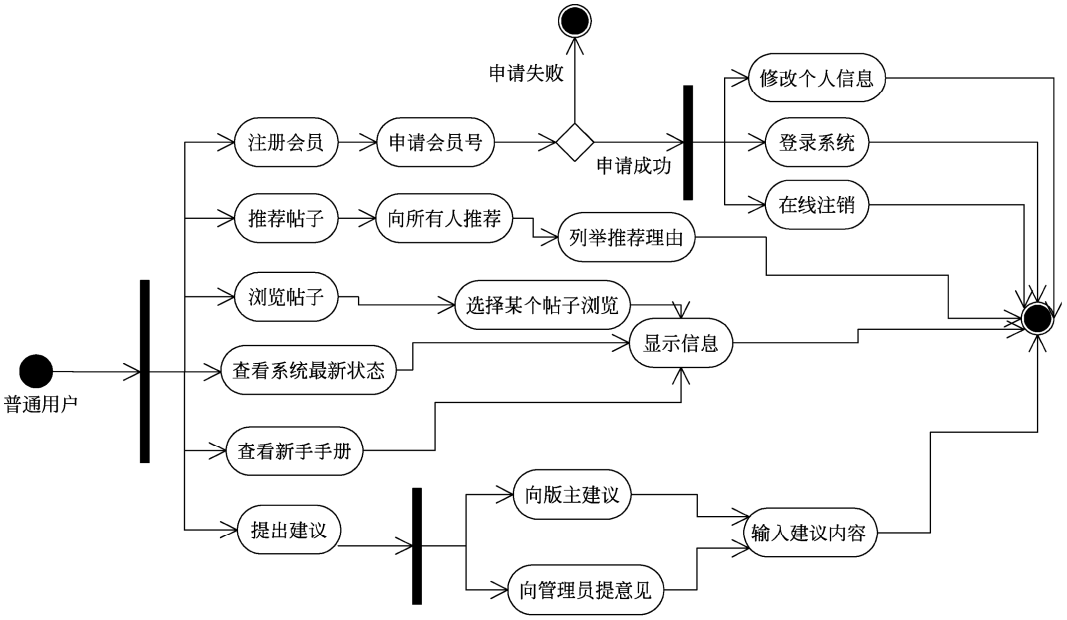

图 5-31 普通用户的活动图

从图 5-31 中可以看出，普通用户注册成为会员时，如果申请失败则会直接退出系统；注册成功后可以进入界面进行简单的操作，如修改个人信息、登录系统和在线注销等。如果普通用户不注册而直接进入系统时，也可以进行推荐帖子、浏览帖子、查看新手手册和提出建议等操作，由于这些操作是并列的，所以普通用户操作某一项完成后会直接退出系统。

5.5 思考与练习

一、填空题

1. UML 中活动图的核心元素是_____，它使用圆角矩形表示。

2. 活动图中的活动结点有 3 种类型，其中_____结点可以包含开始状态。

3．在一个活动图中可以有一个开始状态，有_____个结束状态。

4．在活动图中使用_____来描述并行的行为。

5．一个异常处理器包含一个异常处理执行体和一个_____。

二、选择题

1．下列不属于活动图组成元素的是_____。

A．开始状态

B．消息调用

C．泳道

D．判定

2．活动图中的动作不可以执行如下哪个动作？_____

A．创建实例

B．执行加法运算

C．发送一个信号

D．关联属性值

3．下列关于活动的描述不正确的是_____。

A．在一张活动图中活动允许多处出现

B．活动是构造活动图中的最小单位

C．活动的入转换可以是动作流，也可以是对象流

D．活动使用实心圆表示

4．下列关于判定的描述不正确的是_____。

A．判定中的分支路径是并行的

B．判定中的分支路径是互斥的

C．判定使用菱形表示

D．判定的条件用中括号括起来

5．在活动图中，_____明确地表示了哪些活动是由哪些对象进行的。

A．汇合

B．对象流

C．泳道

D．转移

6．_____表示等待满足特定条件的某个事件发生。

A．接收事件动作

B．发送信号动作

C．调用动作

D．触发器

三、问答题

1．简述活动图的概念和用途。

2．简要介绍分叉和汇合。

3．说明活动图中使用泳道的益处。

4．简要概括建模活动图的步骤。

5．简述使用发送信号动作和接收事件动作的情况。