第22章
进程和线程

本章内容：

- 主动对象、进程和线程
- 对多控制流的建模
- 对进程间的通信建模
- 建立线程安全的抽象

现实的世界不仅是一个严厉无情的地方，而且还是一个非常忙碌的地方。事件可能发生，而且事情可能都在同一时间发生。所以，当你对现实世界的一个系统建模时，你必须考虑它的进程视图，包括形成系统的并发与同步机制的线程和进程。【在第2章中讨论软件体系结构的语境中的进程视图。】

在UML中，可以将每一个独立的控制流建模为一个主动对象，它代表一个能够启动控制活动的进程或线程。进程是一个能与其他进程并发执行的重量级的流；而线程是一个能与同一进程中的其他线程并发执行的轻量级的流。

建立抽象以使它们在多控制流同时存在的情况下安全地工作，这是困难的。尤其是，你不得不考虑比顺序系统更复杂的通信和同步方法。你还不得不非常小心，对进程视图的工程化程度既不能过度（太多的并发流会使你的系统产生颠簸），也不能过低（并发不足则不能优化系统的吞吐）。

22.1 入门

一条狗生活在它的狗窝中，世界对它来说是一个相当简单和有顺序的地方。它整日里吃东西、睡觉、追逐一只猫，吃更多的东西，并梦想着和猫游戏。因为只有狗才需要从狗窝的门进进出出，所以对狗来说，使用狗窝来睡觉或遮风避雨从来不成问题。这里没有任何对资源的争用。【在第1章中讨论对狗窝和高大的建筑物建模。】

在一个家庭的生活以及所住的房子中，世界就不那么简单了。实际上，家庭的每个成员都有他或她自己的生活，并且要与家庭的其他成员打交道（如一起进餐、看电视、玩游戏、打扫卫生）。家庭成员们分享某些资源。孩子们共享一间卧室；整个家庭共享一部电话或一台计算机。家庭成员们还共同承担家务。爸爸洗衣服和去杂货店购物，妈妈管理帐务和在庭院工作；孩子们帮助打扫和烹饪。这些共享资源间的争用以及这些独立家务之间的协调问题是富于挑战性的。当每个人都准备去上学或上班时，共享一个浴室就是一个问题；如果爸爸不先去购物，那么晚餐就不能做好。

在高楼和它的房客的生活中，世界真正变得复杂起来。数百的（如果不是成千的）人可能在同一座建筑物中工作，每个人都遵循他或她的日程。大家都必须通过一些有限的入口，都必须乘坐同一部电梯，共享同一个加热、制冷、水、电、卫生设备和停车场。如果他们想一起最优化地工作，那他们就必须保持通信，并使他们的交互适当地同步。

在UML中，每个独立的控制流被建模为一个主动对象。一个主动对象是一个可以启动控制活动的进程或线程。像每种对象一样，主动对象是类的一个实例，但在这种情况下，它是主动类的实例。也像每种对象一样，主动对象通过传送消息进行互相之间的通信，然而，消息的传送必须用某些并发语义来加以扩充，以帮助在相互独立的流之间对交互进行同步。【在第13章中讨论对象。】

在软件中，许多编程语言直接支持主动对象的概念。Java、Smalltalk 和Ada都有内置的并发。C++通过建立在宿主操作系统的并发机制上的各种库来支持并发。使用 UML来可视化、详述、构造、文档化这些抽象是非常重要的，因为不这样做，研究并发、通信和同步等问题几乎是不可能的。

UML提供了对主动类的图形化表示，如图 22-1所示。主动类是类的一种，所以具有类的所有通常部分，如类名、属性和操作。主动类经常接收信号，通常把这些信号列在一个附加栏中。【在第4章和第9章中讨论类；在第20章中讨论信号。】

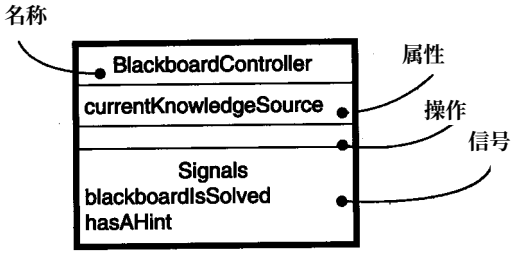

图22-1 主动类

22.2 术语和概念

主动对象（*active object*）是拥有一个进程或线程，并能够启动控制活动的一个对象。主动类（*active class*）是一个其实例为主动对象的类。进程（*process*）是一个能同其他的进程并发执行的重量级的流；而线程（*thread*）是一个能与同一进程中的其他线程并发执行的轻量级的流。从图形上，一个主动类用一个粗线矩形框来表示。进程和线程用构造型化的主动类来表示（而且，也在交互图中作为序列出现）。【在第18章中讨论交互图。】

1. 控制流

在一个纯顺序的系统中，只有一个控制流。这意味着在一个时间点有且仅有一件事情在发生。当一个顺序程序开始时，控制处于程序的开头，操作一个接一个地被执行。即使在系统外的参与者之间有并发的事情发生，顺序程序在一个时间点上也只执行一个事件，任何并发的外

部事件都要排队或者被丢弃。【在第16章中讨论参与者。】

这就是它为什么被称作一个控制流的原因。如果你想跟踪一个顺序程序的执行过程，你就会看到流的执行轨迹是顺序地从一个程序语句到另一个程序语句。你可能看到动作的分支、循环和跳转，并且如果有任何递归或迭代，你会看到流转一个圈又转回到自身。然而在一个顺序的系统中，将只有一个单个的执行流。【在第15章中讨论动作。】

在一个并发系统中，存在着多个控制流——也就是说，在一个时间点上有多件事情发生。在一个并发系统中，有多个同时发生的控制流，每个都以一个独立的进程或线程的头部为根。如果在并发系统运行时给它拍一个快照，从逻辑上，你将看到多个执行点。

在UML中，用主动类来表示进程或线程，该进程或线程是一个独立的控制流的根，并且与所有对等的控制流并行。【在第26章中讨论节点。】

注释 你可以用下面三种方法之一获得真正的并发：一是把主动对象分布到多个节点；二是把主动对象放在具有多个处理器的节点上；三是以上两种方法的结合。

2. 类和事件

主动类仍然是类，尽管它具有很特殊的性质。一个主动类代表一个独立的控制流，而普通的类不能体现这样的流。与主动类相反，普通类隐含地被称作被动的，因为它们不能独立地启动控制活动。【在第4章和第9章中讨论类。】

主动类用于对进程或线程的公共家族建模。在技术术语中，这意味着一个主动对象，即一个主动类的一个实例，使进程或线程具体化（或者说它是进程或线程的一种表示）。通过用主动对象对并发系统进行建模，要为每个独立的控制流起一个名称。当创建一个主动对象时，其相关的控制流就开始；当撤销这个主动对象时，其相关的控制流就终止。【在第13章中讨论对象。】

主动类拥有与所有其他类相同的特性。主动类可以有实例。主动类可以有属性和操作。主动类也可以参与到依赖、泛化和关联（包括聚合）关系中。主动类可以使用 UML的任何扩展机制，包括构造型、标记值和约束。主动类可以是接口的实现。主动类可以通过协作来实现，并且，一个主动类的行为可以通过使用状态机来说明。【在第4章中讨论属性和操作；在第 4章和第10章中讨论关系；在第6章中讨论扩展机制；在第 11章中讨论接口。】

在你的模型图中，主动对象可以出现在被动对象出现的任何地方。你可以用交互图（包括顺序图和协作图）对主动对象和被动对象的协作建模。一个主动对象在状态机中可以作为一个事件的目标而出现。【在第21章中讨论状态机。】

说到状态机，被动对象和主动对象均可发送和接收信号事件及调用事件。【在第20章中讨论事件。】

3. 标准元素

UML中的所有扩展机制都可以应用于主动类。大多数情况下，你将使用标记值来扩充主动类的特性，例如说明这个主动类的调度策略。【在第6章中讨论UML的扩展机制。】

UML定义了两个应用于主动类的标准构造型。【在附录B中概述UML的标准元素。】

1) 进程（process） 说明一个能与其他进程并发执行的重量级的流；

2) 线程（thread） 说明一个能与同一进程中的其他线程并发执行的轻量级的流。

进程和线程之间的区别来自一个控制流可以由这个对象所在节点的操作系统以两种不同的

方式管理。【在第26章中讨论节点。】

进程是重量级的，这意味着它是操作系统自己知道的、并在一个独立的地址空间中运行的事物。在大多数操作系统（如 Windows和Unix）中，每个程序都在它自己的地址空间里作为一个进程运行。一般情况下，一个节点上的所有进程是互相平等的，竞争这个节点上提供的所有相同的资源。进程从不相互嵌套。如果在一个节点上有多个处理器，那么在这个节点上实现真正地并发是可能的。如果这个节点只有一个处理器，那么这只是真正并发的错觉，它是由基础的操作系统来完成的。

线程是轻量级的。操作系统本身可以知道它。大多数情况下，它隐藏在一个重量级的进程内部，并在该进程的地址空间内部运行。例如在 Java中，一个线程是类Thread的一个子类。位于一个进程的语境中的所有线程相互之间都是对等的，竞争该进程内部提供的相同资源。线程从不相互嵌套。总之，线程之间的真正并发仅仅是一种错觉，因为只有进程（而不是线程）才由一个节点上的操作系统进行调度。

4. 通信

当对象相互协作时，它们通过从一个对象到另一个对象发送消息来进行交互。在一个既有主动对象又有被动对象的系统中，有四种你必须考虑的可能的交互组合。【在第15章中讨论交互。】

第一种是，一个消息的传送是从一个被动对象到另一个被动对象。假定在一个时间点只有一个控制流通过这些对象，这样的一个交互其实就是对一个操作的简单引用。

第二种是，一个消息的传送是从一个主动对象到另一个主动对象。当这种情况发生时，就有了进程间的通信，并且有两种可能的通信类型。一种类型是，一个主动对象可能同步地调用另一个主动对象的操作。这种通信有多种语义，即：调用者调用操作；调用者等待接收者接收这个调用；操作被调用；一个返回对象（如果有）被传送回调用者；然后两者分别继续它们的各自独立的路径。在这种调用的过程中，两个控制流是前后相接的。另一种类型是，一个主动对象可能异步地发送一个信号或调用另一个对象的一个操作。这种通信具有邮箱的语义，这意味着调用者发送信号或调用操作，然后就继续它自己的独立的路径。与此同时，接收者在准备好时（用插入事件或调用队列）才接收信号或调用，完成后接着向下执行它的路径。之所以称这种通信为邮箱，是因为这两个对象不是同步的，而是一个对象给另一个对象留下一个消息后就离开了。【在第20章中讨论信号事件和调用事件。】

在UML中，用一个完整的箭头来表示一个同步消息，用一个半箭头来表示一个异步消息，如图22-2所示。

第三种是，一个消息的传送是从一个主动对象到一个被动对象。如果在一个时间点上，有多于一个主动对象通过一个被动对象传送它们的控制流，那么困难就产生了。在这种情况下，你将不得不非常仔细地对这两个流的同步问题建模，这将在下一节讨论。

第四种是，一个消息的传送是从一个被动对象到一个主动对象。初看起来，这显得不合法，但如果你记住每个控制流都是以某些主动对象为根，你就会理解从一个被动对象传送一个消息到一个主动对象与从一个主动对象传送一个消息到一个主动对象具有相同的语义。【在第6章中讨论约束。】

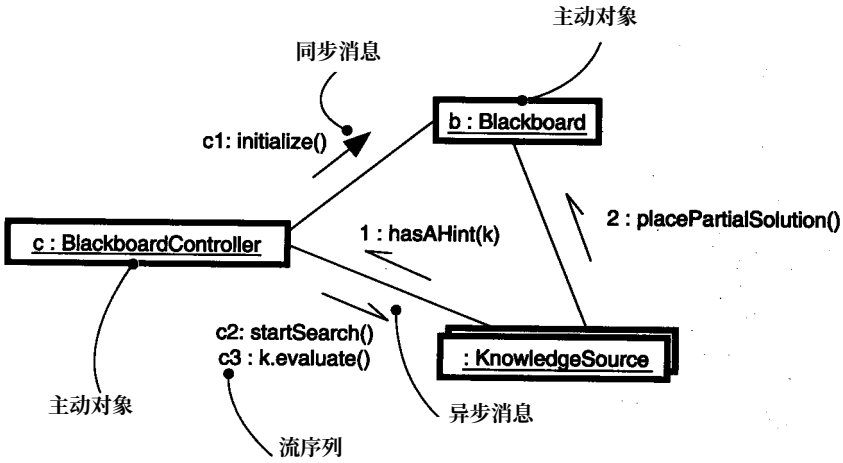

图22-2 通信

注释 通过使用约束可以对同步或异步消息传送的变体建模。例如，对可在 Ada中见到的那种阻塞聚点问题建模时，可以使用带有一个约束（如 {wait = 0}）的同步消息来实现，表示调用者不等待接收者的响应。类似地，使用一个约束（如 {wait = 1 ms}）来对一个等待时间建模，表示调用者等待接收者接收消息的时间不超过 1毫秒。

5. 同步

让我们想象一下，迂回穿行于一个并发系统的多个控制流。当一个流通过一个操作时，我们称在给定的时间中，控制焦点在这个操作中。如果这个操作是为某些类定义的，那我们也可以说在给定的时间内控制焦点在这个类的一个特定实例中。在一个操作中可以有多个控制流（所以在一个对象中也是如此），并且在不同的操作中也可以有不同的控制流（但仍将导致在一个对象中有多个控制流）。

当一个对象在同一时刻有多个控制流时，就出现了问题。如果你不仔细，多个流中的任何流都可能和另一个流冲突，破坏对象的状态。这是典型的互斥问题。对这个问题处理的失败可能会产生各种竞争条件和冲突，导致并发系统以一种神秘的和不可重复的方式失败。

在面向对象系统中，解决这种问题的关键是把一个对象当做临界区。有三种可供选择的方法，每一种都包含将某些特定的同步特性附加到在类中定义的操作上。在 UML中，你可以对所有三种方法建模。

1）顺序的（sequential） 调用者必须在对象外部协调，使得在一个时刻这个对象内仅有一个流。当有多个控制流出现时，就无法保证该对象的语义和完整性。

2）监护的（guarded） 当有多个控制流出现时，该对象的语义和完整性是通过把对该对象的监护操作的所有调用进行顺序化来保证的。其效果是，在一个时刻该对象恰好只有一个操作能被调用，使之简化为顺序的语义。

3）并发的（concurrent） 当有多个控制流出现时，该对象的语义和完整性是通过把操作当做原子来处理而得到保证的。

某些编程语言直接支持这些构造。如 Java，它具有 synchronized 特性，这个特性等价于 UML 中的 concurrent 特性。在每种支持并发的语言中，你可以用信号量来构造它们，从而为所有这些特性提供支持。

如图 22-3 所示，你可以将这些特性附加到一个操作上，在 UML 中你可以使用约束符号来表示。【在第 6 章中讨论约束。】

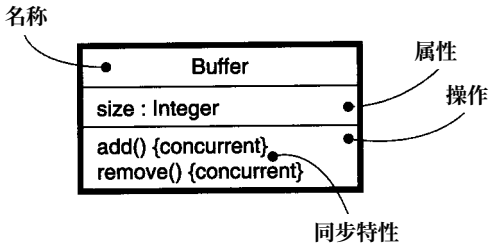

图 22-3 同步

注释 使用约束可以对这些同步图元的变体建模。例如，可以允许有多个读者，但只允许有单独一个写入者来修改 concurrent 特性。

6. 进程视图

主动对象在可视化、详述、构造和文档化一个系统的进程视图中扮演着主要的角色。系统的进程视图包括形成系统的并发和同步机制的线程和进程。这种视图主要表示系统的性能、可扩展性和吞吐量。使用 UML，可用与设计视图同样的图，即类图、交互图、活动图和状态图，来捕获这个视图的静态和动态方面，但是要把重点放在表示这些线程和进程的主动类上。【在第 2 章中讨论软件体系结构语境中的进程视图。】

22.3 普通建模技术

22.3.1 对多控制流建模

建造含有多个控制流的系统是困难的。你不仅要决定如何在并发的主动对象之间划分工作，而且即使你做好了这些，你还得为系统中主动和被动对象之间的通讯与并发设计正确的机制，以确保它们在多控制流出现时可以正确地工作。因此，它有助于可视化这些流相互之间的交互方式。在 UML 中，你可以应用包括主动类和对象的类图（捕捉它们的静态语义）和交互图（捕捉它们的动态语义）来做此事。【在第 28 章中讨论机制；在第 8 章中讨论类图；在第 18 章中讨论交互图。】

对多控制流建模，要遵循如下的策略：

- 识别并发动作出现的机会，把每个流具体化为一个主动类。将主动对象的共同集合泛化为一个主动类。注意不要引入太多的并发，以免使系统的进程视图过度工程化。【在第 2 章中讨

论进程视图；在第4章和第9章中讨论类；在第5章和第10章中讨论关系。】

- 考虑这些主动类之间职责的均衡分布，然后检查每个与之静态协作的其他主动类和被动类。保证每个主动类与相关的邻居类之间既是紧密的内聚，又是松散的耦合，并且每个主动类都具有合理的属性、操作和信号集合。
- 在类图中捕捉这些静态决策，显式地突出表示每个主动类。
- 考虑每一组类相互之间是如何动态协作的。在交互图中捕捉这些决策。显式地表示作为这些流的根的主动对象。用主动对象的名称来标识每个相关的序列。
- 密切注意这些主动对象之间的通信。视情况而定，应用同步和异步消息。
- 密切注意这些主动对象以及与它们协作的被动对象之间的同步。恰当地运用顺序、监护或并发操作语义。

例如，图22-4显示的是一个商务系统的进程视图的一部分。你可以发现3个对象并发地把信息放入系统中，这3个对象分别是：StockTicker、IndexWatcher和CNNNewsFeed（名称分别为s、i和c）。这些对象中的两个（s和i）与它们自己的Analyst实例（a1和a2）通信。至少对于这个模型来说，可以简单地假定这个Analyst一个时刻只有一个控制流在它的实例中是活动的。然而，Analyst的两个实例都同时与AlertManager（名称为m）通信。所以，m必须被设计为在多个控制流出现时能维持它的语义。 m和c同时与t（一个TradingManager）进行通信。每个流都被给定一个由拥有它的控制流来区分的顺序号。

注释 像这样的交互图，对下述问题的可视化很有帮助——两个控制流可能交叉，因此需要特别注意通信和同步问题。允许工具提供更多的、有区别的可视化提示，例如以清楚的方式为每个流加上颜色，以示区分。

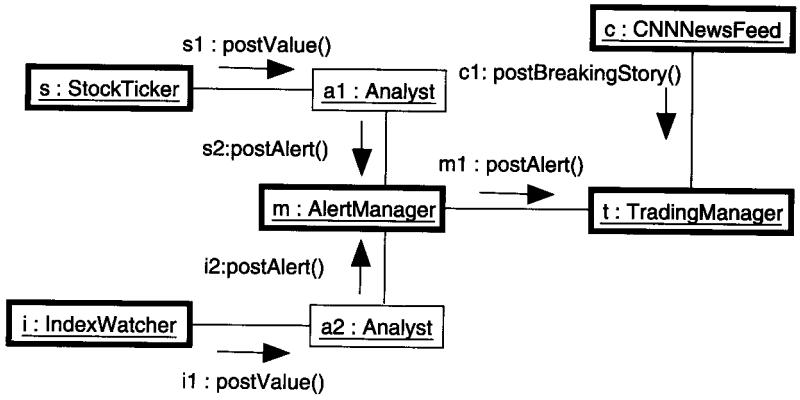

图22-4 对控制流建模

在像这样的图中，还常常附加上对应的状态机，用正交状态来显示每个主动对象的详细行为。【在第21章中讨论状态机。】

22.3.2 对进程间通信建模

作为体现系统中多个控制流的一部分工作，你还必须考虑供存在于不同流中的对象相互通

信的机制。跨线程（存在于相同的地址空间）的对象可以通过信号或调用事件进行通信，后一种方式既可以展示异步的语义，又可以展示同步的语义。而对于跨进程（存在于不同的地址空间中），通常必须使用不同的机制。【在第20章中讨论信号和调用事件。】

进程间的通信问题含有这一事实：即在分布式系统中，进程可能存在于分散的节点上。从分类来看，进程通信有两种方式，即消息传送和远程过程调用。在 UML中，你仍可以分别把这种方式建模为异步或同步事件。但是因为这不再是简单的进程内的调用，所以你需要使用更进一步的信息来修饰你的设计。【在第23章中讨论对位置建模。】

对进程通信建模，要遵循如下的策略：

- 对多个控制流建模。
- 考虑这些主动对象中哪个代表进程，哪个代表线程。使用合适的构造型来区别它们。
- 用异步通信对消息建模；用同步通信对远程过程调用建模。
- 用注解来非形式化地说明基本通信机制，或者用协作来较形式化地说明基本通信机制。

【在第6章中讨论构造型；在第6章中讨论注解；在第27章中讨论协作。】

图22-5显示了一个分布式预订系统，它的进程跨越 4个节点。每个对象都用 process构造型来标记。每个对象还用一个location标记值来标记，以说明它的物理位置。在ReservationAgent、TicketingManager和HotelAgent之间的通信是异步的。通过用一个注解来建模，通信被描述为是建立在一个 JavaBeans消息传送服务上的。在TripPlanner和ReservationSystem之间的通信是同步的。它们的交互语义可以在命名为 CORBA ORG的协作中找到。TripPlanner作为一个client工作，ReservationAgent作为一个server工作。通过放大这个协作，你将发现有关服务器端与客户端之间是如何协作的细节。【在第26章中讨论节点。】

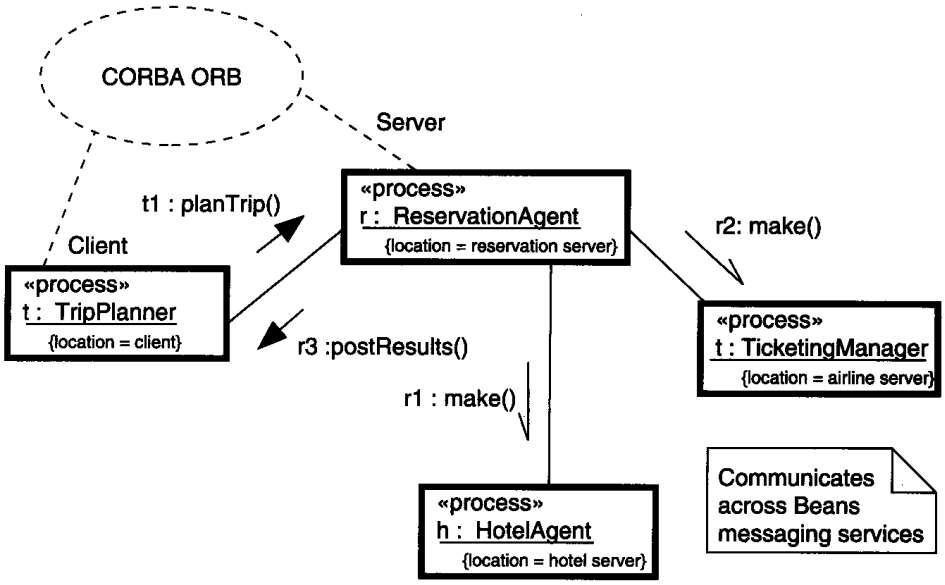

图22-5 对进程间通信建模

22.4 提示和技巧

一个结构良好的主动类和主动对象，应满足如下的要求：

- 代表一个独立的控制流，该控制流最大限度地挖掘了系统中真正并发的潜力。
- 不要太细地粒化，否则将需要大量的其他主动元素，从而导致一个设计过度的、脆弱的进程体系结构。
- 仔细地管理对等的主动元素之间的通信，在异步和同步的消息传送之间作出选择。
- 仔细地把每个对象当做一个临界区域来对待，使用合适的同步特性，以便在出现多控制流时能维持它的语义。
- 显式地区分进程和线程的语义。

在UML中绘制一个主动类或主动对象时，要遵循如下的策略：

- 只显示那些对理解它的语境中的抽象是重要的属性、操作和信号。
- 显式地显示所有操作的同步特性。