第23章
时间和空间

本章内容：

- 时间、持续时间和位置
- 对时间约束建模
- 对对象的分布建模
- 对移动的对象建模
- 处理实时系统和分布式系统

真实的世界是一个严厉无情的地方。事件可能在不可预料的任何时刻发生，还会在一个特殊的时刻要求一个特殊的响应。系统的资源可能需要被分布在世界各地——某些资源甚至还要移动，这就产生了关于传输时间、同步、安全性和服务质量的问题。

对时间和空间建模是任何实时系统或分布式系统的基本要素。你可以使用许多 UML的特性，包括时间标记、时间表达式、约束和标记值，来可视化、详述、构造和文档化这些系统。

处理实时系统和分布式系统是困难的。好的模型可以揭示系统的时间和空间特性中的必需的和足够的特性。

23.1 入门

当你开始对大多数软件系统建模时，你通常假定一个理想的环境——消息在零时间内被发送，网络从不断开连接，工作站从不失败，网络上的负载甚至总是平衡的。不幸的是，现实世界并不以这种方式工作——消息要花费时间来传送（并且有时永远传递不到），网络会断开，工作站会失败，网络上的负载常常是不平衡的。所以，当你遇到必须在现实世界操作的系统时，你一定要把时间和空间的问题考虑进去。

实时系统是这样一个系统，它的某些行为必须在一个精确的绝对或相对时刻，或在一个可预见的（常常是受限的）持续时间内完成。在一种极端的情况下，这样的系统是严格实时的，需要在几纳秒或几微秒内完成可重复的行为。在另一种极端的情况下，模型可能是近似实时的，也需要可预料的行为，但只要求在几秒或更长的时间内完成。

分布式系统是这样一个系统，它的构件可以被物理地分布在各个节点上。这些节点可以代表物理上位于同一个机壳中的不同的处理器，甚至也可以代表彼此相距大半个地球远的计算机。【在第25章中讨论构件；在第26章中讨论节点。】

为了表示实时系统和分布式系统所需的建模， UML提供了时间标记、时间表达式、时间约束和位置的图形化表示，如图 23-1所示。

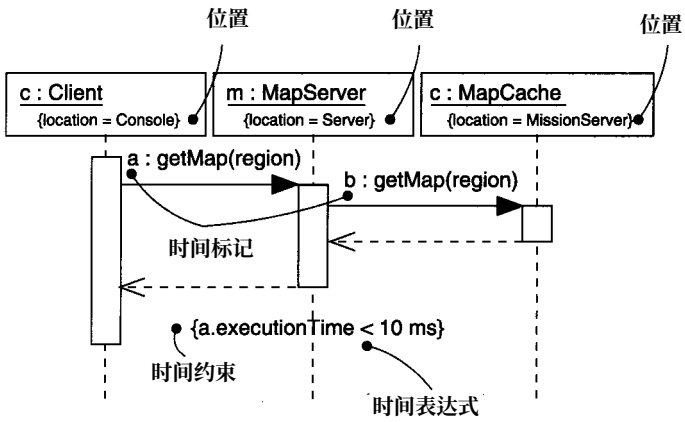

图23-1 时间约束和位置

23.2 术语和概念

时间标记（*timing mark*）是一个表示事件发生时刻的符号。从图形上，时间标记是由给予消息的名称（通常有别于通过消息执行的动作的名称）形成的一个表达式。时间表达式（*timing expression*）是一个用来判断绝对或相对时间值的表达式。时间约束（*timing constraint*）是关于绝对或相对时间值的一个语义陈述。从图形上看，时间约束的表示同任何约束一样，即用一个括号括起来的字符串表示，并一般通过一个依赖关系连接到一个元素。位置（*location*）是指一个构件在一个节点上的位置。从图形上看，位置用一个标记值表示，即用括号括起字符串，并把它放在一个元素名下面；或是用节点内构件的嵌套来表示。

1. 时间

实时系统就像它们的名称所表示的，是时间关键系统。事件可以在规则或不规则的时间发生；对一个事件的响应却必须在可预料的绝对时间或者相对于事件本身的时间发生。【在第20章中讨论事件（包括时间事件）。】

消息的传送表示系统的动态方面，所以当你用 UML对一个系统的时间关键特性建模时，你可以为交互中的每一个消息取一个作为时间标记来使用的名称。一个交互中的消息通常是没有名称的。它们主要用事件（如信号或调用）的名称来表示。结果是你不能用一个事件的名称来写一个表达式，因为同一事件可能触发不同的消息。如果所指的消息是有歧义的，就要在时间标记中用一个显式的消息名来指出这个想在时间表达式中提到的消息。时间标记不过是由交互中的消息名形成的表达式。对于一个给定的消息名，你可以引用该消息的三个函数中的任一个函数：开始时间（`startTime`）、终止时间（`stopTime`）和执行时间（`executionTime`）。然后你可以用这三个函数来说明任何复杂的时间表达式，甚至可以使用权值或偏移量，这些权值和偏移量可以是常量，也可以是变量（只要这些变量能在执行时间得到确定）。最后，如图23-2所示，可以把这些时间表达式放进一个约束中，用来说明系统的时间行为。至于约束，你

可以把它们放在合适的消息的附近，或者用依赖关系显式地连接到消息上来进行表示。【在第15章中讨论消息和交互；在第6章中讨论约束。】

注释 尤其对于复杂系统，用命名的常量写表达式要比显式地写时间好。这样，你就可以在你的模型的一个地方定义这些常量，然后在多个地方引用它们。采用这种方式，如果系统对时间的需求发生了变化，就很容易更新模型。

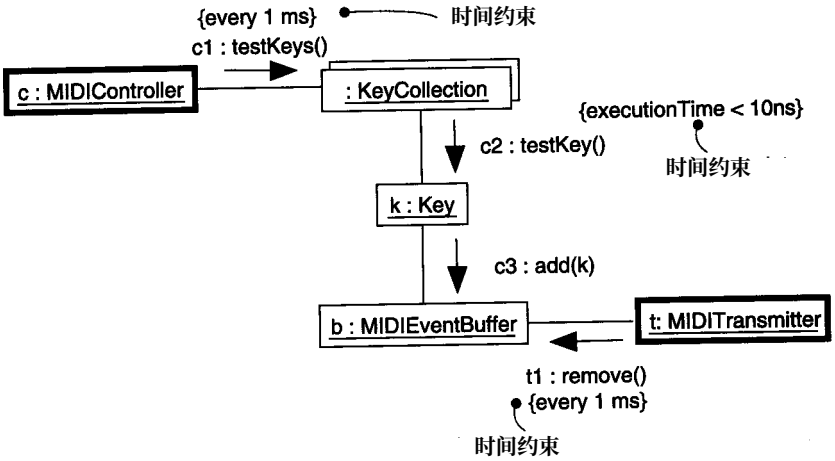

图23-2 时间

注释 你也可以对操作使用时间表达式。称为 semantics的标准标记值可以附加到任何操作上，并在那里使用一个时间表达式来说明该操作的时间复杂性。时间复杂性通常用于对你希望操作完成的最小、最大和（或）平均时间建模。说明一个操作的时间复杂性也就是说明该操作的时间预算。可以用以下两种方法：一种是，通过在开发过程中宣称你的时间预算，然后度量运行的系统，你就可以对设计时和建造时的系统行为进行明智的比较；另一种是，通过将一个交互中所有操作的时间表达式的结果值（设计的或实际的）加在一起，你就可以计算出整个事务的时间复杂性。【在第9章中讨论标记值semantics。】

2. 位置

分布式系统的本质是包含物理上分散于系统各节点上的构件。对许多系统而言，构件在被装载到系统上时，位置是固定的；而在另一些系统中，构件是可以从节点到节点移动的。【在第25章中讨论构件；在第26章中讨论节点。】

在UML中，用实施图来对一个系统的实施视图建模。实施图代表系统在其上执行的处理器和设备的拓扑结构。构件（如可执行体、库、表等）存在于这些节点上。一个节点的每个实例将拥有某些构件实例，而一个构件的每个实例肯定属于一个节点的一个实例（尽管同一种构件的实例可能分散到不同的节点上）。例如，图23-3所示，可执行的构件vision.exe存在于名为KioskServer的节点上。【在第30章中讨论实施图；在第2章和第13章中讨论类/对象二分法。】

普通类的实例也可能存在于一个节点上。如图 23-3中所示，类LoadAgent的一个实例存在于名为Router的节点上。【在第4章和第9章中讨论类。】

像图形所显示的那样，在 UML中你可以用两种方式对一个元素的位置建模。第一种方法如KioskServer所示，你可以将这个元素（文本或图形）物理地嵌套在包含它的那个节点的附加栏中。第二种方法如LoadAgent所示，用定义的标记值location来指定类的实例存在于哪个节点上。当你看重在图中用一个可视化的提示来显示构件的空间分隔和分组时，通常使用第一种方式。类似地，当你认为对一个元素的位置建模虽然重要，但对你手头的图来说它较为次要，例如要可视化实例间消息的传送时，则可使用第二种形式。【在第6章中讨论标记值和附加栏。】

> 注释 当你想显示一个构件随时间重新分布的情况时，对元素位置建模的第二种形式尤为有用。例如，你可以使用一个become消息来对当前存在于一个位置、并向另一个位置移动的对象进行建模。类似地，你可以使用一个 copy消息来显示远程对象间的语义关系。【在第13章中讨论构造型become和copy。】

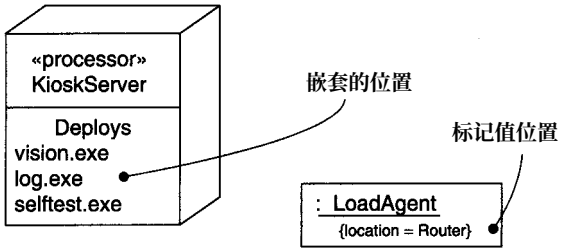

图23-3 位置

23.3 普通建模技术

23.3.1 对时间的约束建模

对事件的绝对时间建模、对事件间的相对时间建模和对完成动作所需要的时间建模是使用时间约束的实时系统的三个主要时间关键特性。【在第6章中讨论约束是UML的扩展机制之一。】

对时间约束建模，要遵循如下的策略：

- 对于交互中的每个事件，考虑它是否必须从某一绝对时间开始。将这个实时特性建模为消息上的时间约束。
- 对于交互中每个值得关注的消息序列，考虑是否有一个相关的最大的相对时间。将这个实时特性建模为该序列上的时间约束。
- 对于每个类中的每个时间关键操作，考虑它的时间复杂性。将这些语义建模为操作上的时间约束。

例如，图23-4中，最左边的约束说明调用事件“刷新”refresh的重复的开始时间。类似

地，中间的时间约束说明调用 getImage 的最大持续时间。最后，最右边的约束说明调用事件 getImage 的时间复杂性。

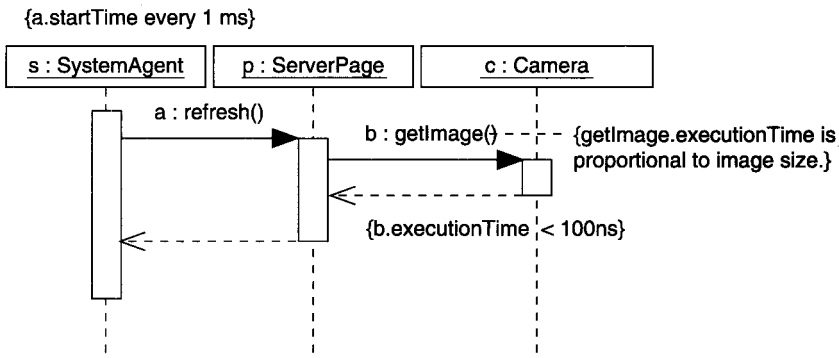

图23-4 对时间约束建模

注释 注意executionTime可应用于像getImage这样的动作，也可应用于像a和b这样的时间标记。此外，像这样的时间约束可以用自由格式的文本书写。如果你想更精细地说明其语义，可以用UML的对象约束语言（OCL）来描述。OCL在《UML参考手册》一书中有更详细的描述。

通常为消息选择简短的名称，以使其不与操作的名称相混淆。

23.3.2 对对象的分布建模

当你对分布式系统的拓扑结构建模时，你将要考虑构件和类的实例的物理位置。如果你的重点是已实施系统的配置管理，那么对构件的分布建模将显得尤其重要，以便可视化、详述、构造和文档化物理事物（如可执行体、库和表）的位置。如果你的重点是在系统的功能、可扩展性和吞吐量上，那么对对象的分布建模则是重要的。【在第25章中讨论对构件的分布建模。】

决定如何分布一个系统中的对象是一个棘手的问题，这不仅仅因为分布问题和并发问题是相互影响的。而且，幼稚的解决办法将导致低下的性能；同时，过度工程化的解决办法也不好。事实上，由于它们通常以脆弱而告终，所以可能更坏。【在第22章中讨论对进程和线程建模。】

对对象的分布建模，要遵循如下的策略：

- 对于系统中每个值得注意的对象类，考虑它的引用位置。换句话说，考虑所有它的邻居和它们的位置。紧密耦合的位置将有很邻近的对象，而松散耦合的位置则有远程的对象（这样同它们通信就有传输时延问题）。暂时将对象分配得靠近对它进行操作的参与者。
- 下一步考虑相关对象集合之间的交互模式。将具有高度交互的对象集合放在一起，以减少通信的开销。将具有低度交互的对象集合分开。
- 下一步考虑系统职责的分布，重新分布对象以平衡每个节点上的负载。
- 也要考虑安全、易变性和服务质量问题，并视情况而定，重新分布对象。
- 用以下两种方法之一来表示分配：

1）将对象嵌套到一个实施图的节点中。

2）用一个标记值显式地指明对象的位置。

图23-5提供了一个对象图，它对一个零售系统的一些对象的分布进行建模。这张图的价值在于，它让你看到某些关键对象的物理分布。如图所示，两个对象（Order和Sales）位于Workstation上，两个对象（ObserverAgent和Product）位于Server上，一个对象（productTable）位于DataWarehouse上。【在第14章中讨论对象图。】

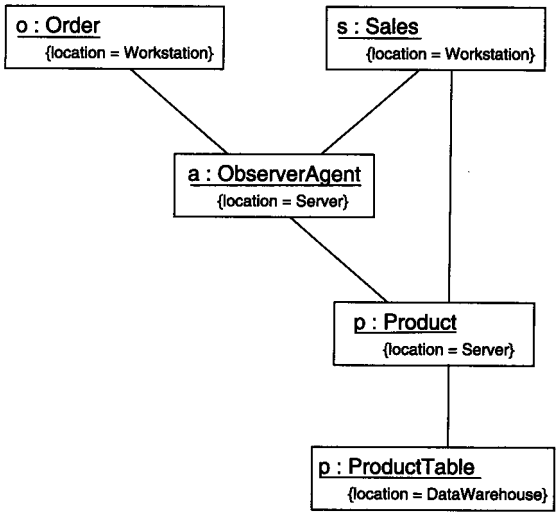

图23-5 对对象的分布建模

23.3.3 对移动的对象建模

在许多分布式系统中，构件和对象一旦装在某个系统上就不再移动了。在它们的生命期中，从创建到撤销，它们不会离开产生它们的那个节点。然而也有某些分布式系统，其中的某些事物要来回移动，通常是由于以下一个或两个原因：

第一，对象移动是为了更接近它们需要与之一起工作的参与者和其他对象，以便工作得更好。例如，在一个全球海运系统中，你会看到代表船只、集装箱和运货单的对象从一个节点移动到另一个节点，以跟踪它们物理上的对应事物。如果你在香港有一艘船，最好是把代表船只、集装箱和运货单的对象放在香港的一个节点上。当该船驶往旧金山时，就把相关的对象移动过去。

第二，对象的移动是为了对一个节点或连接的失败作出反应，或者是为了平衡多个节点间的负载。例如在一个空间交通控制系统中，不允许一个节点的失败，以便跟踪一个国家的所有操作。而像这样的一个容错系统将把元素转移到其他的节点上。尽管性能和生产率有所下降，但却保持了安全的功能。类似地，尤其是在基于 Web的系统中，必须处理不可预料的需求高峰，你常常想要建立机制来自动地平衡进程负载，也许通过把构件和对象移动到使用不充分的站点上来实现这个目标。

决定在一个系统中如何移动对象，是一个比简单的静态分布更加棘手的问题，因为移动会带来同步和一致性的困难问题。

对对象的移动建模，要遵循如下的策略：

- 为了在节点间物理地传输对象，选择一个基础的机制。【在第28章中讨论机制。】
- 通过用一个标记值显式地指明对象的位置，来表示该对象分配到一个节点上。
- 用构造型为become和copy的消息，来表示对象分配到在一个新节点上。【在第13章中讨论构造型become和copy】
- 考虑同步（保持复制对象状态一致）和一致性（当对象移动时保存它的名称）问题。

图23-6提供了一个协作图，它对Web代理的移动建模，这些代理从节点到节点移动、收集信息、对资源投标以求自动地提供最低价格的旅行票。特别是，这张图显示了类TravelAgent的一个实例（命名为t）从一个服务器移动到另一个服务器。沿着这条路，这个对象在每个节点上与匿名实例Auctioneer交互，最后，它向位于client server上的Itinerary对象提供一个标价。【在第8章中讨论协作图。】

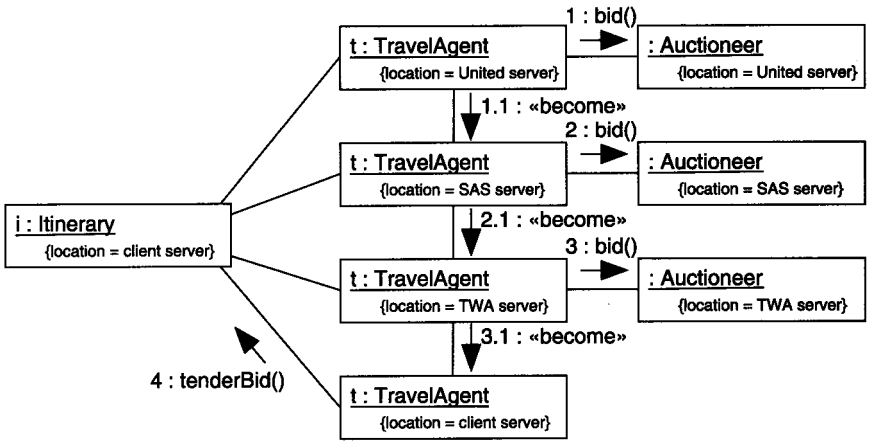

图23-6 对移动的对象建模

23.4 提示和技巧

一个具有时间和空间特性的结构良好的模型，应满足如下的要求：

- 仅揭示那些对于捕捉系统所期望的行为必要而充分的时间和空间特性。
- 集中使用那些特性，使之易于发现和修改。

当在UML中绘制一个时间或空间特性时，要遵循如下的策略：

- 给时间标记（消息名）取一个有意义的名称。
- 清晰地区分相对的和绝对的时间表达式。
- 主要用标记值显示空间特性。仅当可视化实施系统中的元素位置变得重要时，才使用嵌套形式。