第3章

UML 简介

UML 是软件工程中系统建模的一个常用工具。本章对 UML 进行简要介绍，希望读者通过本章了解该工具，在系统开发时能够利用该工具（或其他建模工具），建立初步的软件工程的思想。

3.1 UML 概述

模型是系统的蓝图，可以帮助开发人员规划要建立的系统，保证用户的要求得到满足。对于一个软件系统，模型就是开发人员为系统设计的一组视图。

UML 是在面向对象的系统开发中进行说明、可视化和文档编制的一种标准语言，是现在流行的建模和规约语言，并且 UML 独立于任何具体的程序设计语言。UML 具有广泛的建模能力，不仅可以用于软件建模，而且可以用于其他领域的建模。

UML 可以建立需求模型、逻辑模型、设计模型和实现模型等，从不同角度描述人们观察到的软件视图。它采用一组图形符号描述软件模型，具有简单、直观和规范的特点。

概括地说，UML 主要有以下 3 个作用：

（1）为软件系统建立可视化模型。在 UML 的可视化模型中，视图不仅可以描述用户需要的功能，还可以描述如何实现这些功能，这使得系统更加直观、易于理解，有利于系统分析人员和系统用户之间以及系统分析人员和系统设计、开发人员之间的交流，也有利于后期系统的维护。

（2）为软件系统建立架构。UML 不是面向对象的程序设计语言，但它的模型可以直接映射到各种各样的程序设计语言。例如，它可以使用代码生成器工具将 UML 模型转换为多种程序设计语言（如 C++、Java 等）的代码，也可以使用反向生成器工具将程序源代码转换为 UML 模型。

（3）为软件系统建立文档。UML 可以为系统的体系结构及其所有细节建立文档，不同的 UML 模型图可以作为项目不同阶段的软件开发文档。

UML 中的图主要包括用例图、时序图、类图、活动图、部署图（也称配置图）、状态图、协作图、构件图（也称组件图）等。本章将对本书中使用的部分 UML 图进行简要的介绍。

3.2　用　例　图

3.2.1　用例图概述

用例图(use case diagram)是描述用户与系统功能之间以及各系统功能之间关系的最简表示形式,是进行系统需求分析的一个结果。

用例图的主要作用有以下 3 个:

(1) 提纲挈领地让相关人员(包括系统用户、系统分析人员、系统设计人员和系统开发人员等)了解系统的概况,最主要的是系统能提供的功能。

(2) 展现了用户和(与其相关的)用例(系统功能)之间的关系,从而说明是谁要使用该系统,以及他们使用该系统可以做些什么。因此,用例图是项目参与者间交流的良好工具。

(3) 展现了各用例(系统功能)之间的关系,从而说明系统功能之间是如何关联的,这为系统设计人员和系统开发人员理解系统内部功能提供了便利。

用例图见图 3-1。

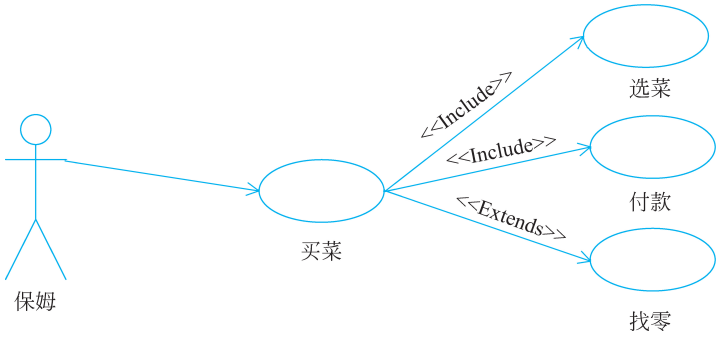

图 3-1　用例图示例

- 角色(actor),也称参与者,是与应用程序或系统进行交互的用户、组织或外部系统,用一个人形符号表示。这里要说明,用户不一定是真实的人。
- 用例(use case)是外部可见的系统功能,用来对系统提供的服务进行描述。用例用椭圆表示。

角色和用例之间的连线表示角色与用例之间存在着使用的可能性,可以有箭头,也可以没有。如果有箭头,则表示了谁是主动方。如果不想表示主动/被动关系,则可以不画出箭头。

用例图经常和表格形式的用例描述配合使用。买菜这个用例的用例描述如表 3-1 所示。

这里要说明的是,用例一般不用颜色进行填充。在本书中,为了便于用户阅读,对于有用例描述的图形填充了灰色。

表 3-1 用例描述示例

名称	买菜
标识	UC0101
描述	带上手机，乘车去市场，根据采购意愿进行选菜，付款后乘车回家
前提	手机中有钱，家里缺菜，心里有采购意愿
结果	买到菜，或者因故中途返回
注意	往返可能有多种途径

3.2.2 用例图中描述的关系

用例图中描述的关系主要有泛化、包含和扩展。

1. 泛化

泛化(inheritance)就是通常理解的继承关系，子用例和父用例相似，但表现出更为特别的行为。子用例将继承父用例的所有结构、行为和关系。对于父用例的行为，子用例可以直接使用它，也可以重载它。泛化关系示例如图 3-2 所示。

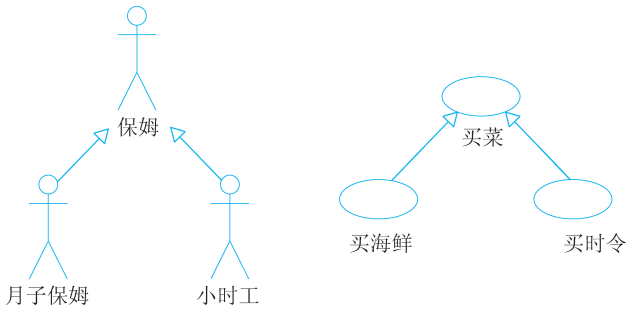

图 3-2 泛化关系示例

2. 包含

包含(include)关系用来描述一个较为复杂的用例(基础用例)，将该用例所表示的功能分解成较小的子功能(子用例)。这种情况类似于将程序的某一段代码封装成一个函数，然后再从主程序中调用该函数。

例如图 3-3，买菜必然涉及去市场、选购以及付款等子功能，如果都要在一个用例中详细描述，就会过于复杂而且容易丢失细节。这时可以用包含关系将复杂用例加以细化，能够帮助相关人员较好地理清主要业务。

用例之间的包含关系用箭线表示，箭线由基础用例出发，延伸至子用例，线上写明<<Include>>或者<<包含>>。

3. 扩展

扩展(extend)关系指对用例功能的延伸，相当于为基础用例提供一个附加的功能。扩展的用例是一段可选的动作，与基础用例相互独立，从而使基础用例行为更简练，目标更清晰。

扩展关系示例如图 3-4 所示。可以看到，买菜可能会涉及提钱、开发票、复称等动作。

用例之间的扩展关系用箭线表示，箭线由基础用例出发，延伸至扩展用例，线上写明<<Extends>>或者<<扩展>>。

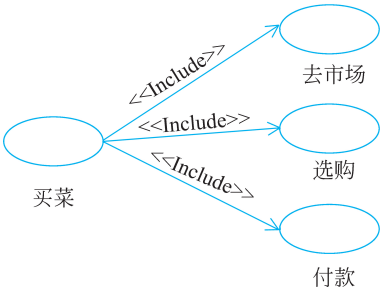

图 3-3　包含关系示例

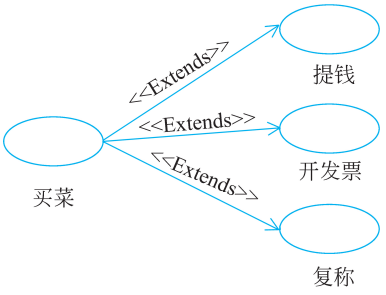

图 3-4　扩展关系示例

3.3　时　序　图

时序图(sequence diagram)又称顺序图，是用来显示角色和对象如何以有一定顺序的步骤与系统其他对象交互的模型，即时序图可以用来展示角色与对象之间以及各对象之间是如何进行交互的。时序图通过描述对象之间发送消息的时间顺序，从而显示多个对象之间的动态协作。

时序图的示例如图 3-5 所示，下面参照该图进行讲解(图 3-5 中的灰色方块为插入的解释，不是时序图的一部分)。

时序图包括如下元素：

(1) 角色。可以是人或者其他系统、子系统，以一个人形符号表示。

(2) 对象(object)。代表交互过程中的实体，位于时序图顶部，以一个矩形表示。

(3) 生命线(lifeline)。代表时序图中的角色和对象在一段时期内的存在，每个角色和对象都有生命线，用一条垂直的虚线表示，实际上可以理解为纵坐标，以显示时间。

(4) 激活(activation)。代表时序图中的角色和对象执行一项操作的活跃时期。

(5) 消息(message)。用来定义交互和协作过程中交换的信息，它在实体间传递。

时序图中的消息可以是信号、操作调用或远程过程调用(Remote Procedure Call，RPC)，允许实体请求其他的服务。消息可以用消息名及参数标识，必要情况下还可带有条件表达式(如图 3-5 中的“[天气下雨]”)。

消息分为 3 种类型：

(1) 同步消息(synchronous message)。消息的发送端把控制传递给消息的接收端，

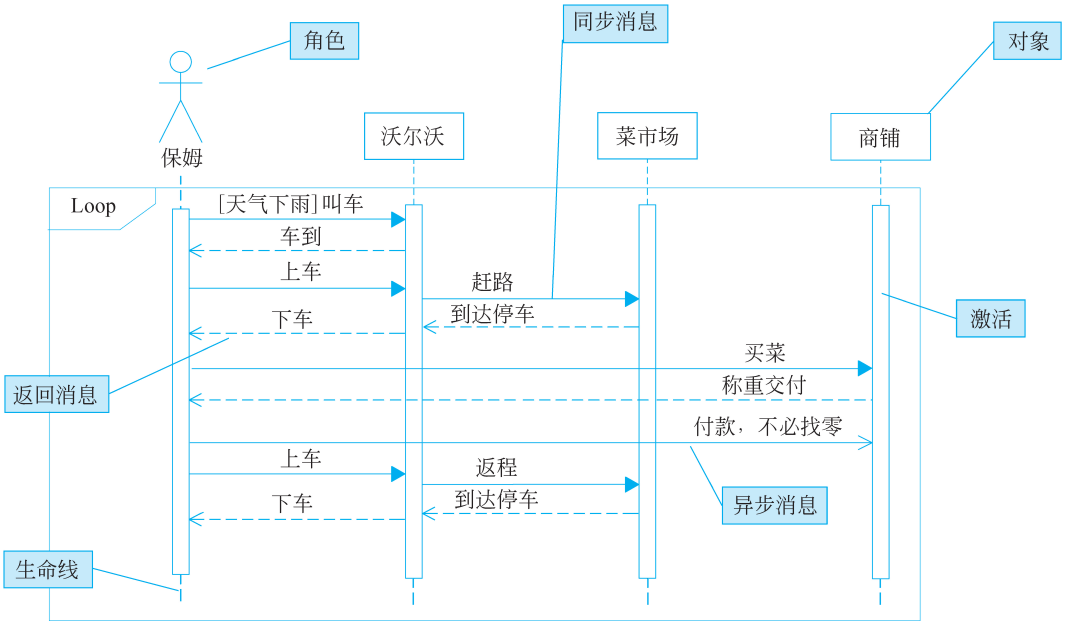

图 3-5 时序图示例

然后停止其他活动，等待接收端返回消息或者控制后再往下执行。同步消息以实线＋实心箭头表示。

(2) 异步消息(asynchronous message)。消息的发送端把信号传递给消息的接收端，然后继续自己的活动，不等待接收端返回消息或者控制，即接收端和发送端是并发工作的。异常消息以实线＋大于号形箭头表示。

(3) 返回消息(return message)。表示从过程调用返回。返回消息以小于号形箭头＋虚线表示。

图 3-5 还给出了一个循环(Loop)，表示保姆每天去买菜。

3.4 类 图

类图是一种静态模型，用于描述系统中的类以及各个类之间的关系。

类图能够让系统开发人员在正确编写代码以前对系统有全面的认识。它不但是系统设计人员关注的核心，而且是系统实现人员关注的核心。

类图示例如图 3-6 所示。下面参照图 3-6 进行讲解。

在类图的每个表示类的框中，上面一格的内容是类名，中间一格的内容是该类拥有的属性(field)，下面一格列出该类拥有的方法(method)。类名是不能省略的，其他两部分可以省略。

在属性和方法前面有一些可见性修饰符：

- ＋表示 public。
- －表示 private。

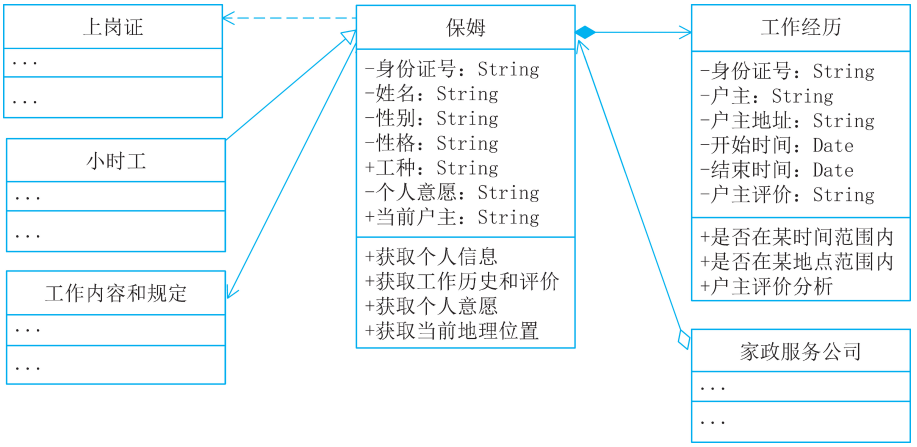

图 3-6　类图示例

- ＃表示 protected(friendly 也归入这类)。

类之间可以存在一定的关系，常见的有以下几种关系。

1. 继承

继承表示一个类(称为子类)继承另一个类(称为父类)的功能，并可以增加它自己的新功能。继承关系使用空心箭头表示。

例如，小时工继承了保姆的相关功能。

2. 依赖

对于两个相互独立的类，当一个类负责构造另一个类的实例，或者一个类依赖于另一个类的服务时，这两个类之间体现为依赖关系。依赖关系用虚线箭头表示。

例如，保姆的工作/角色依赖于上岗证。只有系统为一个人颁发了上岗证，这个人才可以成为一个合法的保姆。

3. 关联

对于两个相互独立的类，当一个类的实例与另一个类的实例存在固定的对应关系时，这两个类之间为关联关系。关联关系用实线箭头表示。

例如，保姆与工作内容和规定有关联关系。

4. 聚合

聚合表示一种弱的拥有关系，即 has-a 的关系，体现的是 A 类可以包含 B 类，但 B 类不是 A 类的一部分。两个类具有各自的生命周期。聚合关系用空心菱形＋实线箭头表示。

例如，现在的保姆大部分只是挂靠在一家保姆公司(甚至挂靠在多家保姆公司)，并不属于任何一家保姆公司的固有资产。

5. 组合

组合是一种强的拥有关系，即 contains-a 的关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样。组合关系用实心的菱形＋实线箭头表示，还可以通过连线两端的数字表示两端各有几个实例。

例如，保姆类包含了工作经历类，两者具有基本相同的生命周期（保姆持有上岗证后才能应聘为保姆，开始拥有工作经历）。

3.5 活 动 图

活动图在本质上是一种流程图，是 UML 中对**系统动态**进行建模的一种主要形式。活动图可以用来对业务过程、工作流程建模，也可以对用例实现和程序实现进行建模。本书主要使用活动图对程序实现加以描述。

活动图示例如图 3-7 所示。下面参照图 3-7 进行讲解。

1. 初始状态和终止状态

初始状态代表工作流程的开始，用实心圆表示。一个活动图只能有一个初始状态。

终止状态代表工作流程的结束，用圆圈内加实心圆表示。一个活动图可能有多个终止状态。

2. 活动和转换

活动用来表示一个动作，是完成某个系统功能所必须发生的某个动作，所有的活动组成了完成这个系统功能的完整任务。活动用圆角矩形表示。

转换用带箭头的直线表示，在两个活动之间存在，表明两个活动的前后顺序。一旦前一个活动（箭线出发点）结束，马上转到下一个活动（箭线终点）。

3. 分支

分支用菱形表示，它有一个进入转换（箭头指向菱形），有两个或多个离开转换（箭头从菱形指向外部）。而每个离开转换都需要一个监护条件，用来表示满足指定条件的时候执行该转换。例如，图 3-7 下部有一个分支，有两个监护条件及其下一步动作，买齐就回家，否则（Else）继续选购。

4. 分叉和汇合

分叉用于将动作流分为两个或者多个并发运行的控制流。分叉用加粗的线段表示，每个分叉可以有一个输入转换（箭头指向粗线段）和两个或多个输出转换（箭头从粗线段指向外部）。每个转换都可以是独立的控制流。

例如，保姆可以一边问菜的价格，一边看其他菜，两个是并行独立的。

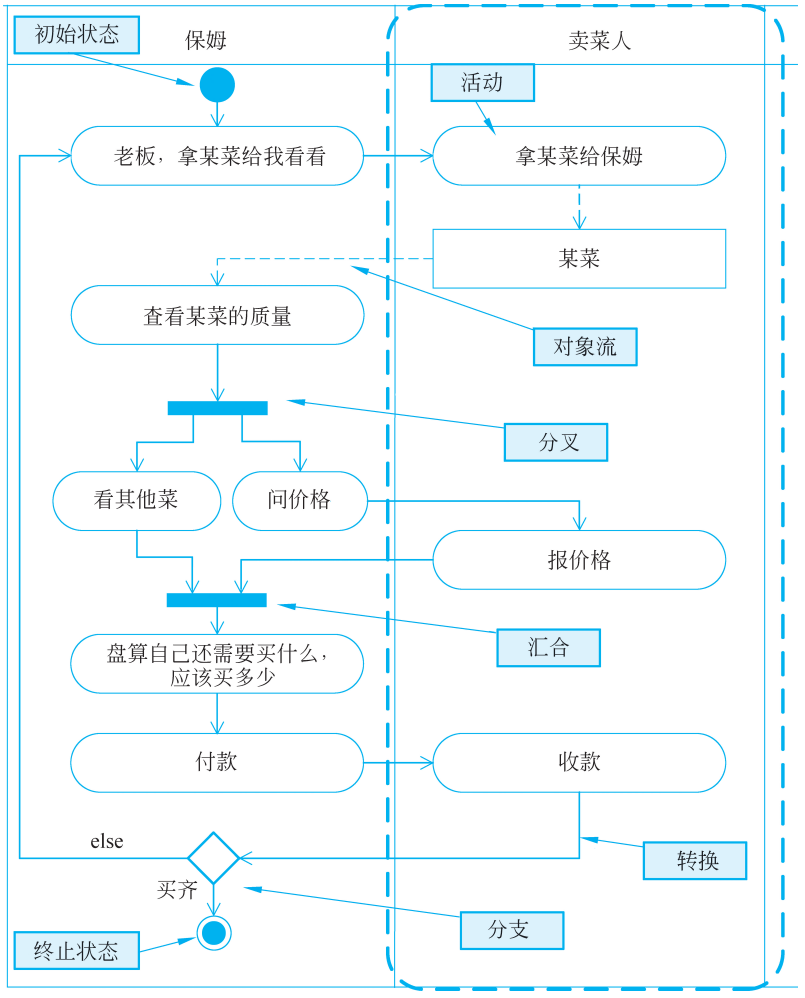

图 3-7　活动图示例

本书的很多实验都需要多线程并发操作，分叉可以很方便地描述并发运行的多个线程。

汇合用于同步并发运行的控制流，以达到共同完成一项事务的目的。只有当所有的控制流都达到汇合点后，控制流才能继续往下进行。汇合也使用加粗的线段表示，每个汇合可以有两个或多个输入转换和一个输出转换。汇合和分叉常常成对使用，汇合表示从对应分叉开始的并行控制流的结束。

5. 泳道

泳道明确地表示哪些活动是由哪些对象进行/负责的。在活动图中，每个活动只能属于一条泳道。例如，图 3-7 中包含两条泳道，分别是保姆泳道和卖菜人泳道，其中用圆角虚线矩形圈住的是卖菜人泳道。

6. 对象和对象流

对象用矩形表示，例如图 3-7 中的“某菜”。

对象流是动作状态或者活动状态与对象之间的依赖关系，表示动作使用对象或者动作影响对象。

对象流用带箭头的虚线表示。如果箭头是从动作状态出发指向对象，则表示动作对对象施加了一定的影响（例如图 3-7 中的“拿某菜给保姆”）。如果箭头从对象指向动作状态，则表示该动作使用对象流所指向的对象（例如图 3-7 中的保姆“查看某菜的质量”）。

3.6 部 署 图

部署图用来建立系统的物理部署模型，它表示一组物理节点的集合及节点之间的相互关系。部署图展示了硬件的配置以及如何将软硬件部署到网络拓扑中，例如，有哪些计算机和设备，它们之间是如何连接的，每一个设备上部署了哪些软件实体。

部署图的使用者是系统开发人员、系统集成人员和系统测试人员。通过部署图，系统的相关人员可以知道软件应该安装在具体的哪个硬件上。

例如，对于大多数网站来说，最简单的部署图如图 3-8 所示。其中，三维方块表示的节点是在运行时代表计算机资源的物理元素。

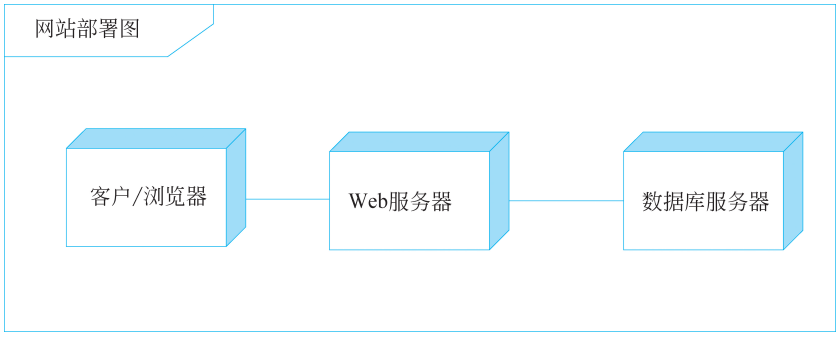

图 3-8 网站部署图