UML

1 可视化建模原理

1.1 什么是建模

1.1.1 什么是模型

模型是对现实的简化，是对主体系统的语义上封闭的抽象。

1.1.2 模型与图的区别

1. 一个模型可以包含一个或多个图。
2. 不同的图以图形方式描述模型不同部分的视图。
3. 在模型中只出现一次的模型元素可以出现在一个或多个图上，在不同的环境中展示它。

1.1.3 软件模型

软件模型：通过一定的形式和方法用来描述软件的模型。

软件建模：建立软件模型的过程被称为软件建模。

软件模型在软件开发中的作用：

1. 软件模型是软件的中间形态。
2. 软件模型是人员交流的媒介。

软件模型的内容：

1. 需求模型：描述软件向用户所能够提供的外在特性，包括软件的目标、功能、性能等。
2. 分析模型：立足于系统的抽象逻辑建模。
3. 设计模型：软件设计方案的规范化描述。包括软件的架构、详细设计、界面设计、数据库设计等模型。
4. 测试模型：测试软件的方案描述。

1.1.4 建模的好处

建模要达到的 4 个目标：

1. 将系统可视化为开发者希望的样子。
2. 允许开发者指定系统的结构或行为。
3. 提供一个用于指导构建系统的模板。
4. 记录开发者所做的决定。

1.1.5 MDA

模型驱动架构（Model Driven Architecture，MDA），是一种在软件开发中使用模型的方法。它将系统操作的规格说明与系统使用其平台功能的方式的细节分开。

功能：

1. 具体说明一个系统，而与支持这个系统的平台无关。
2. 具体说明平台。
3. 为系统选择特定的平台。
4. 将系统规范转换为针对特定平台的规范。

MDA 的视点：

1. 计算独立模型（Computational Independent Model，CIM），重点是系统的环境和需求。
2. 平台无关模型（Platform Independent Model，PIM），关注系统操作，与平台无关。
3. 平台相关模型（Platform Specific Model，PSM），重点是系统在特定平台上的具体使用。

1.2 可视化建模的四条原则

1. 创建的模型会影响问题的解决方式。在软件中，你选择的模型会极大地影响你的世界观，每一种世界观都会导致一种不同的系统。
2. 每个模型可以用不同的精确度表示。
3. 最好的模型是联系实际的。
4. 只有一个模型是不够的。每一个重要的系统最好通过一组几乎独立的模型来实现。

1.3 UML

UML 的作用：

1. 可视化。一个明确的模型有助于沟通。
2. 规格化。UML 构建的模型是精确的、明确的和完整的。
3. 构造。UML 模型可以直接联系到各种编程语言，映射到相应的代码。
4. 文档化。UML 描述了系统架构、需求、测试、项目规划和发布管理的文档。

1.4 过程与可视化建模

最适合使用 UML 的开发过程有以下几个特征：

1. 用例驱动的。
2. 以体系结构为中心的。
3. 迭代渐增的。

1.4.1 用例驱动的过程

为系统定义的用例是整个开发过程的基础。

用例的好处：

1. 简洁、简单，并且能够被各种参与人理解。
2. 帮助同步不同模型的内容。

1.4.2 以体系结构为中心的过程

系统的体系结构被用作概念化、构造、管理和发展正在开发的系统的主要工件。

好处：

1. 对项目进行智能控制，以管理其复杂性并保持系统完整性。
2. 大规模再利用的有效基础。
3. 项目管理的基础。
4. 协助基于组件的开发。

1.4.3 迭代渐增的过程

特征：

1. 在进行大规模投资之前，关键风险已得到解决。
2. 初始迭代支持早期用户反馈。
3. 测试和集成是连续的。
4. 目标里程碑着眼于短期。
5. 进度是通过评估实施情况来衡量的。
6. 可以部署部分实现。

迭代开发：

1. 最早的迭代解决最大的风险。
2. 每次迭代都会产生一个可执行版本，即系统的额外增量。

3. 每次迭代包括集成和测试。

2 面向对象概念

2.1 对象

对象是一个具有良好定义的边界和标识、封装了状态和行为的实体。

对象的状态由属性和关系表示，对象的行为由操作、方法和状态机表示。

状态是对象生命周期中满足某个条件、执行某个活动或等待某个事件的条件或情况。对象的状态通常随时间而变化。

行为决定了对象的行为和反应。对象的可见行为是由一组它可以响应的消息（对象可以执行的操作）建模的。

每个对象都有一个唯一的标识，即使它的状态与另一个对象的状态相同。

在 UML 中，对象表示为带有下划线名称的矩形。例如：

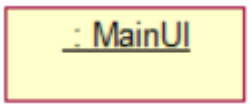

图 2.1　匿名对象

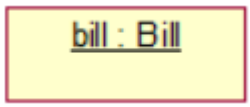

图 2.2　命名对象

2.2 面向对象的四个原则

1. 抽象

抽象用于寻找一个实体区别于所有其他实体的本质特征，定义与观察角度相关的边界。它不是一种具体的表现形式，它表示事物的理想本质。

2. 封装

对客户隐藏实现。客户依赖于接口，通过接口与系统交互。

3. 模块化

把复杂的事情分解成可管理的部分。帮助人们理解复杂系统。

4. 层次化

层次结构中相同级别的元素应该处于相同的抽象级别。

2.3 类

2.3.1 类的定义

类是对具有相同属性、操作、关系和语义的一组对象的描述。对象是类的一个实例。

类是一种抽象，因为它强调相关特征，而抑制其他特征。

在 UML 中，类用有三个隔间的矩形表示，三个隔间中分别填写类的名字、结构（属性）和行为（操作）。例如：

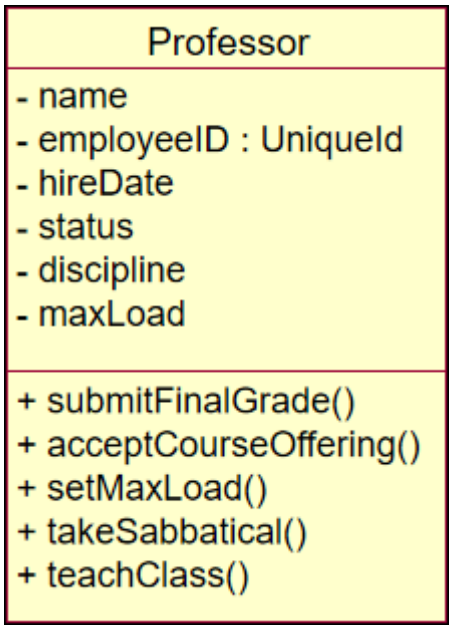

图 2.3　类

2.3.2 类与对象的关系

类是对对象的抽象定义。类定义了类中每个对象的结构和行为。类用作创建对象的模板。

类不是对象的集合。

2.3.3 属性

属性是类的命名性质，它描述性质实例可能包含的值的范围。类可以有任意数量的属性，也可以没有属性。

2.3.4 操作

操作是可以从对象请求以影响行为的服务。一个操作有一个签名，它可以限制可能的实际参数。一个类可以有任意数量的操作，也可以没有操作。

2.4 多态性与泛化

2.4.1 多态性

多态性是在单个接口后面隐藏许多不同实现的能力。

2.4.2 泛化

泛化是类之间的一种关系，其中一个类共享一个或多个类的结构和/或行为。

泛化定义了子类从一个或多个超类继承的抽象层次结构。

泛化是“is a kind of”关系。

2.4.3 继承

子类继承其父类的属性、操作和关系。子类可以添加额外的属性、操作、关系，也可以重新定义继承来的操作。

公共属性、操作和/或关系显示在层次结构中的最高适用级别。

2.5 组织建模元素

2.5.1 包

包是把元素组织成组的通用机制。

包是可以包含其他建模元素的建模元素。

包可以用于组织开发中的模型，也可以作为配置管理的一个单元。

2.5.2 图解说明

每个图都有一个框架、左上角的标题隔室和一个内容区。如果框架没有提供任何附加值，则可以忽略它，并且工具提供的图表区域的边框将是隐含的框架。

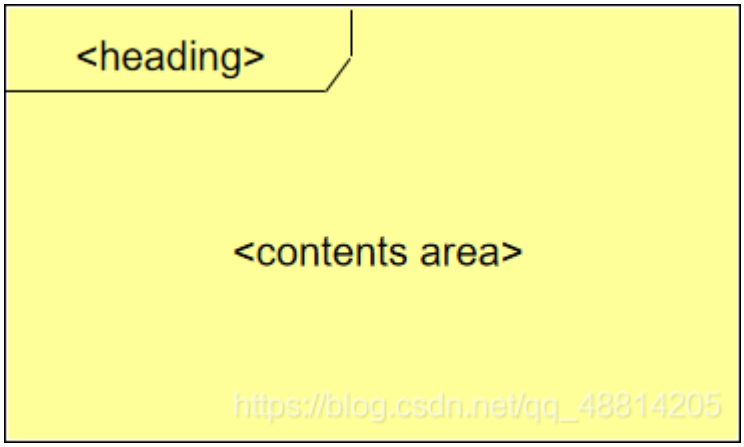

图 2.4　图解说明

3 UML结构

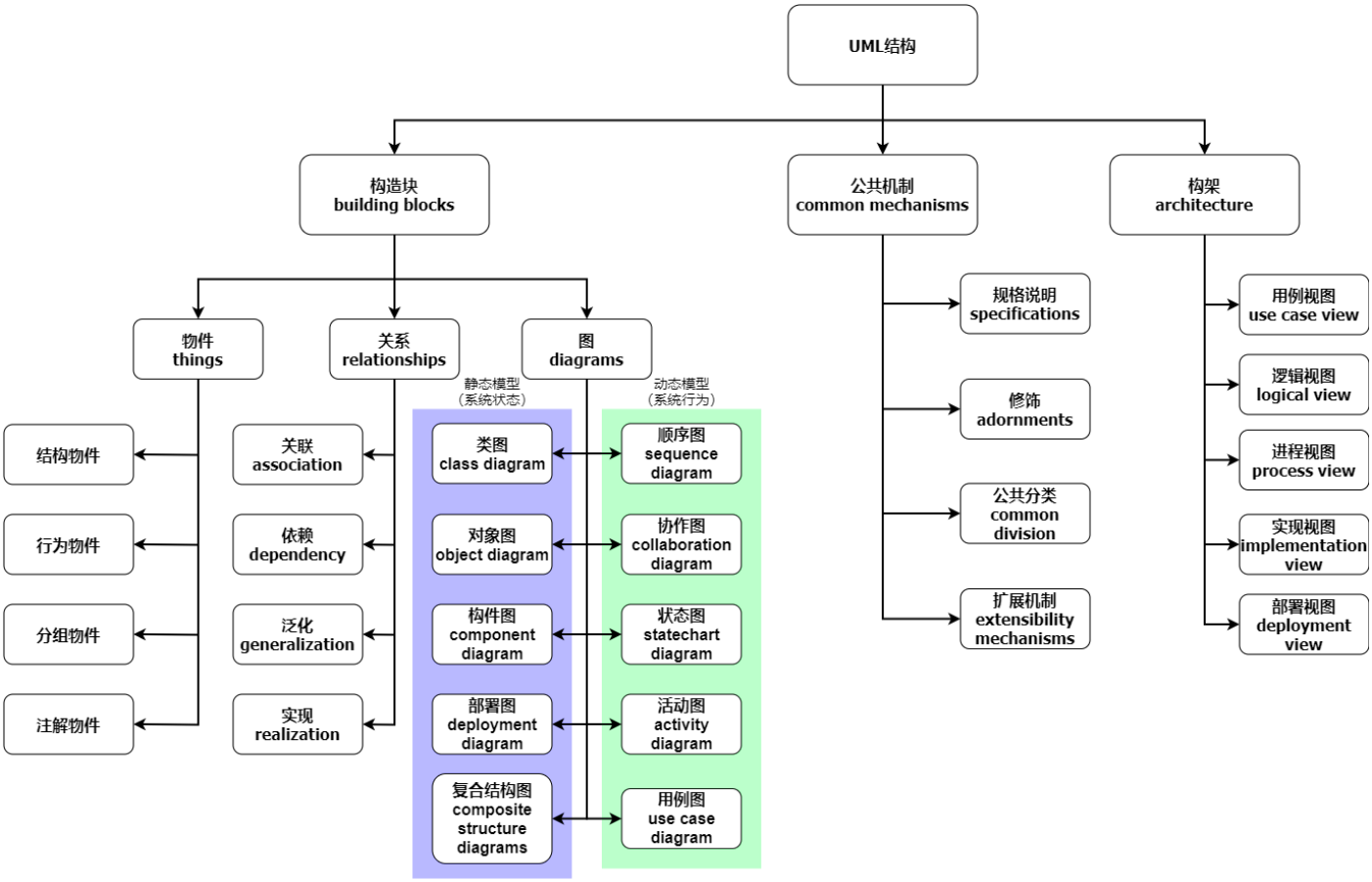

图 3.1　UML 结构

3.1 构造块

构造块：基本 UML 建模元素、关系和图。

3.1.1 物件

物件：建模元素本身。

结构物件：UML 模型中的名词，如类、接口、协作、用例、活动类、组件、节点。
行为物件：UML 模型中的动词，如交互、状态机。
分组物件：包，它用于把语义上相关的建模元素分组为内聚的单元。
注解物件：注解，它附加到模型以捕获特殊信息，同黄色便笺很相像。

3.1.2 关系

关系：把物件联系在一起，关系说明两个或多个物件是如何语义相关的。
关联：描述对象之间的一组链接。
依赖：物件的改变引起依赖物件的语义改变。
泛化：一个元素是另一个元素的特化，而且它可以取代更一般的元素。
实现：类元之间的关系，一个类元说明一份契约，另一个类元保证实现该契约。

3.2 公共机制

公共机制：达到特定目标的公共 UML 方法。

3.2.1 规格说明

规格说明：建模元素的特征和语义的文本描述——模型的“肉”。

规格说明形成了承载模型的语义背板（semantic backplane），赋予模型意义，各种图仅仅是该背板的视图或者可视化投影。

3.2.2 修饰

修饰：图中建模元素上暴露的信息项以表现某个要点。

任何 UML 图仅是模型的视图，因此，只有在修饰增强了图的整体清晰性和可读性或者突出模型的某些重要特征时，你才应该表示那些修饰。

3.2.3 公共分类

公共分类描述认识世界的特殊方法。

1. 类元（classifier）和实例
   类元：一类事物的抽象概念。如 bank account。
   实例：一类事物的特定实例。如 my bank account。

2. 接口和实现

   接口：说明事物行为的契约（做什么）。

   实现：事物是如何工作的特殊细节（如何做）。

3.2.4 扩展机制

1. 约束：允许对模型元素添加新的规则，限制一种或多个元素语义的规则。形式：{约束条件}。
2. 构造型（stereotypes）：基于已有的建模元素引入新的建模元素。
3. 标记值：允许为模型元素添加新的特性，附属于 UML 元素的各种信息，是带有相关值的关键字。形式：{属性名 = 值}。

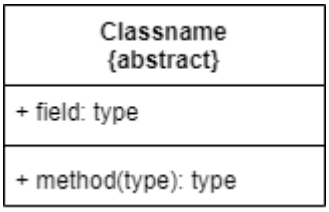

图 3.2　约束

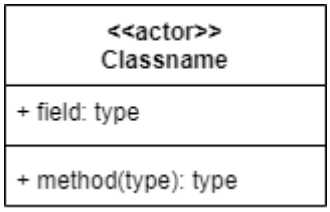

图 3.3　构造型

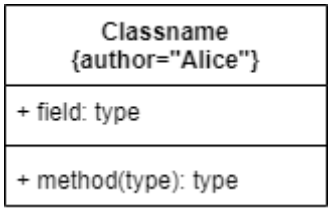

图 3.4　标记值

3.3 构架

构架：系统架构的 UML 视图。

4+1 视图：

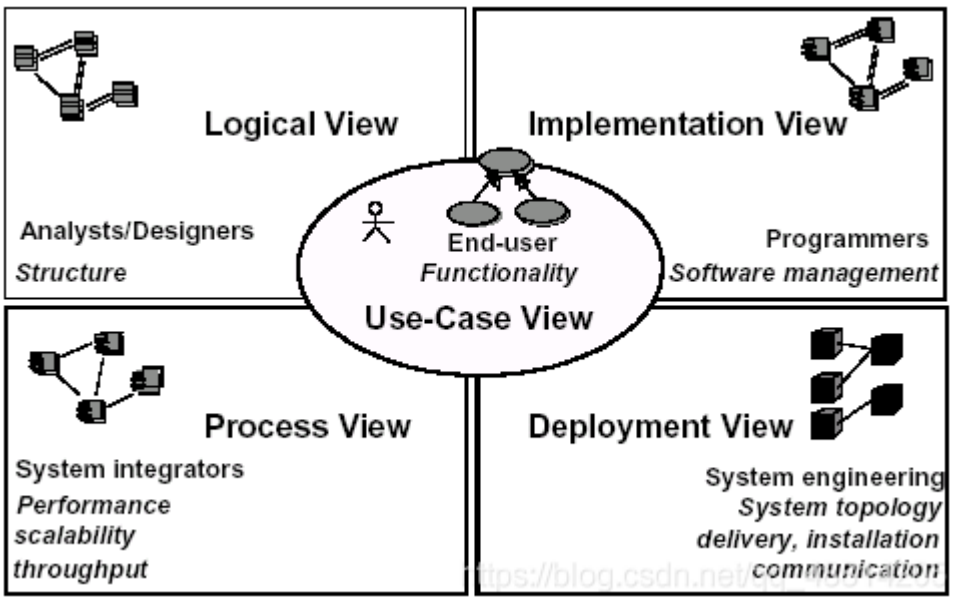

图 3.5　4+1 视图

用例视图：与终端用户有关，用于描述系统功能。这些视图由用例视图所统一，它描述项目利益相关者的需求；所有其他视图都是从用例视图派生而来，该视图把系统的基本需求捕获为用例并提供

构造其他视图的基础。

逻辑视图：与分析师、设计师有关，用于描述系统结构。重点是展示对象和类是如何组成系统、实现所需系统行为的。

进程视图：与系统集成人员有关，用于描述系统性能、可扩展性、吞吐量。对系统中的可执行线程和进程进行建模，将它们作为活动类。其实，它是逻辑视图面向进程的变体，包含所有相同的工件。

实现视图：与程序员有关，用于描述系统组装和配置管理。对组成系统的文件和组件进行建模。

部署视图：与系统工程师有关，用于描述系统的拓扑结构、交付、安装、通信。

4 用例建模

4.1 基本概念

系统行为：系统的行为和反应。它包括系统的动作和活动。

系统行为由用例捕获。用例描述系统与其（部分）环境之间的交互。

用例模型：根据用例描述系统功能需求的模型。系统预期功能（用例）及其环境（参与者）的模型。

用例模型的好处：

1. 沟通：方便用户与领域专家的沟通。
2. 识别：识别参与者与系统中的各种元素。
3. 验证：验证系统的正确性。

参与者（actor）：参与者表示与系统交互的任何东西。

用例（use case）：用例描述了由系统执行的一系列事件，这些事件产生了对特定参与者有价值的可观察结果。

4.2 用例图

4.2.1 参与者

参与者代表系统用户可以扮演的角色，它们可以表示人、机器或其他系统。

参与者可以主动地与系统交换信息，可以提供信息，也可以被动地接收信息。

参与者不是系统的一部分，它们是外来的。

4.2.2 用例

用例定义了一组用例实例，其中每个实例都是系统执行的一系列操作，这些操作为特定的参与者产生一个可观察的结果值。

用例模拟了一个或多个参与者与系统之间的对话。

用例描述了系统为向参与者提供有价值的东西而采取的行动。

用例由参与者发起，以调用系统中的特定功能。

用例与参与者之间的连线称为通信关联，表示参与者与用例的交互。无论有无箭头，通信关联都表示双向会话。箭头表示参与者触发用例。

4.2.3 用例规约

Use Case（用例名）
ID（编号）
Brief description（简要描述）
Primary actors（主要参与者，触发用例的参与者）
Second actors（次要参与者，不触发用例的参与者）
Preconditions（前置条件，约束用例开始之前的系统状态）
Main flow（主事件流，描述理想状态下的执行步骤）
Postconditions（后置条件，约束用例结束后的系统状态）
Alternative flows（备选流）

触发备选流的三种方式：

1. 直接触发备选流，而不是主事件流。
2. 在主事件流的特定步骤之后，可触发备选流。（1. The alternative flow begins after step X of the main flow.）
3. 在主事件流中，可随时触发备选流。（1. The alternative flow begins at any time.）

如果希望备选流返回主事件流，可以表示如下：N. The alternative flow returns to step M of the main flow.

4.2.4 扩展与包含

用例之间的包含关系，由基用例指向包含用例，表示在基用例的执行过程中必须执行包含用例。例如：

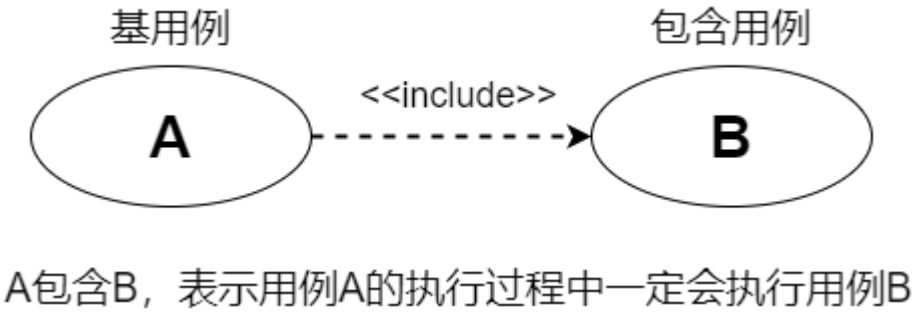

图 4.1　包含

用例之间的扩展关系，由扩展用例指向基用例。基用例本身是完整的，可以单独正确执行，在一定条件下会触发扩展用例。例如：

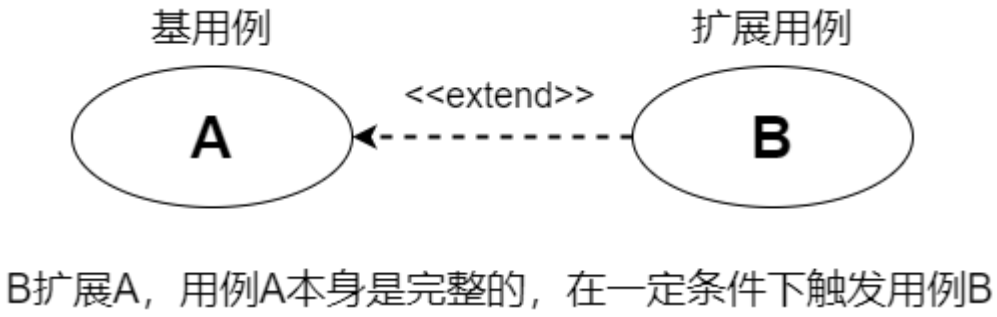

图 4.2　扩展

4.3 活动图

用例模型中的活动图可以用来捕获用例中执行的活动和动作。它本质上是一个流程图，显示从一个活动或动作到另一个活动或动作的控制流。

活动图的用途：

1. 分析用例，用图形化的方式描述用例的事件流。
2. 为工作流建模。
3. 为对象的操作建模。
4. 处理多线程应用。

活动图的组成元素：

1. 动作状态（action state）
2. 活动状态（activity state）
3. 动作流（action flow）
4. 分支（branch）与合并（merge）
5. 分叉（fork）与汇合（join）
6. 泳道（swimlane）
7. 对象流（object flow）

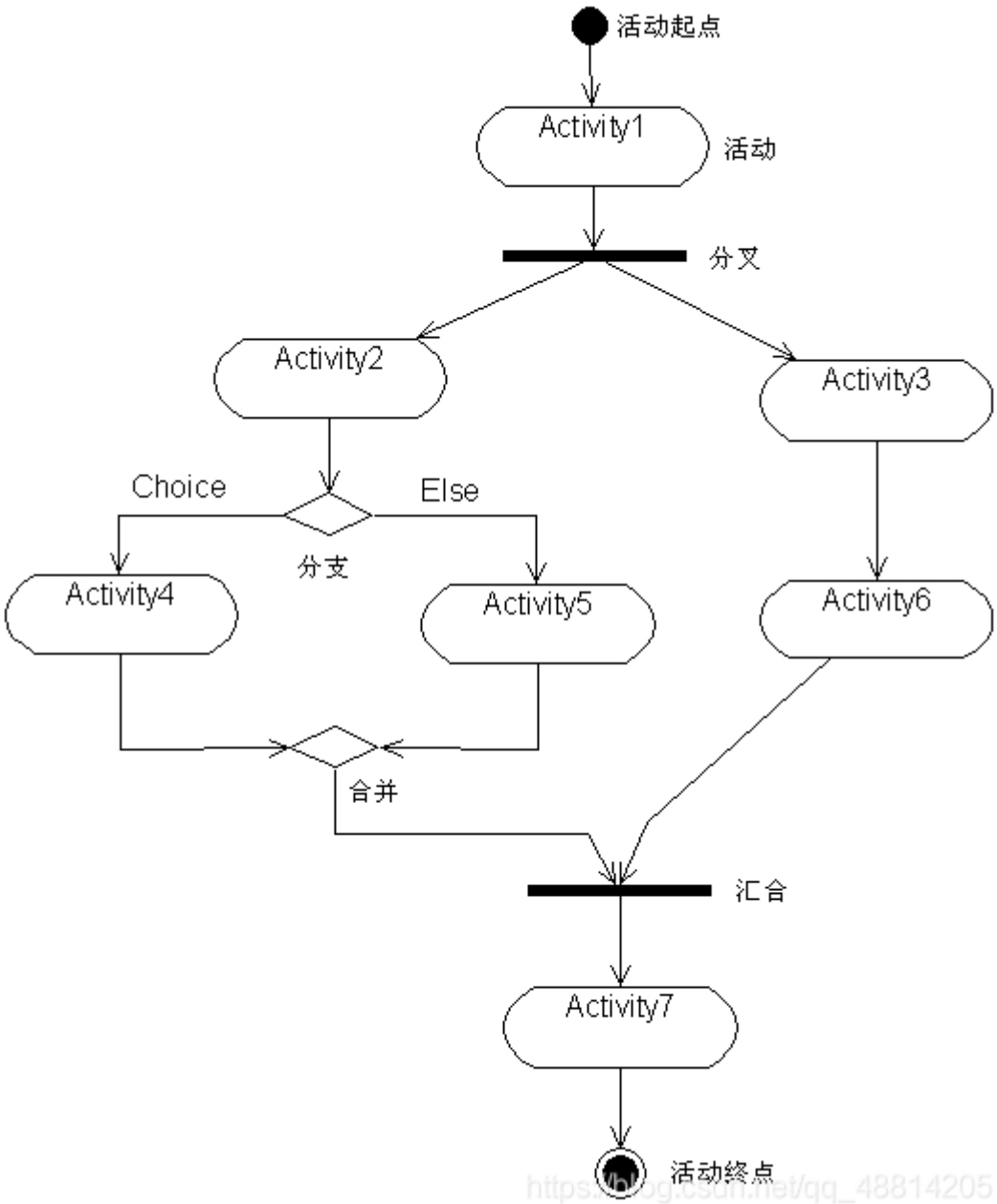

图 4.3　活动图

4.3.1 动作状态

动作状态是指执行原子的、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的状态。

动作状态使用圆角矩形表示，动作状态所表示的动作写在圆角矩形内部，如图 4.4 所示。

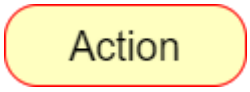

图 4.4　动作状态

动作状态的特点：

1. 动作状态是原子的，它是构造活动图的最小单位，无法分解为更小的部分。
2. 动作状态是不可中断的状态，它一旦开始运行就不能中断，一直运行到结束。
3. 动作状态是瞬时的行为，它所占用的处理时间极短，有时甚至可以忽略。
4. 动作状态可以有入转换，入转换既可以是动作流，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。
5. 动作状态不能有入口动作和出口动作，更不能有内部转移。
6. 在一张活动图中，动作状态允许多处出现。

4.3.2 活动状态

活动状态用于表达状态机中的一个非原子的执行过程。

活动状态也使用圆角矩形表示，并可以在图标中给出入口动作和出口动作等信息。

活动状态的特点：

1. 活动状态可以分解成其他子活动或动作状态。由于它可能是一组不可中断的动作或操作的组合，所以可以被中断。
2. 活动状态的内部活动可以用另一个活动图来表示。
3. 和动作状态不同，活动状态可以有入口动作和出口动作，也可以有内部转移。
4. 动作状态是活动状态的一个特例，如果某个活动状态只包括一个动作，那么它就是一个动作状态。

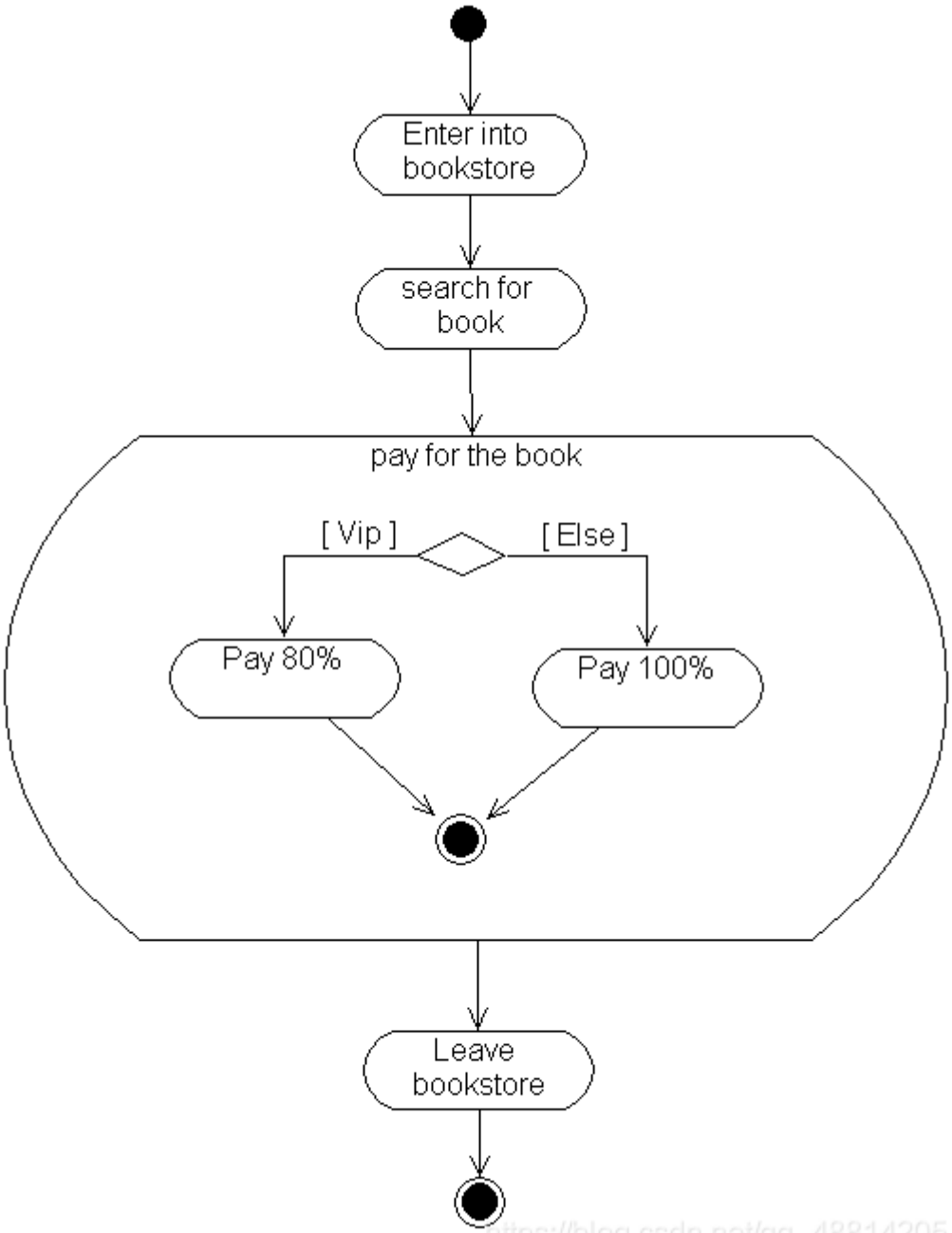

图 4.5　活动状态

4.3.3 动作流

所有动作状态之间的转换流称之为动作流。活动图的“转换”用带箭头的直线表示，如图 4.6 所示。

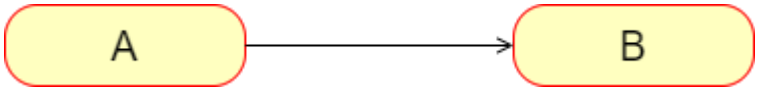

图 4.6　动作流

活动图的“转换”一般都不需要特定事件的触发。一个动作状态执行完本状态需要完成的动作就会自发转换到下一个动作状态。

4.3.4 分支与合并

动作流的条件行为用分支与合并表示。在活动图中分支与合并用空心小菱形表示，如图 4.7 所示。

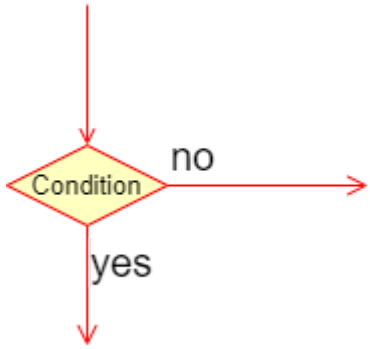

图 4.7　分支

一个分支有一个入转换和两个带监护条件的出转换，出转换的监护条件应当是互斥的，这样可以保证只有一条出转换能够被触发。

一个合并有两个入转换和一个出转换，合并表示从对应的分支开始的条件行为的结束。

4.3.5 分叉与汇合

在活动图中用分叉和汇合来表达并发和同步行为。分叉用于将控制流分为两个或者多个并发运行的分支，而汇合则用于同步这些并发分支，以达到共同完成一项事务的目的。

分叉可以用来描述并发线程，每个分叉可以有一个输入转换和两个或多个输出转换，每个转换都可以是独立的控制流。

汇合代表两个或多个并发控制流同步发生，当所有的控制流都达到汇合点后，控制才能继续往下进行。每个汇合可以有两个或多个输入转换和一个输出转换。

分叉和汇合都使用加粗的水平线段或垂直线段表示，称为同步棒。

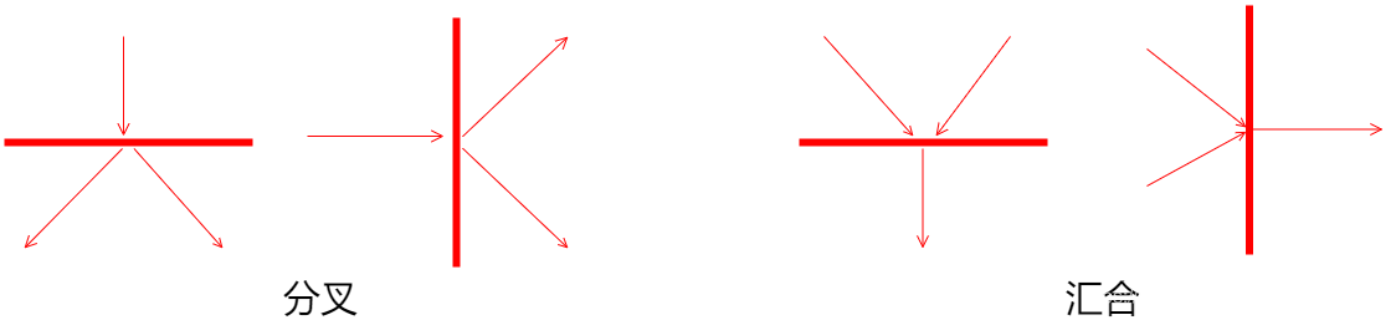

图 4.8　分叉与汇合

4.3.6 泳道

泳道：用矩形框来表示，属于某个泳道的活动放在该矩形框内，将对象名放在矩形框的顶部，表示泳道中的活动由该对象负责。

每个活动只能明确地属于一个泳道。

泳道没有顺序，不同泳道中的活动既可以顺序进行也可以并发进行，动作流和对象流允许穿越分隔线。

泳道可以提高活动图的可读性,可用于建模某些复杂的活动图。

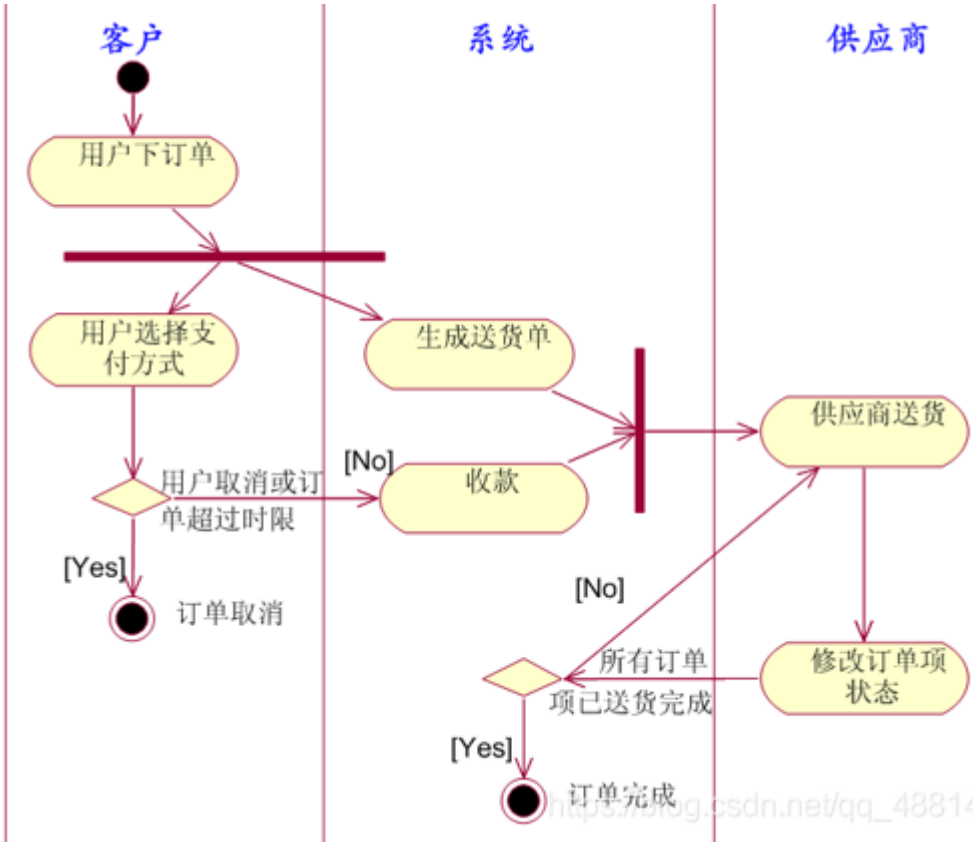

图 4.9　泳道

4.3.7 对象流

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对对象的影响。

对象流中的对象特点：

1. 一个对象可以由多个动作操纵。
2. 一个动作输出的对象可以作为另一个动作输入的对象。
3. 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。

对象流用带有箭头的虚线表示。如果箭头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。施加的影响包括创建、修改和撤销等，该对象是动作的输出。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，该对象是动作的输入。

活动图中对象用矩形表示，矩形内是该对象的名称，名称下的方括号表示对象此时的状态。

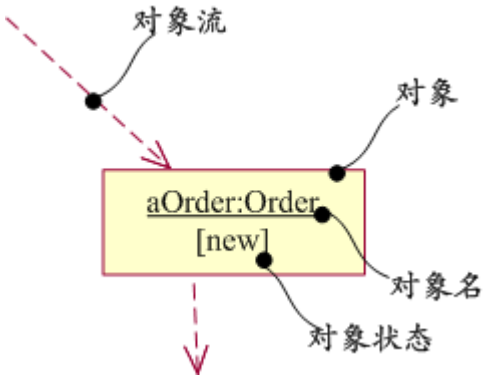

图 4.10　对象流

5 交互图

5.1 顺序图

顺序图是强调消息时间顺序的交互图。顺序图展示参与交互的对象和交换的消息序列。主要用于用例的逻辑建模，描述用例实现。

顺序图将交互关系表示为一个二维图。其中，纵轴是时间轴，时间沿竖线向下延伸。横轴代表了在协作中各独立的对象。

顺序图的组成：

1. 对象
2. 生命线
3. 消息
4. 激活

将对象置于顺序图的顶部意味着在交互开始的时候对象就已经存在了。如果对象的位置不在顶部，那么表示对象是在交互的过程中被创建的。如果要撤销一个对象，只要在其生命线终止点放置一个“X”符号即可，该点通常是对删除或取消消息的回应。

生命线是一条垂直的虚线，表示对象在一段时间内存在。每个对象都有自己的生命线。生命线是一个时间线，从顺序图的顶部一直延伸到底部，所用的时间取决于交互持续的时间。

消息定义的是对象之间某种形式的通信。可以是一个对象（发送者）向另一个对象或几个对象（接收者）发送信号，或一个对象（发送者或调用者）调用另一个对象（接收者）的操作。

在 UML 中，消息使用箭头来表示，箭头的类型表示了消息的类型。

1. 简单消息：表示简单的控制流，用于描述消息如何在对象间进行传递，而不考虑通信的细节。
2. 同步消息：表示嵌套的控制流。通常表示操作调用，调用者发出消息后必须等待消息返回，只有当处理消息的操作执行完毕后，调用者才能继续执行自己的操作。
3. 异步消息：表示异步控制流，调用者发出消息后不用等待消息的返回即可继续执行自己的操作。

激活表示该对象被占用以完成某个任务。在 UML 中，为了表示对象是激活的，可以将该对象的生命线拓宽成为矩形。其中的矩形称为激活条或控制期。当收到消息时，接收对象立即开始执行活动，即对象被激活了。

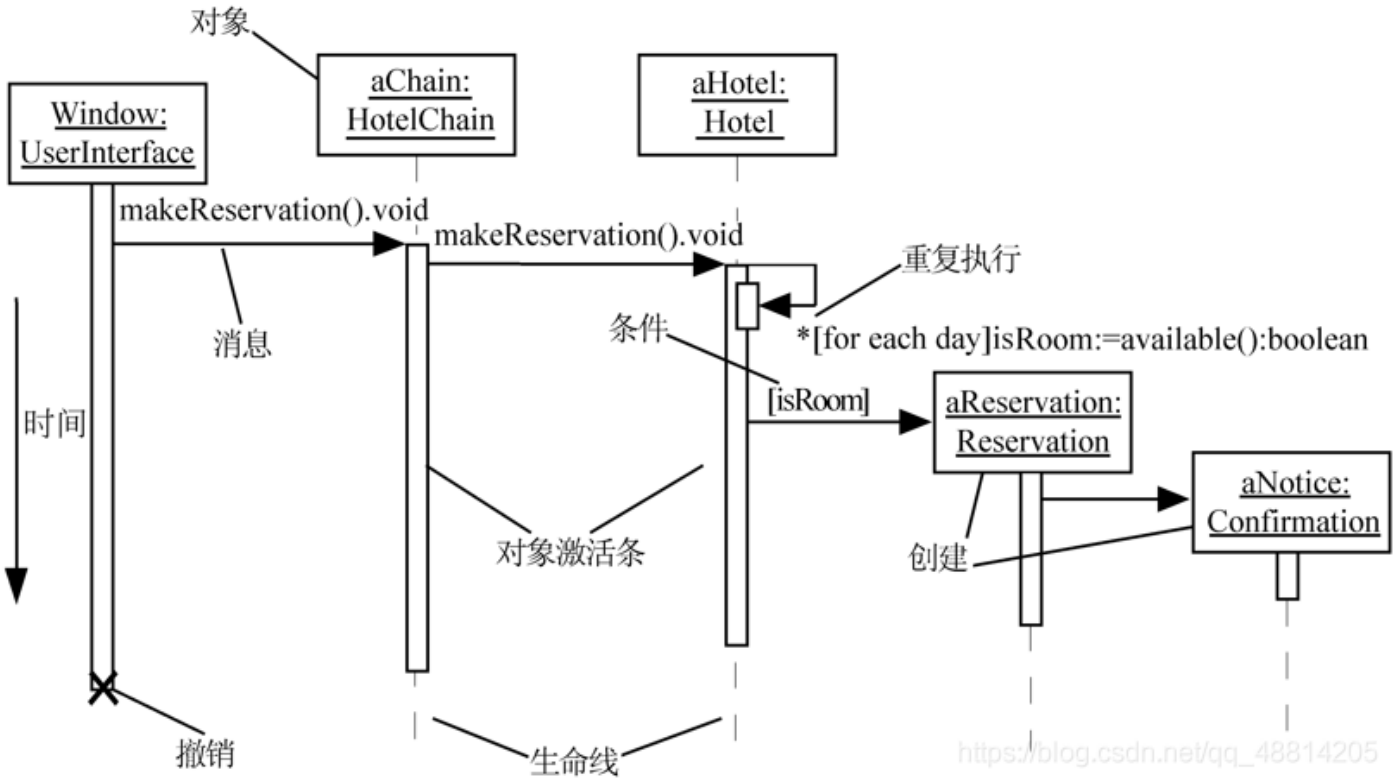

图 5.1　顺序图

有 3 种方式表示消息的重复执行：

1. 在重复执行的消息前添加符号`*`，并在紧跟其后的中括号`[]`内写明具体的条件。（如上图所示）
2. 在时间轴上用注释来指明消息的重复执行。（如下图左侧所示）
3. 使用一个矩形框将重复执行的消息框起来，并在附件的中括号`[]`内指明重复执行的条件。（如图 5.2 右侧所示）

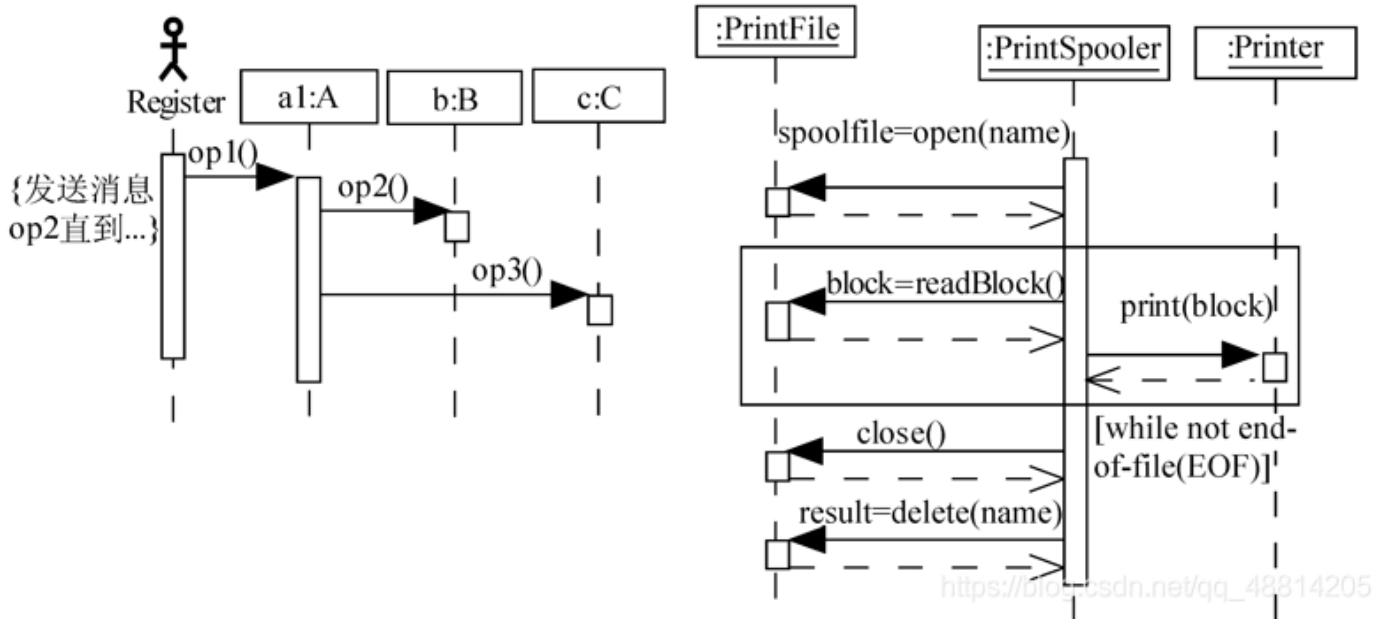

图 5.2　重复执行

顺序图中某个对象消息的传入对应于该类对象所具有的操作。例如：

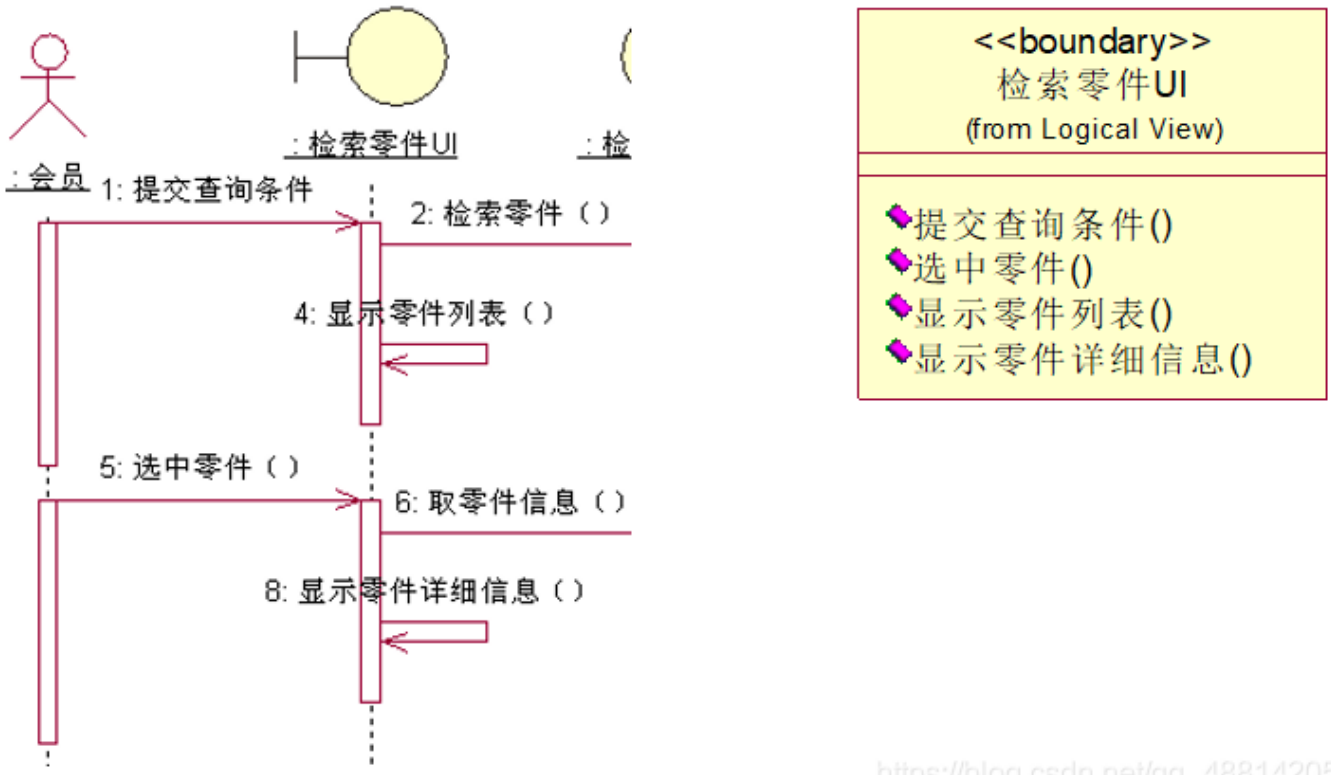

图 5.3　传入消息与对象的操作

5.2 通信图（协作图）

通信图强调参与交互的对象的组织。通信图展示参与交互的对象、对象之间的链接和对象之间传递的消息。

通信图的构成：

1. 对象
2. 链：一条连接两个对象的实线。
3. 消息

通信图与顺序图中的对象的概念是一样的，只不过在通信图中，无法表示对象的创建和撤销，所以对于对象在图中的位置没有限制。

通信图中的消息类型与顺序图中的相同，只不过为了说明交互过程中消息的时间顺序，需要给消息添加顺序号。顺序号是消息的一个数字前缀，是一个整数，由 1 开始递增，每个消息都必须有唯一的顺序号。

消息的编号有两种，一种是无层次编号，它简单直观；另一种是嵌套的编号，它更易于表示消息的包含关系。嵌套的编号可以通过点表示法代表控制的嵌套关系，如 1，1.1，1.2 。

多对象：在通信图中，多对象指的是由多个对象组成的对象集合，一般这些对象是属于同一个类的。当需要把消息同时发送给多个对象而不是单个对象的时候，就使用多对象这个概念。在通信图中，多对象用多个方框的重叠表示。例如：

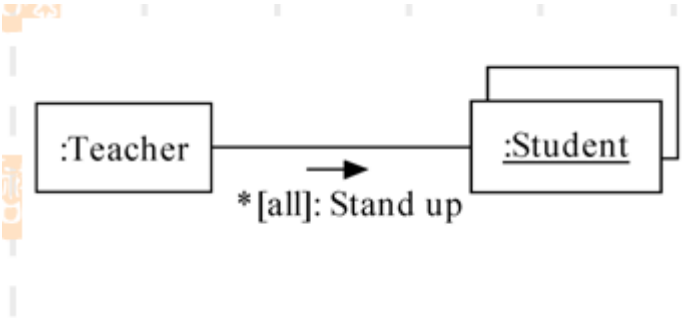

图 5.4　多对象

通信图示例：

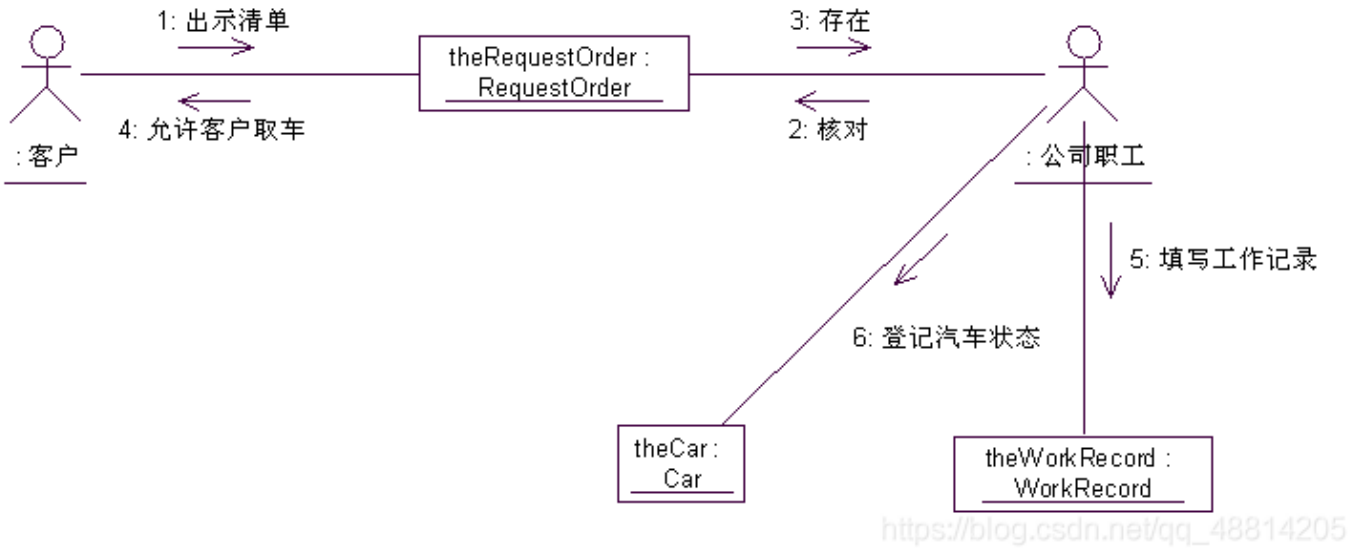

图 5.5　通信图

5.3 顺序图与通信图的比较

相同点：

1. 在语义上是等价的，可以在不丢失任何信息的情况下相互转换。
2. 为系统的动态方面建模。
3. 为用例场景建模。

不同点：

顺序图	通信图
展示显式的消息序列	除了互动之外，还要展示关系
展示事件的执行	更适合于将通信模式可视化
更适合于将整体流程可视化	更适合于将给定对象的所有操作可视化
更适合于实时规范和复杂场景	更容易用于头脑风暴

6 类图

6.1 类

按照其作用，类分为实体类、界面类和控制类三种类型。

1. 实体类用来表示客观实体，一般对应着在业务领域中的客观事物，或者是具有较稳定信息内容的系统元素。实体类的名字用名词或名词短语。
2. 界面类是用来描述系统与外界之间交互的系统要素，也称为边界类。界面类是对外界与系统之间交互的抽象表示，并不表示交互的具体内容或交互界面的具体形式。界面类的名字用名词或名词短语。
3. 控制类表示系统中用来进行调度、协调以及业务处理的系统要素。控制类的名字用动词或动词短语表示。

图 6.1　类的种类

类的属性的格式：`[可见性]属性名[:类型]['['多重性[次序]']'][=初始值][{特性}]`
例：`#studentBirthDay:Date=1999-10-21`

可见性：该属性对外部实体的显现程度。

可见性	符号
public（所有可见）	+
protected（子类及本身可见）	#
private（本身可见）	-
package（包内可见）	~

多重性：表示取值的多少以及有序性。例如，`[0..1]`表示可能有 0 或 1 个值；`[2..* ordered]`表示有 2 个或多个值，有序。

特性：表示属性约束说明。

操作的格式：`[可见性]操作名[(参数列表):返回类型]`
例：`+getName():String`

6.2 类的关系

6.2.1 关联

关联是两个或多个分类器之间的语义关系，用于指定它们的实例之间的连接。

关联是一种结构关系，规定一事物的对象与另一事物的对象相连接。

关联的表示形式：

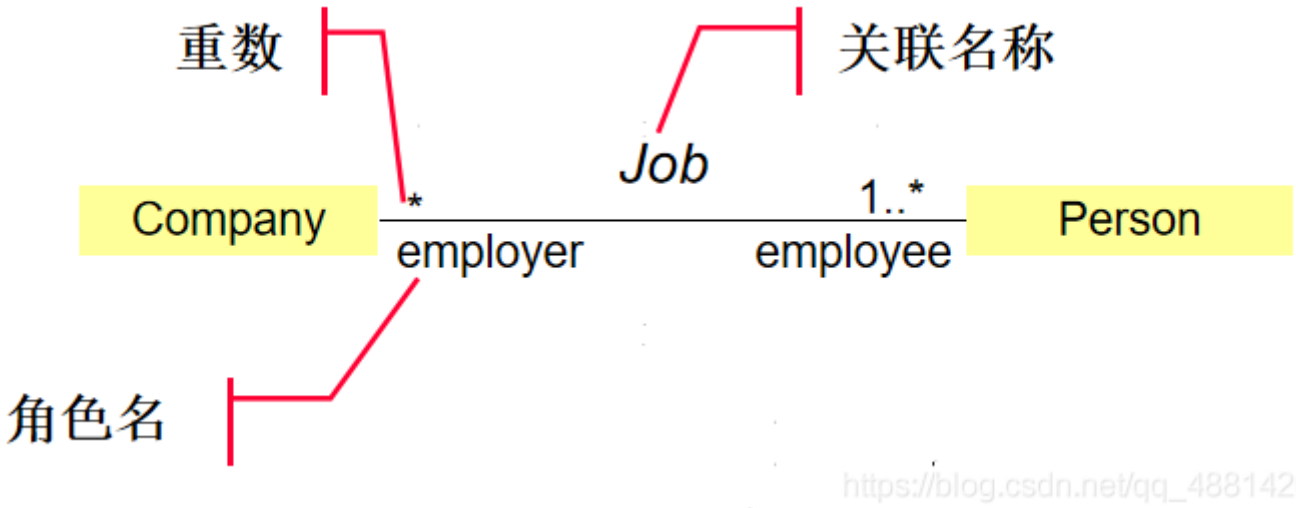

图 6.2　关联

重数是一个类与另一个类的一个实例相关的实例数。例如，在上图中，每个 Company 对象与 1 到多个 Person 对象相关，每个 Person 对象与 0 到多个 Company 对象相关。

关联终端名（即上图中的角色名）的使用是可选的，可加可不加，但是对于同一个类的两个对象之间的关联来说，关联终端名是必需的。关联终端名也可以区分同一对类之间的多重关联。

关联类：是一种关联，也是一种类。描述了关联的属性和操作。例如：

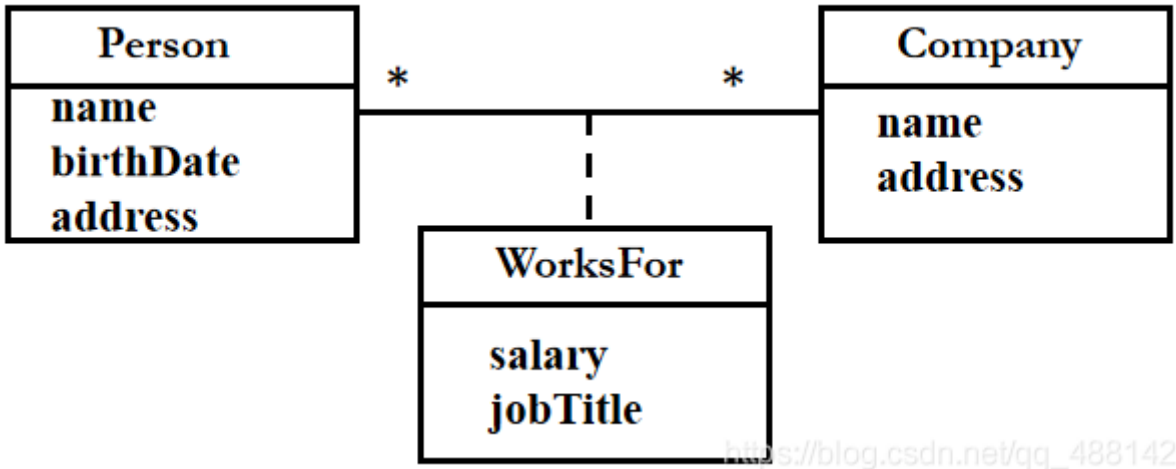

图 6.3　关联类

限定关联：是这样一种关联，其中被称为限定符的属性会消除在“多”关联端上对象的歧义。

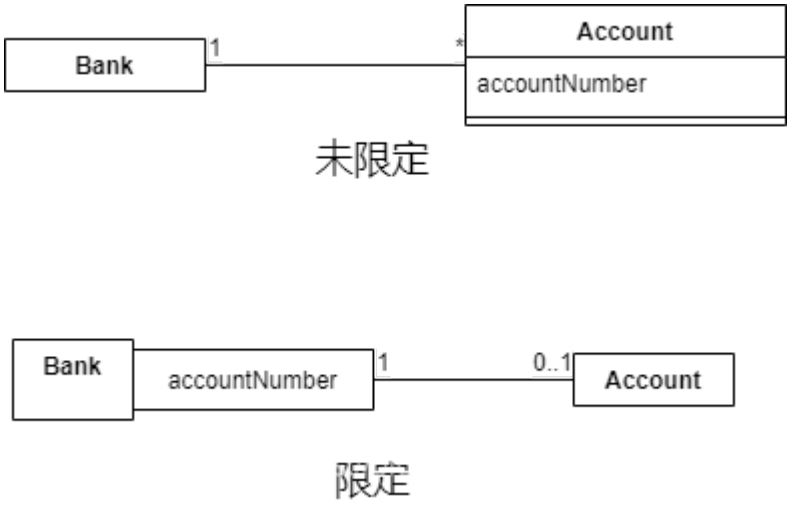

图 6.4　限定关联

n 元关联的 UML 符号是以直线连接相关类的一个菱形。要尽量避免 n 元关联——大部分关联可以分解成带限定符和属性的二元关联。

6.2.2 聚合

聚合是一种特殊的关联形式，它模拟了整体及其部分之间的整体-部分关系。

聚合是“is a part of”关系。

聚合的表示形式：

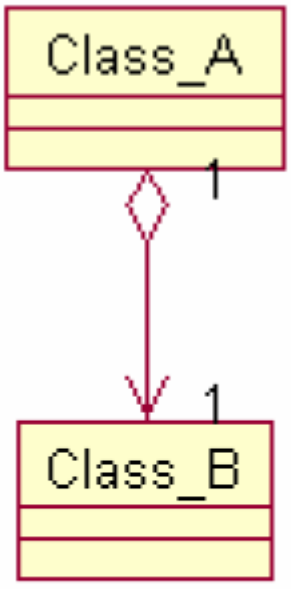

图 6.5　聚合

组合是强语义的聚合，当整体对象消失，部分对象也消失。组合的表示形式为：

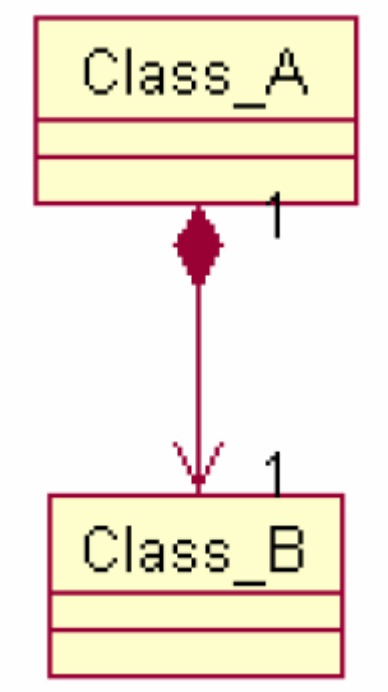

图 6.6　组合

6.2.3 依赖

依赖：如果一个建模元素的变化会影响另一个建模元素，那么二者之间存在依赖关系。

在类图中，当一个类的某个方法中使用了另一个类的对象，或者调用了另一个类的方法，那么这个类就依赖于另一个类。例如：

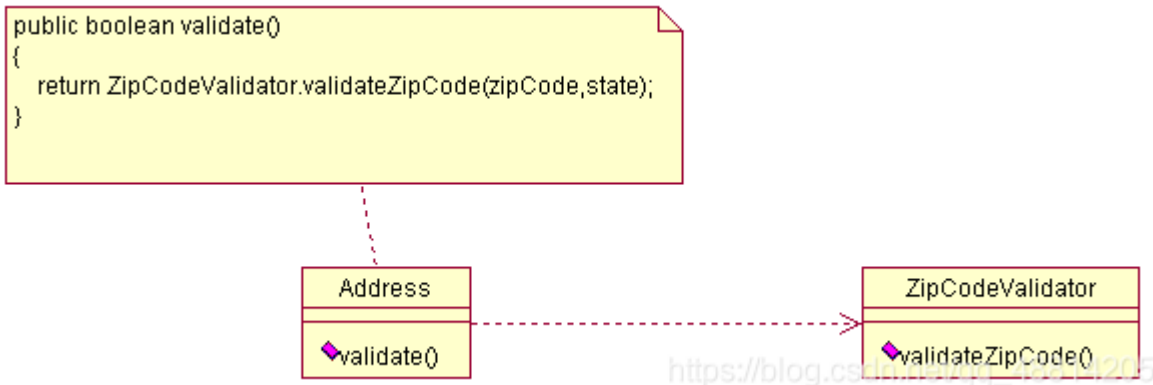

图 6.7　依赖

6.2.4 泛化

泛化是类之间的一种关系，其中一个类共享一个或多个类的结构和/或行为。

泛化定义了子类从一个或多个超类继承的抽象层次结构。

泛化是“is a kind of”关系。

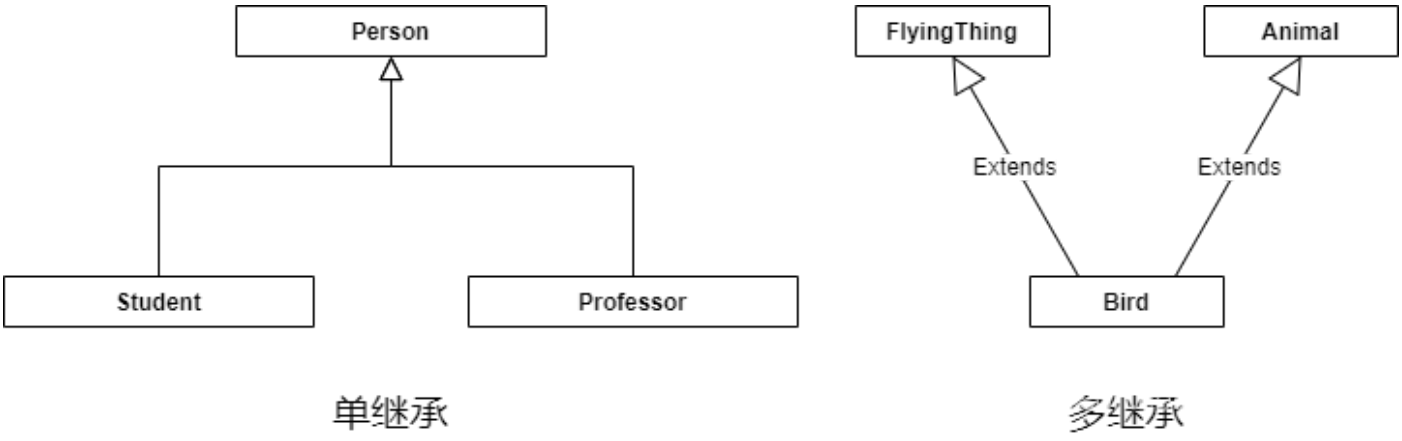

图 6.8　继承

6.2.5 实现

多数情况下，实现关系被用来规定接口和实现接口的类或组件之间的关系。

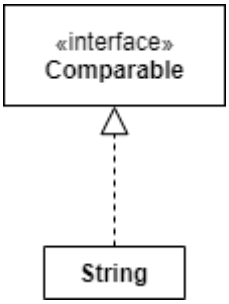

图 6.9　实现

6.3 如何阅读类图

1. 找出类
2. 找出关系
3. 理解多重性
4. 理解属性和方法

6.4 如何构建类图

1. 研究分析问题领域，确定系统需求。
2. 抽取类，明确类的含义和职责，确定类的属性和操作。
3. 确定类之间的关系。
4. 调整和细化类及其关系，解决重复和冲突。

5. 绘制类图，并增加相应说明。

7 其他UML图

7.1 状态图

状态图对动态行为建模。它指定对象可以存在的状态序列，使对象到达这些状态的事件和条件，以及当对象到达这些状态时采取的行动。

一个状态图表示一个状态机。状态图表现从一个状态到另一个状态的控制流。状态图由表示状态的节点和表示状态之间转换的带箭头的直线组成。

7.1.1 状态

状态是指在对象生命周期中满足某些条件、执行某些活动或等待某些事件的一个条件和状况 。

一个状态通常由状态名、入口动作和出口动作、内部转换、子状态和事件等五个部分组成。

状态名表示状态的名字，通常用字符串表示。一个状态的名称在状态图所在的上下文中应该是唯一的。不过，状态允许匿名。 状态的名字通常放在状态图标的顶部。

入口动作和出口动作表示进入或退出这个状态所要执行的动作。入口动作用“entry/要执行的动作”表达，而出口动作用“exit/要执行的动作”表达。

内部转换对事件做出响应，并执行一个特定的活动，但并不引起状态变化，因此不需要执行入口和出口动作。内部转换和自转换不同，虽然两者都不改变状态本身，但是自转换会激发入口动作和出口动作的执行，而内部转换却不会。

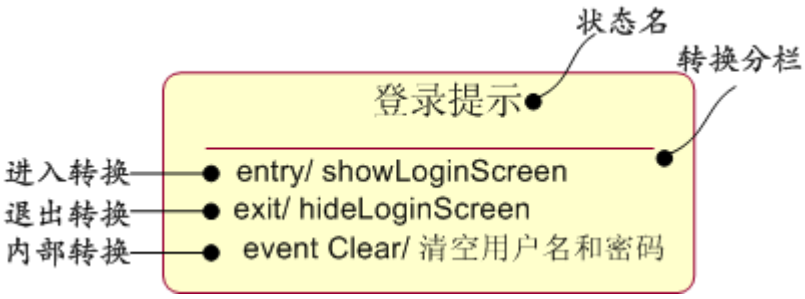

图 7.1　状态

初始状态是创建对象时输入的状态。初始状态是必需的，并且仅允许一个初始状态。初始状态表示为实心圆。

最终状态表示对象的生命结束。最终状态是可选的，并且可能存在多个最终状态。最终状态表示为带圈的实心圆（公牛眼形）。

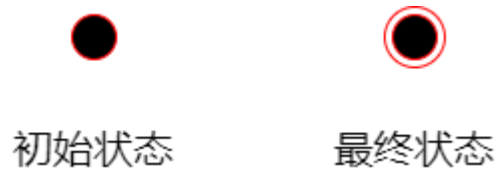

图 7.2　特殊状态

按照有无子结构，可以将状态分为简单状态和组成状态：

1. 简单状态：不包含其他状态的状态。简单状态没有子结构，但它可以具有内部转换、入口动作和出口动作等。
2. 组成状态：可以包含一些嵌套的子状态的状态。组成状态的一个入转换代表对其嵌套子状态区域内的初始状态的入转换，对嵌套子状态区域内的最终状态的转换代表包含它的最终状态的相应活动的完成。

组成状态的子状态又可以分成两类：

1. 顺序子状态：一个组成状态的子状态对应的对象在其生命期内的任何时刻都只能处于一个子状态，即多个子状态之间是互斥的，不能同时存在。
2. 并发子状态：组成状态有两个或者多个并发的子状态机。

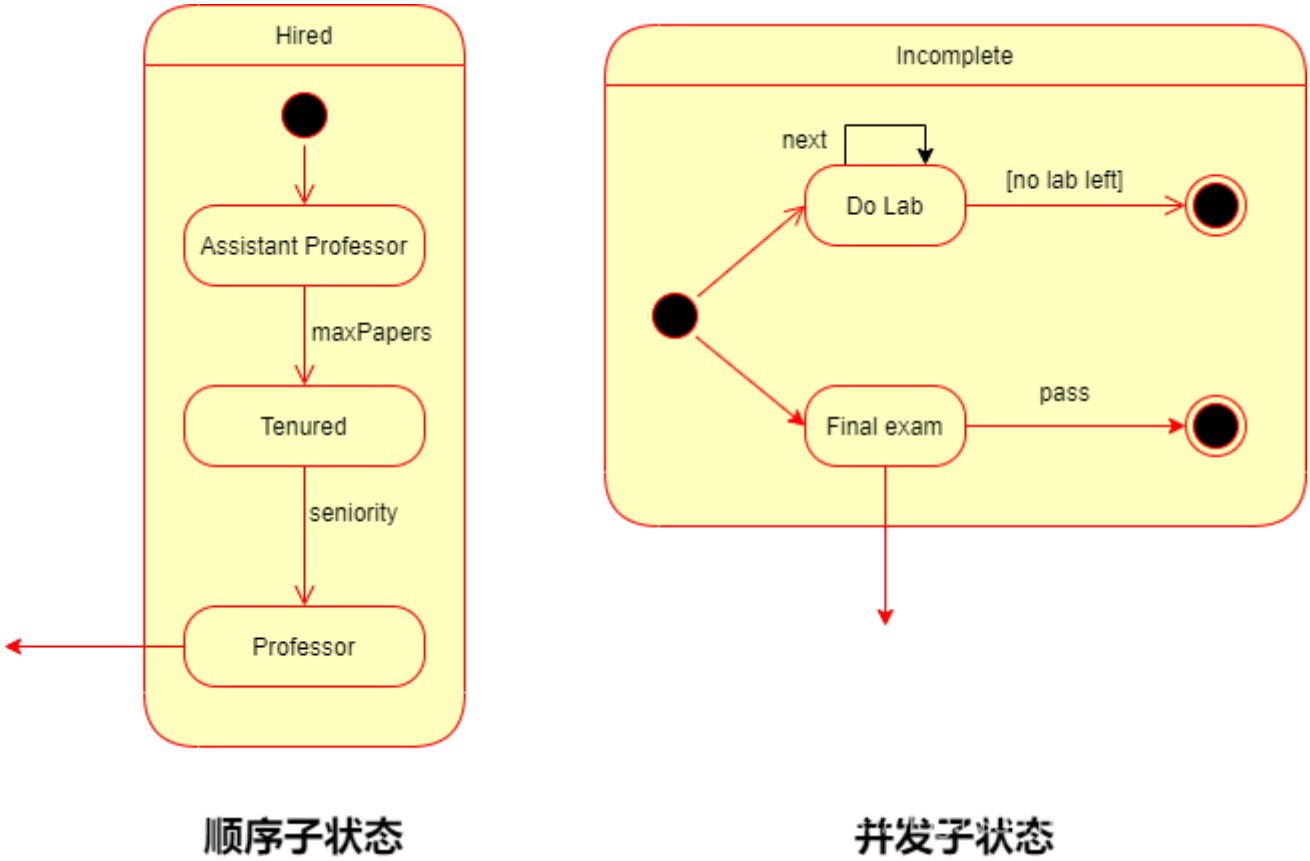

图 7.3　子状态

历史状态：上次离开组成状态时的最后一个活动子状态，用一个包含字母 H 的小圆圈表示。每当转换到组成状态的历史状态时，对象便恢复到上次离开该组成状态时的最后一个活动子状态，并执行入口动作。

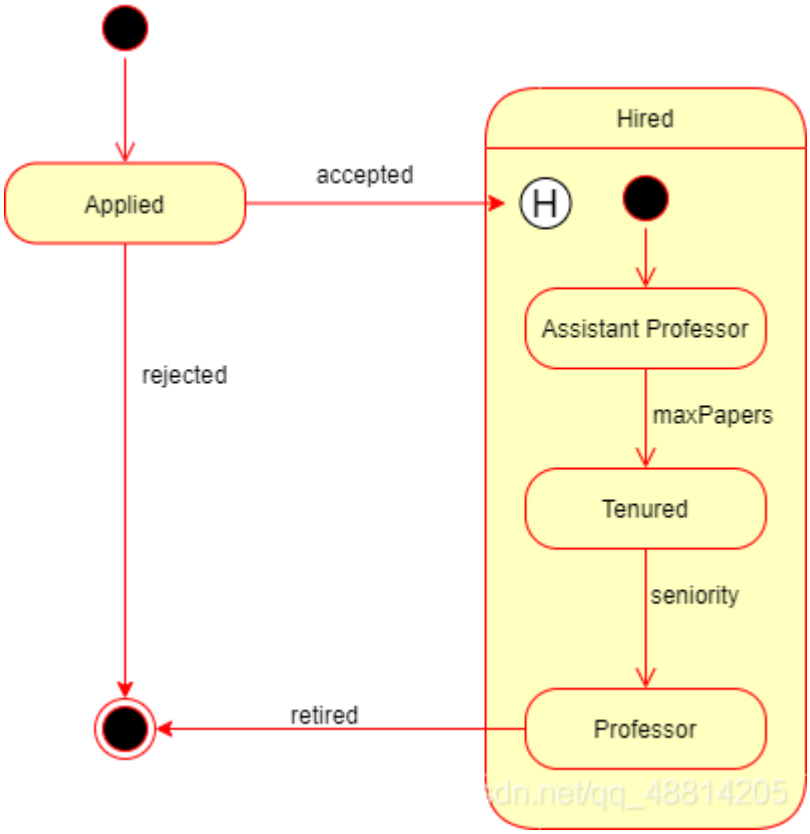

图 7.4　历史状态

7.1.2 转换

转换（transition）表示当一个特定事件发生或者某些条件得到满足时，一个源状态下的对象在完成一定的动作后将发生状态转变，转向另一个称之为目标状态的状态。转换进入的状态为活动状态，转换离开的状态变为非活动状态。

转换的构成：

1. 源状态：受转换影响的状态。
2. 目标状态：当转换完成后对象的状态。
3. 触发事件：能够引起状态转换的事件，包括调用、改变、信号、时间四类事件。
4. 监护条件：布尔表达式，决定是否激活转换。布尔表达式由`[]`括起，放在触发事件后面。当触发事件发生后，求监护条件的值，如果值为真，转换可以触发；如果值为假，转换就不能被触发，如果也没有其他的转换被这个触发事件触发，则事件被忽略。
5. 动作：转换激活时的操作。动作可以包括发送消息给另一个对象、操作调用、设置返回值、创建和销毁对象等。动作是原子的，不可中断的，动作或动作序列的执行不会被同时发生的其他动作影响或终止。整个系统可以在同一时间执行多个动作。

图 7.5　转换

转换的种类：

1. 外部转换：改变对象状态的转换。外部转换对事件做出响应，引起状态变化或自身转换，同时引发一个特定动作。外部转换用从源状态到目标状态的箭头表示。
2. 内部转换：自始至终都不离开本状态。没有出口或入口事件，也就不执行入口和出口动作。
3. 自转换：离开本状态后重新进入该状态。它会激发状态的入口动作和出口动作的执行。

7.1.3 状态图建模步骤

1. 找出适合用模型描述其行为的类
2. 确定对象可能存在的状态
3. 确定引起状态转换的事件
4. 确定转换进行时对象执行的相应动作
5. 对建模的结果进行相应的精化和细化

7.2 组件图

组件图用于描述软件组件以及组件之间的组织和依赖关系。

组件图的作用：

1. 帮助客户理解最终的系统结构
2. 使开发工作有一个明确的目标
3. 复用软件组件
4. 帮助开发组的其他人员理解系统

组件图的组成元素：

1. 组件（component）
2. 接口（interface）
3. 关系（relationship）

除此之外，还可以包括包（package）和子系统（subsystem），它们有助于将系统中的建模元素组织成更大的建模元素。

7.2.1 组件

组件是系统中遵从一组接口且提供实现的一个物理部件，通常指开发和运行时类的物理实现。

组件表示法：

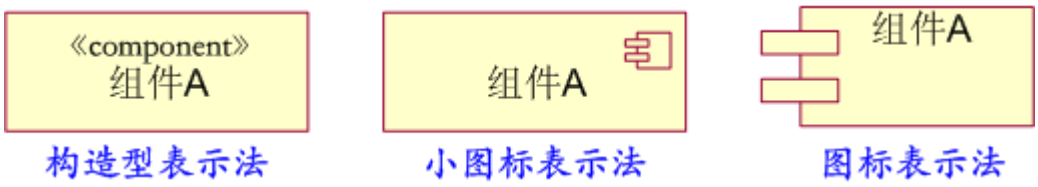

图 7.6　组件表示法

组件与类的相同点：二者都有名称，都可以实现一组接口，都可以参与依赖、泛化和关联关系，都可以被嵌套，都可以有实例，都可以参与交互。

组件与类的不同点：

1. 类表示逻辑抽象，而组件表示存在于计算机中的物理抽象。
2. 组件表示的是物理模块而不是逻辑模块，与类处于不同的抽象级别。
3. 类可以直接拥有属性和操作；而一般情况下，组件仅拥有只能通过其接口访问的操作。

组件的类型：

1. 实施组件（Deployment Component）。如 DLL、EXE、ActiveX 控件和 JavaBean 组件等。
2. 工作产品组件（Work Product Component）。
3. 执行组件（Execution Component）。如由 DLL 实例化形成的 COM+对象。

7.2.2 接口

接口是一组用于描述类或组件的一个服务的操作。它是一个被命名的操作的集合，与类不同，它不描述任何结构（因此不包含任何属性），也不描述任何实现（因此不包括任何实现操作的方法）。

图 7.7　接口

组件的接口可以分为两种类型：

1. 导出接口（expert interface）：为其他组件提供服务的接口。一个组件可以有多个导出接口。
2. 导入接口（import interface）：在组件中所用到的其他组件所提供的接口。一个组件可以使用多个导入接口。

7.2.3 关系

组件图中使用最多的是依赖和实现关系。

组件图中的依赖关系使用虚线箭头表示，如图所示。

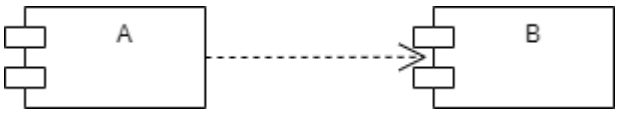

图 7.8　依赖关系

实现关系使用实线表示。实现关系多用于组件和接口之间。

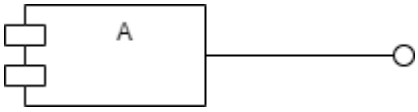

图 7.9　实现关系

组件接口表示法：

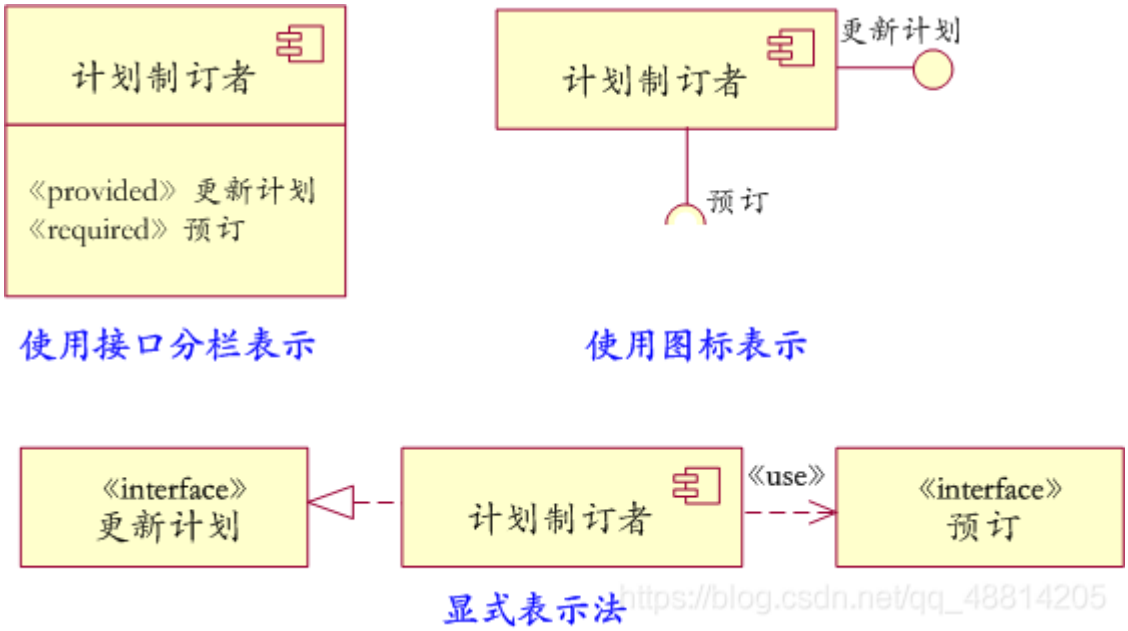

图 7.10　组件接口表示法

7.2.4 组件图建模步骤

1. 对系统中的组件建模。
2. 定义相关组件提供的接口。
3. 对它们间的关系建模。
4. 对建模的结果进行精化和细化。

7.3 部署图

部署图展示运行时处理节点的配置、这些节点之间的通信链路以及驻留在其上的已部署构件。

节点：表示运行时计算资源，通常至少有内存和处理能力。

节点的种类：

1. 设备：具有处理能力的物理计算资源，可以嵌套。
2. 执行环境：表示特定的执行平台。

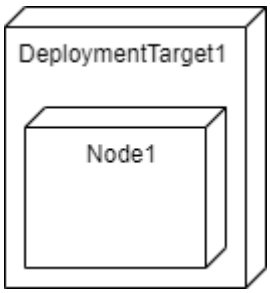

图 7.11　节点

连接：表示沟通机制。包括物理介质、软件协议。

部署图示例：

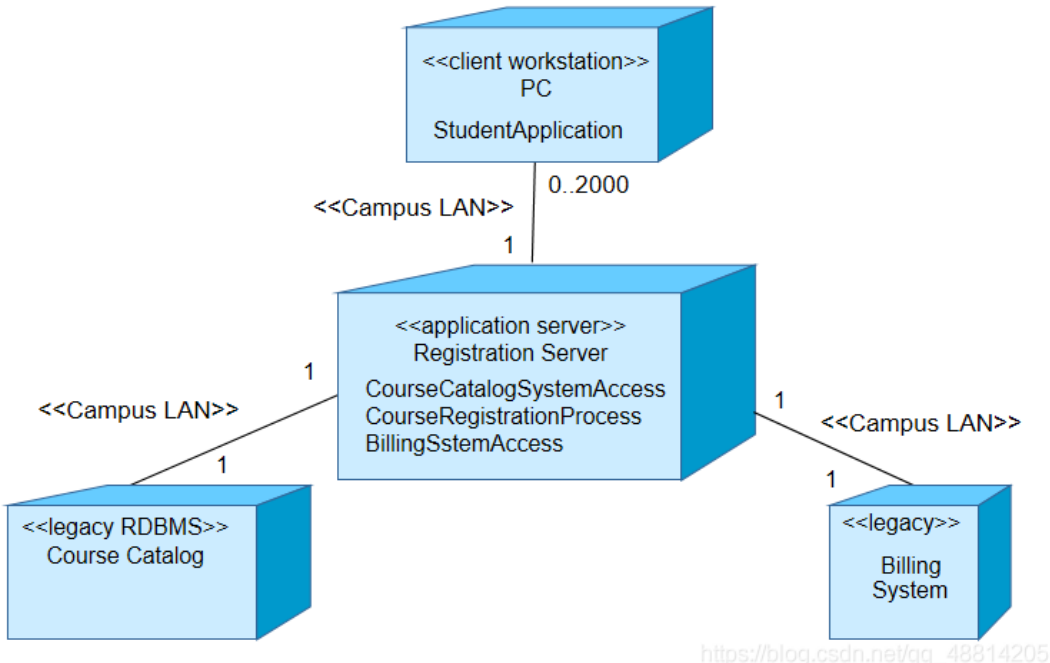

图 7.11　部署图