4.1 用例和用例图的概念

用例模型的基本组成部分有用例、角色（或参与者）和系统。用例用于描述系统的功能， 也就是从用户的角度来说，系统具体应包含哪些功能，帮助分析人员理解系统的行为，它是 对系统功能的宏观的、整体的描述，一个完整的系统通常包含许多用例，每个用例具体说明 应完成的功能；参与者是指那些与系统进行交互的外部实体，通常它是系统的一个用户，但 它也可以是其他系统或硬件设备，总之凡是需要与系统进行交互的任何实体都可以称作参 与者，用例往往必须向参与者传递一些数值.这些数值是参与者在系统中获得的信息。

在用例模型中系统仿佛是实现各种用例的一个“黑盒子”，用户只关心该系统实现了哪 些功能，并不需要知道其内部的具体实现细节，比如系统是如何做的用例，是如何实现的。 用例模型主要应用在工程开发的初期阶段，在进行系统需求分析时使用，通过分析描述使开 发者明确需要开发的系统功能。

**使用用例的主要目的如下。**

（1）明确系统应具备什么功能，这些功能是否满足客户的基本需求，并与系统开发人员达成一致

（2）为系统的功能提供清晰一致的描述，用例模型应用于系统开发的整个过程，为后阶

段的系统设计和开发工作打下良好的基础。

（3）为系统测试打下基础，可以用于验证最终实现的系统所完成的功能是否符合客户最初

需求。

（4）通过从需求的功能用例出发跟踪进入到系统中具体实现的类和方法，可以检查其是否正确。

例如，通过下面这种方法可以简化对系统的修改和扩展：首先修改用例模型，针对受到影响的用例，找到相应的系统设计和实现部分，对其进行相应的修改即可。

用例图(Use Case Diagram)是显示一组用例、参与者以及它们之间关系的一种图。

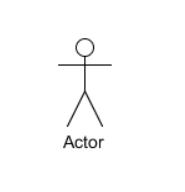
用例图从用户的角度而不是开发者的角度来描述对软件产品的需求，分析产品所需的 功能和动态行为。用例图常用来对需求进行建模，用例图在系统的整个分析、设计和开发阶 段是非常重要的，它的正确与否直接影响到客户对最终实现的产品的满意度。用例图被广 泛使用在各种开发活动中，但它最常用于描述系统以及子系统。

用例图的主要作用如下。

1. 用来描述将要开发系统的功能需求和系统的使用场景。
2. 作为设计和开发过程的基础，促进各阶段开发工作的进展。
3. 用于验证与确认系统需求。

用例图中的各个 组成元素。[1]

1. 参与者(Actor)：也称为角色，它代表系统的用户。



参与者的作用：

（1）建立系统的外部用户模型。

（2）对系统边界之外的对象进行描述。

1. 系统边界(System Scope):它确定系统的范围。



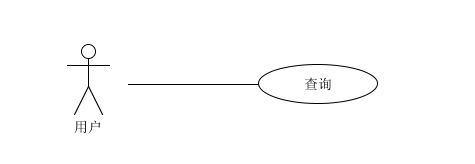
1. 用例(Use Case)：它代表系统提供的服务。



用例是代表系统中各个项目相关人员之间根据系统的行为所达成的契约。用例描述了 在不同条件下，针对某一项目相关人员的请求，系统对其做出的响应。也就是说用例指的是 对一组动作的描述，系统通过执行这些动作将对用例的参与者产生可以看到的结果，用来描 述参与者可以感受到的系统服务或功能。

使用用例进行系统的需求分析时具有如下一些特点。

1. 用例是从系统的使用角度描述系统中的信息，即在系统的外部所能看到的系统的 功能，而不是考虑系统内部对该功能的具体实现方式。
2. 用例描述了用户提出的一些可见需求，对应一个具体的用户目标。使用用例可以 促进与用户的沟通，正确地理解需求，同时也可以用来划分系统与外部实体的界限，是面向 对象分析与设计的起点，是类、对象、操作的来源。
3. 用例通常由某个参与者来执行。
4. 用例把执行的结果反馈给参与者。
5. 用例在功能上具有完整性，即它从参与者接受输入,产生的结果最终再输出给参与者。
6. 关联(Association)：它表示参与者与用例间的关系。



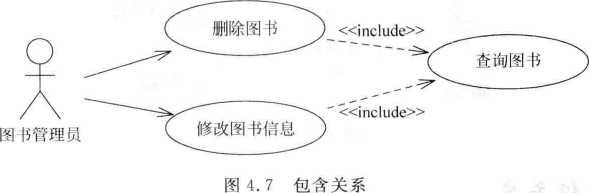
**用例描述**

从软件开发的角度，用例就是需求的文字性描述，主要是说明系统如何工作的功能性或 行为性需求。用例图只是简单地用图形的方式描述了一下系统。实际上，用例是文本形式， 不是图形。用例是作为人与人之间，尤其是没有受过专门培训的人员之间互相交流的一种 手段。因此，编写用例的首选形式通常是简单的文本。因此对于每个用例，还需要有详细的了解，这时候就需要编写用例描述

**4.2用例之间的可视化关系**

包含关系指的是两个用例之间的关系，其中一个用例（称为基本用例，Base Use Case） 的行为包含另一个用例（称为包含用例.Inclusion Use Case）的行为。也就是说基本用例会 用到包含用例，表示基本用例中重用包含用例中的步骤。在UML图中，使用带虚线箭头表 示，并在线上标有« include »,如图4. 7所示。

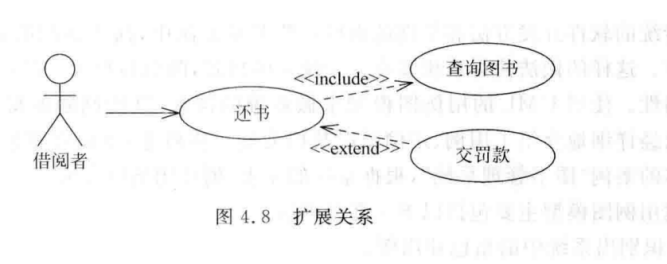
在包含关系中，箭头的方向是从基本用例到包含用例，也就是说，基本用例是依赖于包含用例的

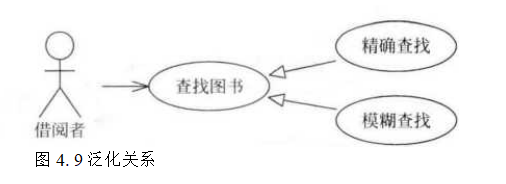


扩展(extend)关系的基本含义与泛化关系类似。extend关系是对基本用例的扩展，基 本用例是一个完整的用例，即使没有子用例的参与，也可以完成一个完整的功能。extend 的基本用例中将存在一个扩展点，只有当扩展点被激活时，子用例才会被执行。在扩展关系 中，对于扩展用例(Extension Use Case)有更多的规则限制，即基本用例必须声明若干“扩展 点"(Extension Point),而扩展用例只能在这些扩展点上增加新的行为和含义。扩展关系是 从扩展用例到基本用例的关系，它说明扩展用例定义的行为如何插入到基本用例定义的行 为中。也就是说,扩展用例并不在基本用例中显示。

在以下几种情况下，可使用扩展用例。

1. 表明用例的某一部分是可选的系统行为(这样，就可以将模型中的可选行为和必选 行为分开)。
2. 表明只在特定条件(如例外条件)下才执行的分支。
3. 表明可能有一组行为，其中的一个或多个可以在基本用例中的扩展点处插入。所 插入的行为和插入的顺序取决于在执行基本用例时与主角进行的交互。



泛化关系指的是一般与特殊的关系。当多个用例共同拥有一种类似的结构和行为的时 候，可以将它们的共性抽象成为父用例，其他的用例作为泛化关系中的子用例。在用例的泛 化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系，

**分组，关系**

在一些用例图中，用例的数目可能很多，这时就需要把这些用例组织起来。这种情况在 一个系统包含很多子系统时就会出现。另一种可能就是，当你按顺序和用户会谈，收集系统需 求时，每个需求必须用一个单独的用例来表达，这时就需要某种方式来对这些需求进行分类。

最直接的方法就是把相关的用例放在一个包中组织起来。一组用例可以放在一个文件 夹中。

综上所述，用例之间存在着一定的关系，这些关系既有联系又有区别，在扩展关系中，基 本用例是一个完整的用例，即是可以独立存在的用例。一个基本用例执行时，可以执行，也 可以不执行扩展部分。

在包含关系中，基本用例可能是也可能不是一个完整的用例。在执行基本用例时，一定 会执行包含用例(Inclusion Use Case)部分。

如果需要重复处理两个或多个用例时，可以考虑使用包含关系，实现一个基本用例对另 一个用例的引用。

当处理正常行为的变型而且只是偶尔描述时，可以考虑只用泛化关系。

当描述正常行为的变型而且希望采用更多的控制方式时，可以在基本用例中设置扩展 点，使用扩展关系。

**4.3** 用例图建模技术及应用

创建用例图模型主要包括以下三部分内容。

1. 识别出系统中的角色和用例。
2. 区分用例之间的先后次序。
3. 创建用例图模型结构。

**5.1**类图和对象图概述

**类图**

类是对一组具有相同属性、操作、关系和语义的对象的抽象。主要包括名称部分 (Name)、属性部分(Attribute)和操作部分(Operation) o 在 UML 中类用一个矩形框表示，它包含三个区域，最上面是类名、中间是 类的属性、最下面是类的方法

1.名称

每个类都必须有一个能和其他类进行区分的名称，类的名称 部分是不能省略的，其他组成部分可以省略。名称(Name)是一个 文本串，类的命名要求为由字符、数字、下划线组成的唯一的字符串即可。

2.属性

属性描述了类在软件系统中代表的事物(即对象)所具 备的特性。类可以有任意数目的属性，也可以没有属性。 类如果有属性，则每一个属性都必须有一个名字

另外还可以有其他的 描述信息，如可见性、数据类型、默认值等

3.操作

操作是对类的对象所能做的事务的一个抽象。一个类可以有任意数量的操作或者根本 没有操作。类如果有操作'则每一个操作也都有一个名字，其他可选的信息包括可见性、参 数的名字、参数类型、参数默认值和操作的返回值的类型等。

**对象图**

对象指的是一个单独的、可确认的物体、单元或实体，它可以是具体的也可以是抽象的， 在问题领域里有确切定义的角色。换句话说，对象是边界非常清楚的任何事物。一个对象 通常包含以下几部分：标识（名字）、状态（属性）、行为（方法，事件）

对象和类的区别如下。

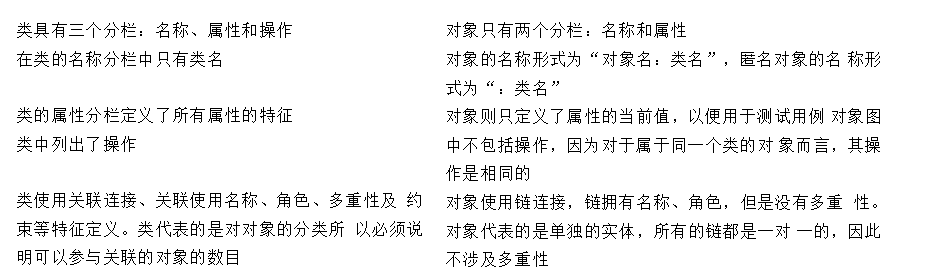
（1）对象是一个存在于时间和空间中的具体实体，而类仅代表一个抽象，抽象出对象的 “本质”。

（2）类是共享一个公用结构和一个公共行为对象集合。

（3）类是静态的，对象是动态的；类是一般化，对象是个性化；类是定义，对象是实例； 类是抽象、对象是具体。

对象图（Object Diagram）描述的是参与交互的各个对象在交互过程中某一时刻的状 态。对象图可以被看作是类图在某一时刻的实例。在UML中，对象图使用的是与类图相 同的符号和关系，因为对象就是类的实例。对象图主要包括对象名、属性。

类图和对象图的区别：



**接口**

接口 (Interface)是描述类的部分行为的一组操作，它也是一个类提供给另一个类的一 组操作。通常接口被描述为抽象操作，也就是只用标识(返回值、操作名称、参数表)说明它 的行为，而真正实现部分放在使用该接口的对象中，也就是说接口只负责定义操作而不具体 地实现。

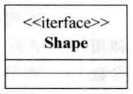
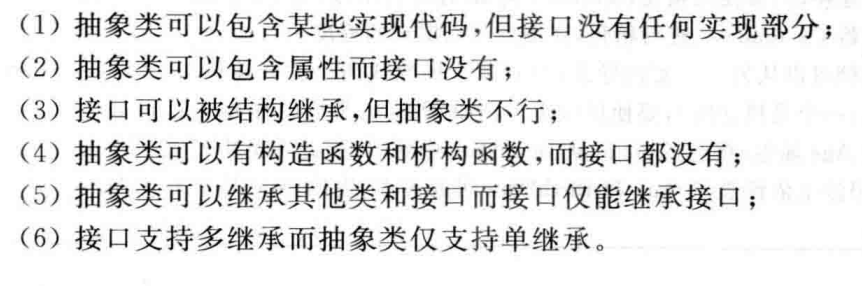
接口的模型表示法和类大致相同，都是用一个矩形图标来代 表。和类的不同之处在于，接口只是一组操作，没有属性。在 UML图形上，接口的表示和类图的表示类似，只是在最上面的一 层类名前加描述<<interface>>，或是简化表示，用一个圆圈表示

图5.5接口

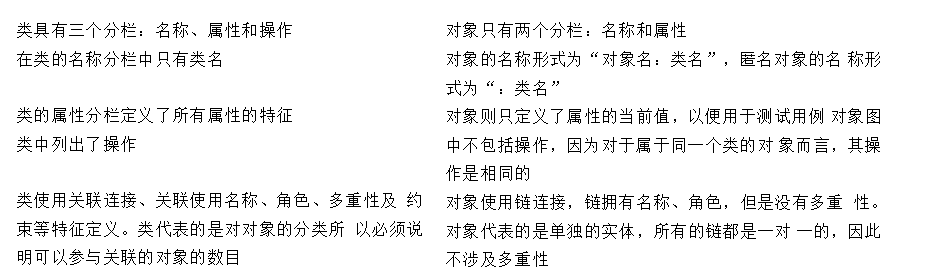
**抽象类**

抽象类是包含一种或多种抽象方法的类，它本身不需要构造实例。定义抽象类后，其他 类可以对它进行扩充并且通过实现其中的抽象方法，使抽象类具体化。

接口与抽象类非常相似，例如，两者都不能产生实例对象，都可以作为一种定义使用。 但接口和抽象类仍有本质的不同，这些不同包括以下几个方面。

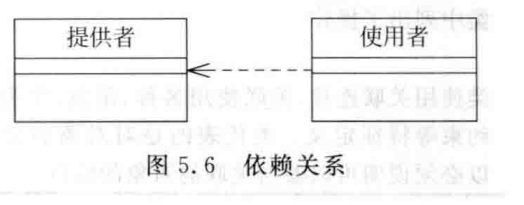


提问：类图和对象图的区别



**5.2类之间的关系**

**依赖关系**

****

依赖关系（Dependency）表示两个或多个模型元素之间语义上的关系。它表示了这样 一种情形，对于一个元素（服务提供者）的某些改变可能会影响或提供消息给其他元素（使用 者），即使用者以某种形式依赖于其他类元。在 UML图形上，把依赖画成一条有向的虚线，指向被 依赖的事物。当要指明一个事物使用另一个事物 时，就使用依赖。UML定义了 4种基本依赖，分别是使用依赖、抽象依赖、授权依赖和绑定依赖。

（1）使用依赖。使用依赖是一种非常直接的关系，它通常表示使用者使用服务提供者 所提供的服务实现它的行为。

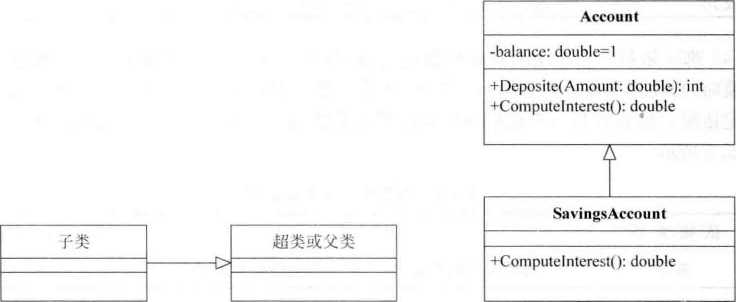
（2）抽象依赖。抽象依赖建模表示使用者和提供者之间的关系，它依赖于在不同抽象 层次上的事物。共有三种类型的抽象依赖：跟踪依赖（« trace »）,精化依赖（« refine »）和 派生依赖（« derive »）

（3）授权依赖。授权依赖表达了一个事物访问另一个事物的能力。提供者可以规定使用者 的权限，这是提供者控制和限制对其内容访问的方法。主要有三种类型的授权依赖：访问依赖 (« access»)、导入依赖(《 import»和友元依赖(《 friend »)

（4）绑定依赖。它表明对目标模板使用给定的实际参数进行实例化。当对模板类的细 节建模时，要使用绑定(«bind»)o例如，模板容器类和这个类的实例之间的关系被模型化 为绑定依赖。绑定包括一个映射到模板的形式参数的实际参数列表

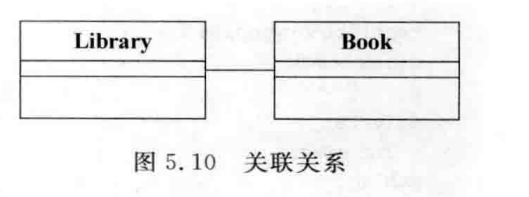
**泛化关系**

泛化关系(Generalization)是一种存在于一般元素和特殊元素之间的分类关系，它只使 用在类型上，而不是实例上。在类中，一般元素被称为超类或父类，而特殊元素被称为子类。 在UML中，泛化关系用一条从子类指向父类的空心三角箭头表示



**关联关系**

关联关系(Association)是一种结构关系，它指明一个事物的对象与另一个事物的对象 之间的联系。也就是说，关联描述了系统中对象或实例之间的离散连接。给定一个连接两 个类的关联，可以从一个类的对象联系到另一个类的对象。关联的两端都连接到一个类在 理论上也是合法的。在UML图形中，关联关系用一条连接两个类的实线表示。

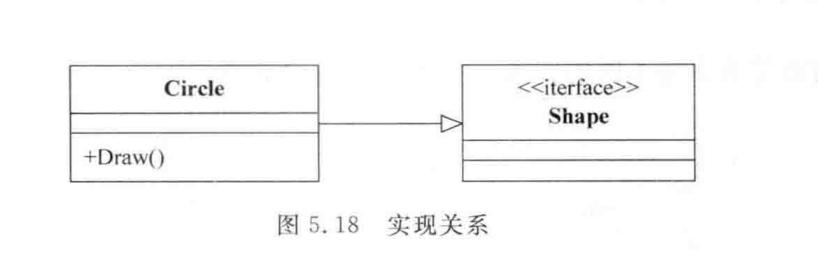


在UML中，有4种可应用到关联的基本修饰：关联名、关联端的角色、关联端的多重性和聚合。

**实现关系**

实现将一种模型元素与另一种模型元素连接起来，比如类和接口。泛化和实现关系 都可以将一般描述与具体描述联系起来。泛化将同一语义层上的元素连接起来，并且通 常在同一模型内。实现关系则将不同语义层内的元素连接起来，通常建立在不同的模 型内。

实现关系通常在两种情况下被使用：在接口与实现该接口的类之间；在用例及实现该用例的协作之间。



5.3 类图的建模技术及应用

1. 概念层类图

概念层的类图描述的是现实世界中对问题领域的概念理解，类图中表达的类与现实世 界的问题领域有着明显的对应关系，类之间的关系也与问题领域中实际事物的关系有着明 显的对应关系。在概念层类图阶段很少考虑或者几乎不需要考虑类的实现问题。

概念层类图中的类和类关系和最终的实现类并不一定有直接和明显的对应关系，在概念层上，类图着重于对问题领域的概念化理解，而不是实现。因此，类名通常都是问题领域中实际事物的名称，并且独立于具体的编程语言。

1. 说明层类图

在说明层阶段主要考虑的是类的接口部分，而不是实现部分。这个接口可能因为实现环境、运行特性等有多种不同的实现。

3.实现层类图

真正需要考虑类的实现问题是在实现层类图阶段。提供实现的细节，在实现层阶段的 类的概念才是真正的严格意义上的类。它揭示了软件 实体的构成情况。说明层的类有助于人们对软件的理解，而实现层的类是最常用的

建立类图的步骤如下。

（1） 研究分析问题领域，对系统进行需求分析，确定系统需求。

（2） 确定系统中的类，明确类的含义和职责以及确定类的属性和操作。

（3） 最后确定类之间的关系。

几种类的识别方法

1） 名词识别法

2） 从用例中识别类

1. 使用CRC分析法
2. 领域分析法

**参考资料：**

1. **UML建模——用例图**

**https://www.cnblogs.com/lcword/p/10472040.html**