第3章 UML介绍

特别说明：如果具有面向对象和 UML 基础知识或有应用经验，则可以跳过本章，直接阅读后面章节。

面向对象的分析方法是目前较为流行的系统分析方法，关于面向对象分析方法的设计描述语言也有许多种类，但是，目前较为流行的和得到业界认可面向对象描述建模语言是 UML，考虑到书籍结构的完整性，笔者阅读了一些关于 UML 介绍方面的资料，整理出来供大家学习，如果读者已经具备了这方面的知识，可以跳过本章。

统一建模语言(Unified Modeling Language，UML)是用来对软件系统进行可视化建模的一种描述语言。UML 为面向对象的开发分析方法系统提供了包含可视化的产品说明和编制文档的一种标准语言，UML 是一种功能强大且得到业绩认可的建模语言。

本章对 UML 及相关知识进行了概要描述，详细内容将在后续章节中逐步开展。

3.1 面向对象介绍

3.1.1 面向对象产生背景

关于面向对象这方面的文章和专著虽然非常多，但像数学公式那样具有逻辑性的定义很少。当然，面向对象的本来就是属于应用性较强的思想，所以没有必要定义得那么明确，因为定义不明确但是应用得可能也很好。

面向对象是在结构化设计方法出现很多问题情况下而出现的。结构化分析方法解决问题的视角是设计功能求解的方法。结构化分析方法将应用程序看成实现某些特定任务的功能模块，其中子过程是实现某项具体操作的底层功能模块。并且用数据结构描述处理数据的组织形式，用算法描述操作过程。在系统过于复杂的情况下，必然会暴露出一系列问题。

从解决问题域的视角出发，我们观察问题的视角通常是从客体出发的，客体的属性反应客体在某一时刻的状态，然后理解针对这些客体的操作，定义这些客体操作后所得到的状态等。针对需要解决的问题，经常存在着多个客体，它们往往是一系列客体，这样需要了解这些客体的相互驱动、相互作用。

结构化分析方法设计思路不是将客体作为一个整体，而是将这些客体看成功能的组成部分，最后以功能模块将这些客体组成主体。结构化分析方法导致在程序设计过程中，必须将客

体构成的真实世界映射到由功能模块组成的空间中来，这种过程，不仅增加了程序的复杂度，而且与人们对观察问题和解决问题的思路发生了改变，使得人们对问题的理解方面带来了许多困难。在软件系统分析中，作为系统的主体它的功能是不稳定的，而作为客体的对象它是相对稳定的。所以，结构化设计方法是从不稳定的功能操作而切入的，并将描述客体的属性和行为分开，使得应用程序的日后维护和扩展相当困难，甚至很小的变化都使得系统全部变化。随着软件规模、需求、复杂度的不断变化，人们将解决问题的思路放在人类对解决问题的习惯之上，这就是提出面向对象的首要原因。

关于抽象级别方面，良好的抽象策略可以控制问题的复杂程度，增强系统的通用性和可扩展性。抽象包括过程抽象和数据抽象两种方法。结构化设计方法应用的是过程抽象，就是解决问题从功能定义中抽象，并且将每个功能作为一个实体来看待，这样就是从设计的视角将系统按照功能定义了。结构化分析方法就是从过程的视角定义了系统的结构。数据抽象是过程抽象的更高级别，描述的不仅是行为也描述数据，将行为和属性有效的结合起来。将描述的客体的属性和方法绑定在一起，实现了统一的抽象，从而达到对现实世界的真正模拟。

封装是将一些客体的属性与行为绑定在一起，组织在一个逻辑单元中，然后逻辑单元将这些属性隐藏起来，对外提供的只是接口。这样做到了对客体的保护，同时也提高了系统的可维护性。只要接口不变，系统内的其他操作就不会改变。结构化分析方法只是封装了功能模块，而将数据属性和方法分割开来，所以一旦任何一部分发生变化，都会影响系统的变化。

可重用性使面向对象语言的重要特性。结构化分析将对象的功能和属性分割开了，这样导致了软件的可重用性大大下降。而面向对象的分析方法弥补了结构化分析方法中可重用性较差这缺点，增加了代码的可重用性。

3.1.2　面向对象的基本概念

1. 对象

对象是抽象的概念，可以认为万物皆为对象。可以是有形的事物，如书、人、汽车等；也可以是有一组自身特性及属性的抽象事物，如字符串、菜单树等。

面向对象方法是以认识论为基础，用对象来理解和分析问题空间，并设计和开发出由对象构成的软件系统(解空间)的方法。由于问题空间和解空间都是由对象组成的，这样可以消除由于问题空间和解空间结构上的不一致带来的问题。简而言之，面向对象就是面向事情本身，面向对象的分析过程就是认识客观世界的过程。需要指出的是，对象是开发系统中所必需的，是用来实现软件系统的。

只要是系统开发需要，可以将需要开发现实世界中的实体都可以看成是对象。程序设计过程中所面对的问题域都可以使用对象来模拟；对象是在面向对象程序设计中用来描述客观事物的程序单位。问题域中的实体和概念都可以抽象为对象，不同的领域都可以抽象出不同的对象；每个对象都是唯一的。对象的唯一性来自于真实世界中事物的唯一性；每个对象都具有属性和行为；对象具有状态，同一个对象在生命周期内，不同阶段有不同的状态；对象属于某个类，每个对象一定会属于某个类的实例。同一个类具有相同的属性，表明它们的含义相同，但是它们的状态不一定相同。

2. 类

类是具有相同属性和功能的所有对象的集合。对象是类的一个具体实例，类可以看成是

对象的模板。比如，保险公司的人身投保书可能有无数份，无论人身投保书数量有多少，但是，对投保书的操作都包括投保建档、扫描、录入等操作。无论投保书数量有多少，都包含相同的属性。这样在系统分析时，可以不考虑投保书的具体数量，只要考虑其具有相同的属性和方法即可，在系统设计时将这些具有相同属性和方法的对象统称为类，对类进行操作就等于对每个对象进行操作了。

3. 接口

在面向对象的范畴中，接口是一个抽象的概念，是指系统对外提供的所有服务。系统的接口描述系统能够提供哪些服务，但不含服务的实现细节。这里的系统既可以指整个软件系统，也可以指一个子系统，每个对象都是服务提供者，因此每个对象都有接口。

在 UML 关于接口描述时，接口仅仅声明一些空的成员，这些成员必须在类中来实现，接口可继承。既然接口的这些没有实际代码的成员还要在类中实现，那还要接口干什么。某种意义上说接口就是一种规范。比如，如果全世界有 1000 种插头和两种接口。如果现在生产一个插座，你是对这两种“接口”两种插座还是对 1000 种插头做 1000 种插座呢？显然是做两种，那么不实现这个接口的插头(不是 2 线或 3 线的)，就不能插进插座。表面看是一种自己束缚自己，其实是一种解放，正是有了这两种“接口”，才有我们现在很方便地使用插座和插头；否则，那么多厂家，各自按照各自的标准去设计插口，可以想象需要成千上万个插座才能满足要求。

从另外一个角度思考，每个对象，具体地说是类，既然每个类都有服务提供者，如何对外提供服务，接口就是提供了这个机制。

站在使用者的角度，对象中所有向使用者公开的方法的声明构成了对象的接口。使用者调用对象的公开方法来获得服务，使用者在获得服务时，不必关心对象到底是如何实现服务的。接口是提高系统之间松耦合的有利手段。

4. 封装

封装是指将一组数据和与这组数据有关的操作放在一起，形成一个能动的实体——对象。封装是一种通过定义严格的外部接口在单独编写的模块之间减少相互依赖的技术。一个对象要具有封装性，应具有一个清楚的边界；对象内部的属性和实现代码受到封装壳的保护。

面向对象系统的封装性是一种信息隐藏技术，使系统设计员能够清楚地表明他们所提供的服务界面，用户和应用程序员则只能看见对象所提供的服务，而看不到其中的数据和操作代码细节。

对象的封装机制的目的在于将对象的使用者和设计者分开。除了对象的封装以外，类概念本身也具有一种封装意义，它将数据和与这个数据有关的操作集合封装在一起。

封装的最大作用包括，便于使用者正确、方便地理解和使用系统，防止使用者错误修改系统的属性；有助于各个系统之间的松耦合关系，提高系统的独立性；提供软件的可重用性，每个系统都有一个相对独立的整体，可以在多种环境中得到重用；降低了构建大型系统的风险，即使整个系统不成功，个别的独立子系统有可能依然是有价值的。

5. 继承

继承是类与类之间的一种关系，是面向对象程序设计中的一项核心机制。继承的本质就是从已有的类基础上扩充、改造得到新的类，提高了代码的复用性。

继承是一个对象可以获得另一个对象的特性的机制，支持层次分类的概念。

将被继承的对象称为基类或父类，而继承的对象称为派生类或子类。

如果父类中的某些行为不适用于子类，则只需对这些操作的实现部分加以修改和重写，以满足子类的要求。

父类可以继承其他类，子类也可以被别的类所继承，从而形成类的继承层次关系，也称为类层次。

若一个类只有一个父类，则称为单继承，这样建立起来的类类层次形成了一个树。当一个类有多于一个基类时，则称为多重继承。

采用继承性的语言具有如下的优点：提高了软件的重用性；符合逐步求精的软件工程原则；便于实现多态性；便于系统的扩展；类层次反映了现实世界中普遍存在的一般与特殊的关系，也反映了人类认识世界的演绎。

6. 多态

多态性是将多种不同的特殊行为进行抽象的一种能力，通过结合继承性，多态性很好地解决了面向对象遇到的很多麻烦，使得面向对象的编程方式最终得到淋漓尽致的推广。

简单地说，多态是具有表现多种形态能力的特征，在面向对象中是指，语言具有根据对象的类型以不同方式处理，即指同样的消息被不同类型的对象接收时导致完全不同的行为，是对类的特定成员函数的再抽象。多态性在不同的编程语言中拥有不同的解决方案，但多态性的最终目标却始终不变，都是“以不变应万变”。

3.1.3　面向对象的分析

面向对象分析方法(Object-Oriented Analysis，OOA)，是在一个系统的开发过程中进行了系统业务调查以后，按照面向对象的思想来分析问题。OOA 与结构化分析有较大的区别。OOA 所强调的是在系统调查资料的基础上，针对 OO 方法所需要的素材进行的归类分析和整理，而不是对管理业务现状和方法的分析。

面向对象进行建模的系统都用对象或类作为其主要构造块。所以关键是识别问题域内的对象，主要是把现实世界中较为复杂的问题对象化，简单地讲，对象通常是从问题域的词汇中抽取出来的；通过分析对象之间的关系可以让开发人员更清楚地理解整个系统，类就是对具有共同性质的一组对象的描述。

使用 UML 表现面向对象的分析与设计，需要绘制的 UML，其中 UML 的动态模型图主要是交互图和行为图。

交互图又分为时序图、协作图。

行为图又分为状态图、活动图。

静态图包括类图，系统行为图包括用例图。

系统组织图分为包图、组建图、配置图等。

功能模型描述与值的变换有关的系统特征，主要是展现了类中的方法和属性，展示了动态模型中不可分解的动作和活动的定义。

3.2　面向对象设计过程与设计准则

3.2.1　UML 面向对象分析和设计过程

UML 是一种功能强大的，面向对象的可视化系统分析的建模语言，它的各个模型可以帮

助开发人员更好地理解业务流程，减少一些理解不同而带来的不必要错误、保障分析的正确性，建立更可靠、更完善的系统模型，并且还能提高整个开发的效益。软件设计的一般步骤是需求分析、分析设计、编码实现、测试和维护。下面我们重点分析前面四个阶段的一般步骤：

1. 需求分析阶段

需求分析是系统分析与设计的第一步，只有充分理解了用户的需求才能够设计出用户满意的产品，清楚了用户需求后就从业务需求报告中抽取用例，这是整个系统开发的第一步。然后根据用例图写出详细的用例事件流图，事件流图描述的是用例的执行过程，用例事件流的目的就是为了以后更好的分析，能够让开发人员一看就明白，所以把用例的流程写的越详细越好。

2. 分析设计阶段

系统分析设计阶段，确定整个系统中的所有实体，以及实体能做什么以及实体与实体间的关系，从而画出类图，以上所有都是静态的结构，然后根据类图画出时序图，这样就能明白系统间类与类之间是如何交互的，可以帮助程序员进行后面的编码工作，然后可以画出活动图和状态图。

3. 编码实现阶段

该阶段主要是根据前面的结果对整个系统进行编码实现。

4. 测试阶段

测试人员可以对整个系统进行测试，其用例图可以指导测试人员完成系统测试。

3.2.2 面向对象的基本准则

抽象类：类就是模块，把数据结构和操作这些数据的方法紧密地结合在一起构成模块。

抽象：面向对象方法不仅支持对过程进行抽象，而具支持对数据进行抽象，抽象的好坏直接影响整个系统的设计。

信息隐藏：通过对象的封装来实现，对象对外暴露的方法以及接口设计的合理程度对系统设计和扩展有很大影响。

减少通信开销：在设计中应做到在子系统的各个部件之间的通信量达到最小，特别是访问量大的系统，减少系统中的一次通信开支其系统的性能就会提高一个档次。

良好的可扩展性：在面向对象的程序设计中，继承和多态在程序设计中有很好的可扩展性。

高内聚：子系统应该把那些成组的类打包，形成高度内聚。内聚度高的模块很容易理解，很容易被复用，扩展和维护。

低耦合：低耦合是设计的一个重要标准，有助于使系统中某一部分的变化对其他部分的影响降到最低，各个系统模块之间应该相互独立，模块之间应遵循弱耦合。

3.3 UML 介绍

3.3.1 UML 组成

UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言，用模型来描述系统

的静态特征和动态特征。它可以从不同的视角为系统的架构进行建模，形成不同的开发人员一看就能够明白的视图。视图是由一个或多个图组成的对系统的抽象，但它并不是图。视图主要包括用例视图、逻辑视图、实现视图、进程视图和部署视图。

UML 主要由视图、图、模型元素、通用机制组成。视图和图在后面将有详细说明，下面主要介绍模型元素和通用机制。

5 种视图组合构成 UML 完整模型，它们间的关系如图 3-1 所示。

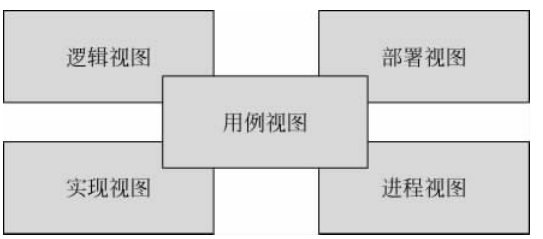

图 3-1　UML 视图关系

UML 中一般提供 9 种基本图，所有的视图都可以由这几种基本图组成。模型元素包括事物和关系。事物分为结构事物，包括类、接口、协作等。行为事物，包括交互图和状态图。分组事物，包括包和注释事物。关系包括关联关系、依赖关系、泛化关系、实现关系和聚合关系。事物是 UML 中重要的组成部分，它代表任何可以定义的东西。在 UML 中事物之间的关系能够把事物联系在一起，组成有意义的结构模型。每一个模型元素都有一个与之相对应的图形元素。

通用机制包括修饰、注释、规格说明、通用划分和扩展机制。扩展机制允许用户对 UML 进行扩展来适应特定的组织或用户，主要包括构造型、约束和标记值。

3.3.2　用例视图

描述可被最终用户、设计人员、分析人员和测试人员看到的系统行为的用例组成，它实际上是从用户角度来描述系统应具有的功能，是对系统的抽象表示，由用例图组成。

3.3.3　逻辑视图

展现系统的静态或结构组件及特征，主要是设计类、包及与系统的各种组织形式，显示的是系统内部的功能是怎样设计的，利用系统的静态结构和动态行为来刻画系统的功能。逻辑视图是从系统的静态结构和动态行为角度分析如何实现系统的功能。

静态结构是指组成系统的类、对象以及它们之间的关系等，由类图和对象图组成。

动态行为主要由时序图、协作图、状态图和活动图组成，逻辑视图的实现者主要是设计人员和开发人员。

3.3.4　组件视图

组件视图用来显示代码组件的组织方式。它描述了模块和它们之间的依赖关系，体现了系统实现的结构和行为特征。组件视图由组件图构成，组件是代码模块，不同类型的代码模块形成不同的组件，组件按照一定的结构和依赖关系呈现。组件视图主要供开发者使用，主要用类图、包图、对象图、顺序图、状态图和活动图来描述。

3.3.5　进程视图

进程视图与系统处理性能相关的元素包括可伸缩性、吞吐量、基本时间性能。包含了系统并发和同步机制的线程和进程，通过分析和设计系统如何有效利用资源、并行执行、处理来自

外界的异步事件。进程视图包括动态图(状态机、交互图、活动图)和实现图(交互图和部署图),它的使用者是系统集成人员和开发人员。

3.3.6 部署视图

描述物理系统的硬件拓扑结构及各种软件模块和构件结构。部署视图由部署图来表示,主要的使用者是开发人员、系统集成人员和测试人员,主要描述系统的部署方式。

3.4 UML 图

图是 UML 的重要组成部分,UML 图分为静态图和动态图,静态图主要包括用例图、类图、对象图、组件图和部署图,动态图主要包括协作图、状态图、活动图和顺序图,下面是各图的详细信息。

3.4.1 静态图

1. 用例图

用例图(Use Case Diagram)是从需求分析报告到软件系统设计的第一步,也是该系统整个分析过程中最重要的图,它的改变将影响到其他图。用例图模型如 3-2 所示。

用例图用于显示若干参与者与系统提供用例之间的连接关系,主要描述系统的功能,可描述整个系统的功能需求。用例图主要包括参与者、用例、关联关系、包含关系、扩展关系和泛化关系。下面主要介绍用例和参与者。

用例:用例可以理解为是系统的功能单元。是在需求分析报告中提取相关的动词或动名词组成。在 UML 中用例用一个椭圆来表示,椭圆的下方一般写的是用例的名字,如图 3-3 所示。

参与者:参与者是直接与系统交互的实体,可以是该系统的用户,可以是其他系统,也可以是一些可以运行的进程。通常用人形图标表示,名字也一般是写在下面,如图 3-4 所示。

图 3-2 用例图一般模型

图 3-3 用例

图 3-4 参与者

2. 类图

前面说过,类是对象的模板,对象是类的实例,所以类是面向对象设计的核心所在。类图(Class Diagram)在软件具体设计中占有很重要的位置,而类图是描述系统中类与类之间关系的静态视图。

类图是面向对象设计中最常见的一个图,描述的是类、接口、协作及它们之间相互关系的图。类图属于静态模型类型。

类图包含类、接口、协作、依赖关系、泛化关系、关联关系和实现关系。

类:在 UML 中用矩形表示,如图 3-5 所示,共由三部分组成,上面的是该类的名字,中间的是类的属性,下面的是该类的方法。并且类的属性和方法可以隐藏。

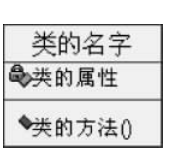

图 3-5 类

接口：在 UML 中接口用一个带有名字的小圆圈表示，如图 3-6 所示。一个类可以实现一个或多个接口，且必须实现所有接口中的方法。在实现的类中，有些方法可以是空方法。接口也可以显示成一个类的符号，特殊的地方就是会带有<< interface >>标识。

3. 对象图

对象图(Object Diagram)是类图的实例，与类图极为相似，对象图显示类的多个对象实例，而不是实际的类。如图 3-7 所示，它描述的不是类之间的关系，而是对象之间的关系。对象图与类图使用相同的符号，因为对象就是类的实例，只不过对象图只有属性和名称。并且对象图的名称是“对象名:类名”或“:类名”。

4. 组件图

组件图(Component Diagram)和部署图是面向对象设计系统，建模必须要用到的图，如图 3-8 所示。

图 3-6　接口

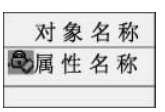

图 3-7　对象图

图 3-8　组件

组件图是用来反映代码的物理结构。从组件图中，可以了解各软件组件之间的编译器和运行时依赖关系。使用组件图可以将系统划分为内聚组件并显示代码自身的结构。

组件图的主要目的是显示系统组件间的结构关系。

在 UML 中用一个左侧带有二个突出小矩形的矩形来表示。

5. 配置图

配置图描述的是组件的配置及系统如何在硬件上部署，描述系统中硬件和软件的物理配置情况和系统体系结构。它的用途是显示该系统组件运行的物理地址和它们的通信方式，部署图是对物理运行情况进行的建模，使用者主要是开发人员、系统集成人员和测试人员。

3.4.2　动态图

1. 状态图

描述状态之间转换的图称为状态图(State Diagram)。状态图是对系统的动态建模，是对类描述的补充，由状态、转换、事件、活动和动作组成。

状态图是由表示状态的节点和状态之间转换的带箭头的直线组成，重点描述对象的状态及其状态之间的转移。通过状态图可以了解一个对象所能达到的所有状态，以及对象收到的事件对对象状态的影响。状态中的事件分为调用事件、变化事件、时间事件和信号事件，状态图主要由状态、初始状态、终结状态、转换、判断等组成。

2. 活动图

活动图(Activity Diagram)描述了需要做的活动以及执行这些活动的顺序，着重描述操作(方法)实现中所完成的工作以及用例实例或对象中的活动，是状态图的一个变种。活动图着重表现从一个活动到另一个活动的控制流，是内部处理驱动的流程。

状态图主要包括动作状态、活动状态、动作流、分支与合并、分叉与汇合、泳道和对象流。

状态图和活动图都是以描述系统状态转移为主，只不过活动图主要描述动作及对象状态改变的结果。状态图主要描述的是事件对对象状态的影响。

3. 时序图

时序图(Sequence Diagram)描述对象随时间的推移消息是如何交互的。其重点放在消息序列上，强调消息是如何在对象之间被发送和接收的。对象、生命线、消息和激活组成一个完整的时序图。

4. 协作图

协作图(Collaboration Diagram)是基于结构来描述对象间的交互关系，用于描述组件及其交互关系的空间组织结构，并不侧重于交互的顺序。和时序图相比，只不过是各自的出发点不同而已，时序图是以交互时的时间为顺序为出发点，而协作图是以对象间的关系为出发点，两者是可以相互转换的。时序图主要包含对象、链和消息。

协作图和时序图以描述系统系统对象通信和交互为主的动态图，虽然都是描述对象交互的，但是时序图强调的是时间，协作图强调的空间。

3.5 UML 关系

前面所讲的都是一些基本的知识，但不管学什么，只知道一些基本的知识是不够的，必须将学习到得知是与实际应用结合起来。下面就来讲 UML 中的各种关系，UML 之所以强大就是利用这些关系能够满足描述清楚用户的需求。本节主要将讲用例关系、类关系、包关系和组件关系。

3.5.1 用例关系

在一些业务比较简单的系统中，直接考虑参与者与用例之间的关系就可以了，这在 UML 中称为关联关系。但是当遇到一些复杂的业务时，用例与用例之间也会有关系。用例与用例的关系有包含关系、扩展关系和泛化关系。

1. 关联关系

关联关系是用例图中非常普遍的一种关系，也是最直观和简单的一种关系。在 UML 中用实线加箭头表示。

2. 包含关系

包含关系是用例与用例之间的关系，当用到某一个用例时必须用到另外一个用例，这就是包含关系。在 UML 中包含关系用虚箭头加<< include >>表示，箭头指向被包含的用例。

3. 扩展关系

简单地说，扩展关系就是在执行某一个用例时可能会去执行其他用例，可能会执行的用例叫做扩展用例，在 UML 中，扩展关系用虚线箭头加<< extend >>表示，并且箭头由扩展用例发出。

包含关系和扩展关系的区别，包含关系是一定要用包含用例，而扩展关系中的可能执行也可能不执行的。

4. 泛化关系

泛化关系与类之间的泛化关系关系相同，子用例也继承父用例所有的属性和方法。在

UML 中用实线加一个三角箭头表示，并且由子用例指向父用例。

3.5.2　类关系

1. 依赖关系

依赖关系是比较好理解的一种关系，就是指两个相互独立的对象，一个对象的存在需要依赖另一个对象时，这种关系就叫做依赖关系，关联关系、实现关系和泛化关系都是语义更强的依赖关系，在 UML 中用虚线加箭头表示，箭头指向依赖的对象。

2. 关联关系

关联关系是两个类、或类与接口之间语义级别的一种强依赖关系，是一种结构化的关系。关联关系可分为一对一、一对多和多对多，在 UML 中用实线来表示。

3. 聚合

聚合是关联关系的一种特例，表示部分和整体的关系，UML 中用带空心菱形头的实线表示聚合关系，菱形头指向整体。聚合指整体在概念上处于比局部更高的一个级别，而关联暗示两个类在概念上位于相同的级别。聚合只能是单向关系。

4. 组合

组合是聚合关系的变种，表示元素间更强的组合关系。如果是组合关系，如果整体被破坏则个体一定会被破坏，而聚合的个体则可能是被多个整体所共享的，不一定会随着某个整体的破坏而被破坏。UML 中用带实心菱形头的实线表示组合关系，菱形头指向整体。

5. 泛化关系

泛化关系描述的是类与类之间的继承关系，接口与接口之间的继承关系。主要体现的是继承的思想。UML 中用带空心箭头的实线表示泛化关系，箭头指向一般个体。

6. 实现关系

实现是类与接口之间最常见的关系，指的是一个类实现接口或多个接口的功能。

其中，依赖的关系最弱；聚合，组合都是依赖关系的一种；而关联、聚合、组合表示的关系依次增强。聚合是表明对象之间的整体与部分关系的关联，而组合是表明整体与部分之间有相同生命周期关系的聚合。

3.5.3　包关系

在建模过程中，可能有很多元素，项目大的话看起来会比较杂乱，包就是将元素根据语义进行分组。包可以理解为文件夹，是用来组织图形的封装，包图可以用来表述功能组命名空间的组织层次。在 UML 中包用两个矩形来表示，如图 3-9 所示。

图 3-9　包

包中可以包含类、接口、组件、节点、协作、用例、图以及其他包。

包和包的关系有引入和访问依赖、泛化二种。

引入和访问依赖是指包可以访问其他的包中的元素，前提是得引入那包的元素。引入和访问依赖是支持传递性。

包引入这一关系是为了让一个命名空间能更方便的引用另一个包中的元素，其方便性在于对于引入的包中的元素可以采用非限定姓名的访问方式进行访问。包访问关系概念的引入是为了建模包的增量式扩展，即我们可以在已有的一个包上通过包合并从而

得到一个功能更加完整或是强大的新包。

泛化：包的泛化和类与类之间的泛化关系差不多。也一样的会继承父包中的所有，也可以定义属于自己的。

3.5.4 组件关系

组件也叫做构件，可以代表从类到应用、系统中的任何事物，组件可以是一组接口、可以有实例。组件给外界提供的并不是包含类的所有接口，这样能够起到安全的作用，组件有配置组件、执行组件和工作产品组件三大类，在 UML 中使用在左侧带有两个小矩形的大矩形表示。

组件关系分为组件与接口的关系和组件与组件的关系。

组件与接口的关系有实现关系和依赖关系。

实现关系：接口和组件之间用实线连接。

依赖关系：接口和组件之间用虚线箭头连接。

组件与组件的关系用依赖关系表示。它与类之间的依赖关系相同。

3.6 UML 机制

UML 提供了几种扩展机制，允许建模者在不用改变基本建模语言的情况下做一些通用的扩展。这些扩展机制已经被设计好，以便于在不需理解全部语义的情况下就可以存储和使用。使 UML 语言变得简单和更易于使用，你可以在 UML 语言的任何时候用同样的方法来使用。UML 机制包括修饰、注释、规格说明、通用划分和扩展机制。

3.6.1 通用机制

规格说明：UML 的每一个图形背后都有一个规格说明，规格说明描述系统的细节。UML 中预定义的特性有文档、职责、永久性和并发性。

修饰：在 UML 建模过程中，可以对模型元素进行修饰，修饰增加了模型元素的语义。类用粗体字显示名字，对象用名字带下划线来表示。

注释：为了让开发人员更能看明白系统中各个模型元素的作用和意义，注释是一种图形符号用来限制或给一个元素或者组元素加上注解。注释是最重要的一种修饰，注释画成一个带折角的矩形，在矩形中加上文字或图形的注解，一般用虚线将注释与被注释模型元素连接起来。

通用划分：UML 中通用划分分为类/对象二分法和接口/实现二分法。

3.6.2 扩展机制

扩展机制：UML 扩展机制允许使用人员自定义一些自己的构造型语言成分。包括构造型、约束和标记值。

构造型：它不仅允许用户对模型元素进行必要的扩展和调整，还能够有效地防止 UML 变得过于复杂，是对 UML 词汇的扩展，用它可以自己创建新种类的模型元素。通过向新的模型元素中添加属性，新的模型元素可以扩展原模型元素，构造型可以被看成特殊的类，用双尖括号内的文字字符串表示。

约束：可以用这对 UML 元素语义的扩展，可以增加自己各种规则。用文字表达式表示

的施加在某个模型元素上的语义限制，约束可以附加在表元素、依赖关系或注释上。用于加入新的规则或修改已经存在的规则，即利用一个表达式把约束信息应用于元素上。约束用大括号内的字符串表达式表示。

标记值：是对 UML 元素特性的扩展，可以在模型元素中创建新的信息。可以对某种属性“键-值”对的明确定义，标记值扩展 UML 构造块的特性或标记其他模型元素，为 UML 事物增加新特性。使用标记值的目的是赋予某个模型元素新的特性。标记值可以用来存储元素的任意信息，也可以用来存储有关构造型模型元素的信息。标记值用字符串表示，字符串有标记名、等号和值。它们被规则地放置在大括号内。