

## 考点分析



### 第9章 面向对象方法

面向对象方法是软件设计师级别考试的一个考试重点，上午考试的分数中它大约占12分，下午考试通常有一道面向对象设计的试题，占15分。

#### 9.1 考点分析

本节把历次考试中面向对象方法的试题进行汇总，得出本章的考点，如表9-1所示。

表9-1 面向对象方法试题知识点分布

考试时间	分数	考查知识点
10.11	8	类的实例化 (1)、重置 (1)、OMT 模型 (3)、UML (3)
11.05	7	类的定义 (2)、类的实例化 (1)、类之间的关系 (2)、UML (2)
11.11	8	重载 (1)、对象之间的关系 (1)、类 (2)、对象消息 (1)、类之间的关系 (2)、面向对象方法 (1)
12.05	11	类的定义 (5)、设计模式概念 (1)、单身模式 (1)、MVC 模式 (1)、类图 (2)、关联的多重度 (1)
12.11	12	面向对象分析 (1)、面向对象语言 (1)、面向对象接口 (2)、UML (5)、设计模式 (3)
13.05	13	类的实例化 (1)、UML (1)、面向对象分析 (2)、MVC 模式 (2)、设计模式的作用 (1)、设计模式的分类 (3)、关联关系 (1)、类之间的关系 (2)
13.11	12	类之间的关系 (1)、用例 (1)、UML 图形 (9)、设计模式分类 (1)
14.05	12	泛化 (1)、封装与继承 (2)、类库与框架 (2)、设计模式 (4)、类之间的关系 (3)

根据表9-1,我们可以得出面向对象方法的考点主要有：

- (1) 面向对象分析：包括面向对象方法、面向对象分析、OMT模型。
- (2) 面向对象语言：包括类的定义、类的实例化、消息、重载、接口。
- (3) 统一建模语言 (UML)：包括图形、类之间的关系、多重度、用例。
- (4) 设计模式：包括设计模式的分类、设计模式的作用、MVC、单身模式等。

对这些知识点进行归类，然后按照重要程度进行分类，如表9-2所示。其中的五角星号 (\*) 代表知识点的重要程度，星号越多，表示越重要。

在本章的后续内容中，我们将对这些知识点进行逐个讲解。

表9-2 面向对象方法各知识点重要程度

知识点	10.11	11.05	11.11	12.05	12.11	13.05	13.11	14.05	合计	比例	重要程度
面向对象分析	3		1		1	3			8	9.52%	★
面向对象语言	2	3	4	5	3	1		4	22	26.19%	★★★
UML	3	4	3	3	5	4	11	4	37	44.05%	★★★★★
设计模式				3	3	6	1	4	17	20.24%	★★★

版权方授权希赛网发布，侵权必究

上一节      本书简介      下一节

---

## 9.2 面向对象分析

面向对象方法包括面向对象的分析、面向对象的设计和面向对象的程序设计。面向对象分析是面向对象方法的核心之一，而且拥有大量不同的方法，主要包括OMT、Coad/Yourdon方法、OOSE、Booch方法等，而OMT、OOSE、Booch最后统一成为UML ( United Model Language, 统一建模语言 )。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

第 9 章：面向对象方法

作者：希赛教育软考学院    来源：希赛网    2014年05月21日

## Coad/Yourdon方法

---

### 9.2.1 Coad/Yourdon方法

Coad/Yourdon方法由P. Coad和E. Yourdon于1990年推出，该方法主要由面向对象的分析 ( Object-Oriented Analysis, OOA ) 和面向对象的设计 ( Object-Oriented Design, OOD ) 构成，特别强调OOA和OOD采用完全一致的概念和表示法，使分析和设计之间不需要表示法的转换。该方法的特点是表示简练、易学，对于对象、结构、服务的认定较系统、完整，可操作性强。

在Coda/Yourdon方法中，OOA的任务是主要建立问题域的分析模型。分析过程和构造OOA概念模型的顺序由5个层次组成，这5个层次是类与对象层、属性层、服务层、结构层和主题层，它们表示分析的不同侧面。

OOA首先要确定问题域，然后需要经过5个步骤来完成整个分析工作，即确定对象类、确定结构与关联、划分主题、定义属性、定义服务。但这5项活动完全没必要按顺序依次完成，也无须彻底完成一项活动之后再开始另外一项活动。

(1) 确定类与对象：类与对象是在问题域中客观存在的，系统分析的重要任务之一就是找出这些类与对象。首先找出所有候选的类与对象，然后进行反复筛选，删除不正确或不必要的类与对象。

(2) 确定结构与关联：结构与关联反映了对象 ( 或类 ) 之间的关系，主要有以下几种：  
一般-特殊结构，又称分类结构，是由一组具有一般-特殊关系 ( 继承关系 ) 的类所组成的结构。一般-特殊关系的表达式为：is a kind of.

整体-部分结构，又称组装结构，是由一组具有整体-部分关系 ( 组成关系 ) 的类所组成的结构。整体-部分关系的表达式为：has a.

实例关联，即一个类的属性中含有另一个类的实例 ( 对象 ) ，它反映了对象之间的静态联系。

消息关联，即一个对象在执行自己的服务时需要通过消息请求另一个对象为它完成某个服务，它反映了对象之间的动态联系。

(3) 划分主题：在开发大型、复杂软件系统的过程中，为了降低复杂程度，需要把系统划分成几个不同的主题。注意应该按问题域而不是用功能分解方法来确定主题，应该按照使不同主题内的对象相互间依赖和交互最少的原则来确定主题。

(4) 定义属性：为了发现对象的属性，首先考虑借鉴以往的OOA结果，看看相同或相似的问题

域是否有已开发的OOA模型，尽可能复用其中同类对象的属性定义。然后，按照问题域的实际情况，以系统责任为目标进行正确的抽象，从而找出每一对象应有的属性。

(5) 定义服务：发现和定义对象的服务，也应借鉴以往同类系统的OOA结果并尽可能加以复用。然后，研究问题域和系统责任以明确各个对象应该设立哪些服务，以及如何定义这些服务。

OOD中将贯穿OOA中的5个层次和5个活动，它由4个部件组成，分别是人机交互部件、问题域部件、任务管理部件、数据管理部件，其主要的活动就是这4个部件的设计工作。

(1) 设计问题域部件。通过OOA所得出的问题域精确模型，为设计问题域部件奠定了良好的基础。通常，OOD仅需从实现角度对问题域模型做一些补充和修改，主要是增添、合并或分解类与对象、属性及服务、调整继承关系等。

(2) 设计人机交互部件。在OOA过程中，已经对用户界面需求做了初步分析。在OOD过程中，则应该对系统的人机交互部件进行详细设计，以确定人机交互的细节，其中包括指定窗口和报表的形式、设计命令层次等内容。

(3) 设计任务管理部件。主要是识别事件驱动任务，识别时钟驱动任务，识别优先任务，识别关键任务，识别协调任务，审查每个任务并定义每个任务。

(4) 设计数据管理部件。提供数据管理系统中存储和检索对象的基本结构，隔离具体的数据管理方案（如普通文件、关系数据库、面向对象数据库等）对其他部分的影响。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

## Booch方法

### 9.2.2 Booch方法

Booch认为软件开发是一个螺旋上升的过程，每个周期包括4个步骤：

(1) 标识类和对象：在给定的抽象层次上识别类和对象，包括找出问题空间中关键的抽象和产生态行为的重要机制。开发人员可以通过研究问题域的术语发现关键的抽象。

(2) 确定类和对象的含义：建立前一阶段识别出的类和对象的语义。

(3) 标识关系：识别这些类和对象之间的关系。开发人员确定类的行为（即方法）和类及对象之间的互相作用（即行为的规范描述）。该阶段利用状态转移图描述对象的状态模型，利用时态图（系统中的时态约束）和对象图（对象之间的互相作用）描述行为模型。

(4) 说明每个类的接口和实现：实现类和对象。

这4种活动不仅仅是一个简单的步骤序列，而是对系统的逻辑和物理视图不断细化的迭代和渐增的开发过程。

Booch强调基于类和对象的系统逻辑视图与基于模块和进程的系统物理视图之间的区别。

Booch方法的开发模型包括静态模型和动态模型，静态模型分为逻辑模型（类图、对象图）和物理模型（模块图、进程图），描述了系统的构成和结构。动态模型包括状态图和时序图。该方法对每一步都作了详细的描述，描述手段丰富，灵活。不仅建立了开发方法，还提出了设计人员的技术要求，不同开发阶段的人力资源配置。Booch方法的基本模型包括类图与对象图，主张在分析和设计

中既使用类图，也使用对象图。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

第 9 章：面向对象方法

作者：希赛教育软考学院    来源：希赛网    2014年05月21日

## OMT方法

---

### 9.2.3 OMT方法

OMT ( Object Model Technology,对象建模技术 ) 作为一种软件工程方法学，它支持整个软件生存周期，覆盖了问题构成分析、设计和实现等阶段。OMT方法使用了建模的思想，讨论如何建立一个实际的应用模型。从3个不同而又相关的角度建立了3类模型，分别是对象模型、动态模型和函数模型，OMT为每一个模型提供了图形表示。

(1) 对象模型。描述系统中对象的静态结构、对象之间的关系、属性、操作。它表示静态的、结构上的、系统的"数据"特征。主要用对象图来实现对象模型。

(2) 动态模型。描述与时间和操作顺序有关的系统特征，如激发事件、事件序列、确定事件先后关系的状态。它表示瞬时、行为上的、系统的"控制"特征。主要用状态图来实现动态模型。

(3) 功能模型。描述与值的变换有关的系统特征：功能、映射、约束和函数依赖。主要用数据流图来实现功能模型。

在进行OMT建模时，通常包括4个活动，分别是分析、系统设计、对象设计和实现。

(1) 分析：建立可理解的现实世界模型。通常从问题陈述入手，通过与客户的不断交互以及对现实世界背景知识的了解，对能够反映系统的三个本质特征（对象类及它们之间的关系，动态的控制流，受约束的数据的函数变换）进行分析，构造出现实世界的模型。

(2) 系统设计：确定整个系统的体系结构，形成求解问题和建立解答的高层策略。

(3) 对象设计：在分析的基础上，建立基于分析模型的设计模型，并考虑实现细节。其焦点是实现每个类的数据结构及所需的算法。

(4) 实现：将对象设计阶段开发的对象类及其关系转换为程序设计语言、数据库或硬件的实现。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

第 9 章：面向对象方法

作者：希赛教育软考学院    来源：希赛网    2014年05月21日

## OOSE

---

### 9.2.4 OOSE

OOSE ( Object-Oriented Software Engineering,面向对象的软件工程 ) 在OMT的基础上，对功能模型进行了补充，提出了用例 ( use case ) 的概念，最终取代了数据流图进行需求分析和建立

功能模型。

OOSE方法采用5类模型来建立目标系统，这5个模型是：

(1) 需求模型：获取用户的需求，识别对象，主要的描述手段有用例图、问题域对象模型及用户界面。

(2) 分析模型：定义系统的基本结构。通过将分析模型中的对象分别识别到分析模型中的实体对象、界面对象和控制对象三类对象中。每类对象都有自己的任务、目标并模拟系统的某个方面。实体对象模拟那些在系统中需要长期保存并加以处理的信息，实体对象由使用事件确定，通常与现实生活中的一些概念符合。界面对象的任务是提供用户与系统之间的双向通信，在使用事件中所指定的所有功能都直接依赖于系统环境，它们都放在界面对象中。控制对象的典型作用是将另外一些对象组合形成一个事件。

(3) 设计模型：分析模型只注重系统的逻辑构造，而设计模型需要考虑具体的运行环境，将在分析模型中的对象定义为模块。

(4) 实现模型：用面向对象的语言来实现。

(5) 测试模型：测试的重要依据是需求模型和分析模型，测试的方法与技术4.5节所介绍的类似，而底层是对类（对象）的测试。测试模型实际上是一个测试报告。

OOSE的开发活动主要分为3类，分别是分析、构造和测试。其中分析过程分为需求分析和健壮分析两个子过程，分析活动分别产生需求模型和分析模型。构造活动包括设计和实现两个子过程，分别产生设计模型和实现模型。测试过程包括单元测试、集成测试和系统测试3个过程，共同产生测试模型。

用例是OOSE中的重要概念，在开发各种模型时，用例是贯穿OOSE活动的核心，描述了系统的需求及功能。用例实际上是描述系统用户（使用者、执行者）对于系统的使用情况的，是从使用者的角度来确定系统的功能的。因此，首先必须分析确定系统的使用者，然后进一步考虑使用者的主要任务、使用的方式，识别所使用的事件，即用例。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

## 面向对象语言

### 9.3 面向对象语言

本节主要介绍面向对象语言的一些基本概念。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

## 类和对象

### 9.3.1 类和对象

本节主要考查类和对象的基本概念，以及类的定义和实例化。

#### 1.对象的概念

在计算机系统中，对象是指一组属性以及这组属性上的专用操作的封装体。属性可以是一些数据，也可以是另一个对象。每个对象都有它自己的属性值，表示该对象的状态，用户只能看见对象封装界面上的信息，对象的内部实现对用户是隐蔽的。封装目的是使对象的使用者和生产者分离，使对象的定义和实现分开。一个对象通常可由3部分组成，分别是对象名、属性和操作（方法）。

#### 2.类的概念

类是一组具有相同属性和相同操作的对象的集合。一个类中的每个对象都是这个类的一个实例（instance）。在分析和设计时，我们通常把注意力集中在类上，而不是具体的对象上。通常把一个类和这个类的所有对象称为类及对象或对象类。

一个类通常可由3部分组成，分别是类名、属性和操作（方法）。每个类一般都有实例，没有实例的类是抽象类。抽象类不能被实例化，也就是不能用new关键字去产生对象，抽象方法只需声明，而不需实现。抽象类的子类必须覆盖所有的抽象方法后才能被实例化，否则这个子类还是个抽象类。

#### 3.继承

继承关系表示了对象间"is-a"的关系，即子类（派生类）是父类（超类、基类）的一种。继承是在某个类的层次关联中不同的类共享属性和操作的一种机制。一个父类可以有多个子类，这些子类都是父类的特例。父类描述了这些子类的公共属性和操作，子类还可以定义它自己的属性和操作。一个子类只有唯一的父类，这种继承称为单一继承。一个子类有多个父类，可以从多个父类中继承特性，这种继承称为多重继承。对于两个类A和B,如果A类是B类的子类，则说B类是A类的泛化。

继承是面向对象方法区别于其他方法的一个核心思想。继承机制实现的是父类和子类之间的关系，子类又能够作为其他类的父类，因此组织而成一个层次结构。在程序中，凡是引用父类对象的地方都可以使用子类对象来代替。

#### 4.类库

类库是实现各种功能的类的集合，面向对象的语言中都提供了一些已知功能的类，这些功能类采用继承等方式进行组装，就可以实现常见的一些功能。例如：界面处理、网络连接、数据库处理等。是否建立了丰富的类库是衡量一个面向对象程序设计语言成熟与否的重要标志之一。

#### 5.框架

框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法；另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面，而后者是从目的方面给出的定义。

可以说，框架是一个可复用的设计构件，它规定了应用的体系结构，阐明了整个设计、协作构件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间协作的方法，它为构件复用提供了上下文（Context）关系。因此构件库的大规模重用也需要框架。

构件领域框架方法在很大程度上借鉴了硬件技术发展的成就，它是构件技术、软件体系结构研究和应用软件开发三者发展结合的产物。在很多情况下，框架通常以构件库的形式出现，但构件库只是框架的一个重要部分。框架的关键还在于框架内对象间的交互模式和控制流模式。

框架比构件可定制性强。在某种程度上，将构件和框架看成两个不同但彼此协作的技术或许更

好。框架为构件提供重用的环境，为构件处理错误、交换数据及激活操作提供了标准的方法。

应用框架的概念也很简单。它并不是包含构件应用程序的小片程序，而是实现了某应用领域通用完备功能（除去特殊应用的部分）的底层服务。使用这种框架的编程人员可以在一个通用功能已经实现的基础上开始具体的系统开发。框架提供了所有应用期望的默认行为的类集合。具体的应用通过重写子类（该子类属于框架的默认行为）或组装对象来支持应用专用的行为。

应用框架强调的是软件的设计重用性和系统的可扩充性，以缩短大型应用软件系统的开发周期，提高开发质量。与传统的基于类库的面向对象重用技术比较，应用框架更侧重于面向专业领域的软件重用。应用框架具有领域相关性，构件根据框架进行复合而生成可运行的系统。框架的粒度越大，其中包含的领域知识就更加完整。

## 6.类的定义

类实际上就是由一组描述对象属性或状态的数据项和作用在这些数据项上的操作（或称为方法、成员函数等）构成的封装体。类的定义由关键字class打头，后跟类名，类名之后的括号内是类体，最后以分号";"结束。

类与C语言中的结构体大致相似，其不同之处在于类中规定了哪些成员可以访问，哪些成员不可以访问。这些都通过访问指明符予以说明。访问指明符有三种，分别是private、protected和public。

（1）private 成员私有化，除了该类的成员函数以外，谁也不能访问它们。

（2）public 成员公有化，程序中的所有函数（不管是类内定义的还是类外定义的），都可以访问这些成员。

（3）protected 成员受限保护，只有该类及该类的子类的成员函数才能够访问。在类的成员定义中，如果没有指明符，则系统默认为private。

要注意的是在C++语言中，一个类的成员是可以访问该类的所有成员的。

继承的限定也有三种，分别是private（私有继承）、protected（保护继承）和public（公有继承）。

在public继承时，派生类（子类）的public、private、protected型的成员函数可以访问基类中的public成员和protected成员，派生类的对象仅可访问基类中的public成员。

在private继承时，派生类的public、private、protected型的成员函数可以访问基类中的public成员和protected成员，但派生类的对象不可访问基类中的任何成员。

在protected继承时，派生类的public、private、protected型的成员函数可以访问基类中的public成员和protected成员，但派生类的对象不可访问基类中的任何成员。

使用class关键字定义类时，默认的继承方式是private,也就是说，当继承方式为private继承时，可以省略private。

另外，类的成员有动态和静态之分，默认情况下，为动态成员。如果在成员说明前加上static,则说明是静态成员。静态成员与对象的实例无关，只与类本身有关。它们用来实现类要封装的功能和数据，但不包括特定对象的功能和数据。

静态成员包括静态方法和静态属性。静态属性包含在类中要封装的数据，可以由所有类的实例共享。实际上，除了属于一个固定的类并限制访问方式外，类的静态属性非常类似于函数的全局变量。

## 封装和消息

---

### 9.3.2 封装和消息

封装就是将客户端不应看到的信息包裹起来，使内部执行对外部来看是一个黑箱，客户端不需要内部资源就能达到他的目的。这样，事物的内部实现细节隐藏起来，对外提供一致的公共的接口，间接访问隐藏数据，提高可维护性。

#### 1.封装的优点

面向对象系统中的封装单位是对象，对象之间只能通过接口进行信息交流，外部不能对对象中的数据随意地进行访问，这就造成了对象内部数据结构的不可访问性，也使得数据被隐藏在对象中。封装的优点体现在以下3个方面：

- (1) 好的封装能减少耦合。
- (2) 类的内部的实现可以自由改变。
- (3) 一个类有更清楚的接口。

#### 2.接口

一个对象通过封装以后，提供给其他对象的可见部分就是接口。在面向对象的程序设计过程中，要针对接口编程，而不是实现编程。

#### 3.消息

消息是对象间通信的手段。一个对象通过向另一对象发送消息来请求其服务。一个消息通常包括接收对象名、调用的操作名和适当的参数（如有必要）。消息只告诉接收对象需要完成什么操作，但不能指示接收者怎样完成操作。消息完全由接收者解释，接收者独立决定采用什么方法来完成所需的操作。

应用程序之间可以相互发送消息，应用程序还可以向操作系统发送消息，所有I/O设备输入/输出时也会产生消息。消息通信机制与传统的子程序调用机制不同，子程序被调用是完全被动的，而消息的接收方是处理的主体。

## 多态性

---

### 9.3.3 多态性

多态性是指同一个操作作用于不同的对象可以有不同的解释，产生不同的执行结果。与多态性密切相关的一个概念就是动态绑定。传统的程序设计语言把过程调用与目标代码的连接放在程序运



行前进行，称为静态绑定。而动态绑定则把这种连接推迟到运行时才进行。在运行过程中，当一个对象发送消息请求服务时，要根据接收对象的具体情况将请求的操作与实现的方法连接，即动态绑定。

### 1.多态的分类

在使用多态技术时，用户可以发送一个通用的消息，而实现的细节则由接收对象自行决定，这样同一消息就可以调用不同的方法。

多态有多种不同的形式，其中参数多态和包含多态称为通用多态，过载多态和强制多态成为特定多态。

(1) 参数多态应用比较广泛，被称为最纯的多态。这是因为同一对象、函数或过程能以一致的形式用于不同的类型。

(2) 包含多态最常见的例子就是子类型化，即一个类型是另一类型的子类型。

(3) 过载多态是同一变量被用来表示不同的功能，通过上下文以决定一个类所代表的功能。即通过语法对不同语义的对象使用相同的名字，编译能够消除这一模糊。

(4) 强制多态是通过语义操作把一个变元的类型加以变换，以符合一个函数的要求，如果不做这一强制性变换将出现类型错误。类型的变换可在编译时完成，通常是隐式地进行，当然也可以在动态运行时来做。

### 2.类属类

类属类仅描述了适用于一组类型的通用样板，由于其中所处理对象的数据类型尚未确定，因而程序员不可用类属类直接创建对象实例，即一个类属类并不是一种真正的类类型。

类属类必须经过实例化后才能成为可创建对象实例的类类型。类属类的实例化是指用某一数据类型替代类属类的类型参数。类属类定义中给出的类型参数称为形式类属参数，类属类实例化时给出的类型参数称为实际类属参数。如果类属类实例化的实际类属参数可以是任何类型，那么这种类属类称为无约束类属类。然而在某些情况下，类属类可能要求实际类属参数必须具有某些特殊的性质，以使得在类属类中可应用某些特殊操作，这种类属类称为受约束类属类。

### 3.重载

在同一可访问区内被声名的几个具有不同参数列的（参数的类型、个数、顺序不同）同名函数，程序会根据不同的参数列来确定具体调用哪个函数，这种机制叫重载。重载也称为重置，重载不关心函数的返回值类型。例如，在同一可访问区内有：

(1) double calculate ( double )。

(2) double calculate ( double,double )。

(3) double calculate ( double,int )。

(4) double calculate ( int,double )。

(5) double calculate ( int )。

(6) float calculate ( float )。

(7) float calculate ( double )。

7个同名方法calculate,前6个中任意两个均构成重载，第6个和第7个也能构成重载。而第1个和第7个却不能构成重载，因为它们的参数相同。

## 统一建模语言

---

### 9.4 统一建模语言

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。它融入了软件工程领域的新思想、新方法和新技术。它的作用域不限于支持面向对象的分析与设计，还支持从需求分析开始的软件开发的全过程。

在这个知识点，要求我们掌握UML的类图、用例图、状态图、顺序图，类之间的关系，以及用例之间的关系。

版权方授权希赛网发布，侵权必究

## UML的结构

---

### 9.4.1 UML的结构

UML规范的结构包括构造块、公共机制和架构三个方面。

#### 1.构造块

构造块也就是基本的UML建模元素、关系和图。

(1) 建模元素：包括结构元素（类、接口、协作、用例、活动类、组件、节点等）、行为元素（交互、状态机）、分组元素（包）、注解元素。

(2) 关系：包括关联关系、依赖关系、泛化关系、实现关系。

(3) 图：图是一组元素的图形表示，在理论上，图可以包含事物及其关系的任何组合。然而，实际上仅出现少量的常见组合，它们要与组成软件密集型系统的体系结构的5种最有用的视图相一致。

#### 2.公共机制

公共机制是指达到特定目标的公共UML方法，主要包括规格说明、修饰、公共分类和扩展机制四种。

(1) 规格说明：规格说明是元素语义的文本描述，它是模型真正的“肉”。

(2) 修饰：UML为每一个模型元素设置了一个简单的记号，还可以通过修饰来表达更多的信息。

(3) 公共分类：包括类元与实体（类元表示概念，而实体表示具体的实体）、接口和实现（接口用来定义契约，而实现就是具体的内容）两组公共分类。

(4) 扩展机制：包括约束（添加新规则来扩展元素的语义）、构造型（用于定义新的UML建模元素）、标记值（添加新的特殊信息来扩展模型元素的规格说明）。

### 3.架构

架构是系统的组织结构，包括系统分解的组成部分、它们的关联性、交互、机制和指导原则，这些内容提供系统设计的信息。具体来说，就是指5个系统视图：

- (1) 逻辑视图：以问题域的语汇组成的类和对象集合。
- (2) 进程视图：可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例。
- (3) 实现视图：对组成基于系统的物理代码的文件和组件进行建模。
- (4) 部署视图：把组件物理地部署到一组物理的、可计算节点上。
- (5) 用例视图：最基本的需求分析模型。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

第9章：面向对象方法

作者：希赛教育软考学院 来源：希赛网 2014年05月21日

## UML的图形

### 9.4.2 UML的图形

UML 2.0包括14种图，本节简单介绍这些图形，其中对经常考查的几种图形给出实例。

#### 1.类图

类图 ( Class Diagram ) 描述了一组类、接口、协作和它们之间的关系。在面向对象系统的建模中所建立的最常见的图就是类图。类图给出系统的静态设计视图。包含主动类的类图给出系统的静态进程视图。

在类图中，每个类分为3个部分，分别是类名、属性和操作（方法）。这3个部分划分成3个格子的长方形。有关类图的例子，请参考9.4.4节。

#### 2.对象图

对象图 ( Object Diagram ) 描述了一组对象以及它们之间的关系。对象图描述了在类图中所建立的事物的实例的静态快照。和类图一样，这些图给出系统的静态设计视图或静态进程视图，但它们是从真实案例或原型案例的角度建立的。

在UML中，对象图与类图具有相同的表示形式。对象图可以看做是类图的一个实例。对象是类的实例；对象之间的链 ( Link ) 是类之间的关联的实例。对象与类的图形表示相似，均为划分成3个格子的长方形（下面的两个格子可省略）。最上面的格子是对象名，对象名带有下划线；中间的格子记录属性值。链的图形表示与关联相似。对象图常用于表示复杂类图的一个实例。

#### 3.构件图

构件图 ( Component Diagram ) 描述了一个封装的类和它的接口、端口以及由内嵌的构件和连接件构成的内部结构。构件图用于表示系统的静态设计实现视图。对于由小的部件构建大的系统来说，构件图是很重要的。构件图是类图的变体。

根据《UML参考手册》的定义：“构件是系统中可替换的物理部分，它包装了实现而且遵从并提供一组接口的实现。”通常来说，每个构件可能包含很多类并实现很多接口。在构件图中，构件可以分为3种类型：

- (1) 实施构件：构成一个可执行系统必要和充分的构件。

(2) 工作产品构件：开发过程的产物，并不是直接地参与可执行系统，而是用来产生可执行系统的中间工作产品。

(3) 执行构件：是作为一个正在执行的系统的结果而被创建的。

构件图是对面向对象系统的物理方面进行建模时要用的3种图之一（另2种是部署图和制品图）。它可以有效地显示一组构件，以及它们之间的关系。构件图中通常包括构件、接口以及各种关系。如图9-1所示。

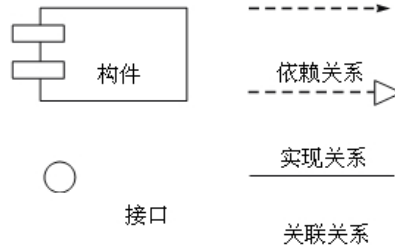


图9-1 构件图中的各种符号

通常可以使用构件图完成4种不同的工作：

(1) 对源代码进行建模：可以清晰地表示出各个不同源程序文件之间的关系。这可以帮助开发团队更好地理解各个源代码文件之间的对应关系，例如：表示C++代码之间、h和.cpp文件之间的关联与包含关系。

(2) 对可执行体的发布建模：清晰地表示出各个可执行文件、DLL文件之间的关系。这可以帮助开发团队更好地理解整个系统中各个执行部分之间的依赖与关联关系。

(3) 对物理数据库建模：用来表示各种类型的数据库、表之间的关系。这可以帮助开发团队对数据库结构之间的关系有一个更清晰的认识。

(4) 对可调整的系统建模：例如：对于应用了负载均衡、故障恢复等系统的建模。

#### 4. 组合结构图

组合结构图 (Composite Structure Diagram) 可以描述结构化类 (例如构件或类) 的内部结构，包括结构化类与系统其余部分的交互点。它显示联合执行包含结构化类的行为的部件配置。组合结构图用于画出结构化类的内部内容。

#### 5. 用例图

用例图 (Use Case Diagram) 描述一组用例、参与者 (一种特殊的类) 及它们之间的关系。用例图给出系统的静态用例视图。这些图在对系统的行为进行组织和建模上是非常重要的。

在UML中，用例表示为一个椭圆。用例的定义是：“用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例则定义一组用例实例。”从这个定义中，我们可以得知用例是由一组用例实例组成的，用例实例也就是常说的“使用场景”，就是用户使用系统的一个实际、特定的场景；其次，可以知道，用例应该给参与者带来可见的价值，这点很关键；最后还可以得知，用例是存在于系统中的。

参与者 (Actor) 也称为外部执行者，它是同系统交互的所有事物，是指代表某一种特定功能的角色，因此参与者都是虚拟的概念。在UML中，用一个小人表示参与者。该角色不仅可以由人承担，还可以是其他系统、硬件设备、甚至是时钟。

用例模型描述的是参与者所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。

有关用例图的例子，请参考9.4.3节。

## 6. 顺序图

顺序图 (Sequence Diagram) 是一种交互图 (Interaction Diagram)，交互图描述了一种交互，它由一组对象或角色以及它们之间可能发送的消息构成。交互图专注于系统的动态视图。

例如，图9-2就是某机票预订系统从订单生成送货单的顺序图。

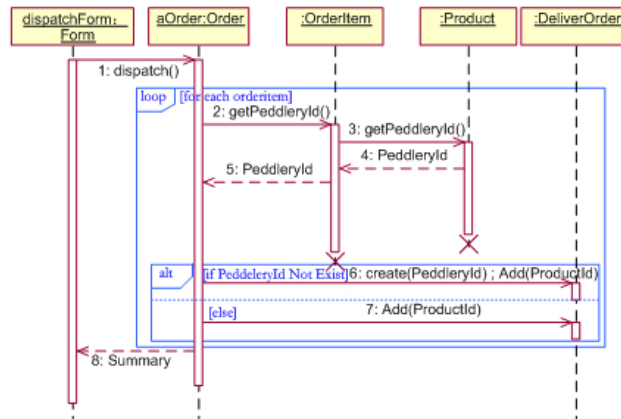


图9-2 某机票预订系统顺序图

如图9-2所示，顺序图存在两个轴，水平轴表示不同的对象，垂直轴表示时间，表示对象及类的生命周期。而对象间的通信通过在对象的生命线间绘制消息来表示。消息的箭头指明消息的类型。顺序图中的消息可以是信号、操作调用或类似于C++中的RPC (Remote Procedure Calls) 和Java中的RMI (Remote Method Invocation)。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。通过在对象生命线上显示一个细长矩形框来表示激活。

消息可以用消息名及参数来标识，消息也可带有顺序号。消息还可带有条件表达式，表示分支或决定是否发送消息。如果用于表示分支，则每个分支是相互排斥的，即在某一时刻仅可发送分支中的一个消息。

## 7. 通信图

通信图 (Communication Diagram) 也是一种交互图，强调收发消息的对象或角色的结构组织。顺序图和通信图表达了类似的基本概念，但每种图所强调概念的不同视图，顺序图是强调消息的时间次序 (时序)，通信图则强调消息流经的数据结构。

例如，图9-3是图9-2对应的通信图。

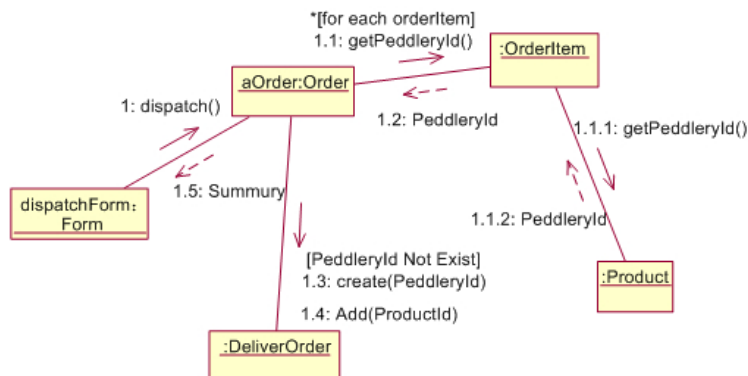


图9-3 某机票预订系统通信图

## 8. 状态图

状态图 (State Diagram) 描述一个状态机，它由状态、转移、事件和活动组成。状态图展现了对象的动态视图。它对于接口、类或协作的行为建模尤为重要，而且它强调事件导致的对象行为，这非常有助于对反应式系统建模。

状态图适合用于描述在不同用例之间的对象行为，但并不适合于描述包括若干协作的对象行

为。通常不会需要对系统中的每一个类绘制相应的状态图，我们可以在进行业务流程、控制对象、用户界面等方面的设计时使用状态图。例如，图9-4就是某机票预订系统的状态图。

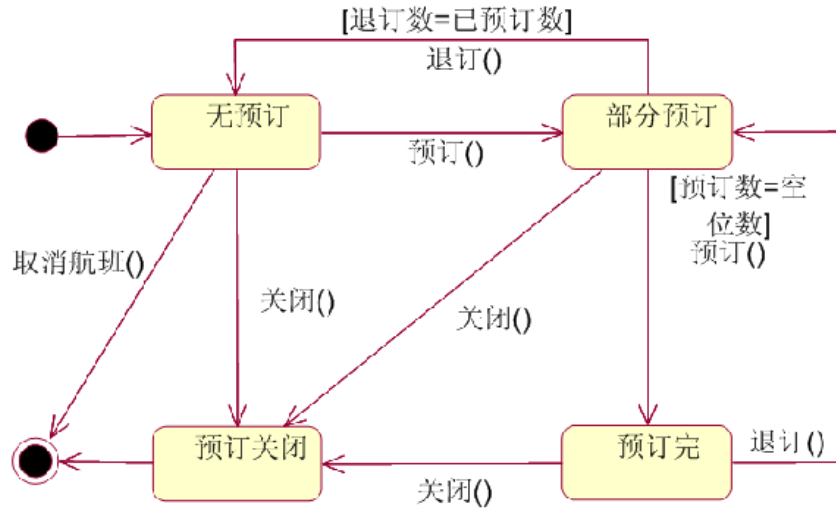


图9-4 某机票预订系统状态图

(1) 状态：又称为中间状态，用圆角矩形框表示。

(2) 初始状态：又称为初态，用一个黑色的实心圆圈表示，在一张状态图中只能够有一个初始状态。

(3) 结束状态：又称为终态，在黑色的实心圆圈外面套上一个空间圆，在一张状态图中可能有多个结束状态。

(4) 状态转移：用箭头说明状态的转移情况，并用文字说明引发这个状态变化的相应事件是什么。

一个状态也可能被细分为多个子状态，那么如果将这些子状态都描绘出来的话，那么这个状态就是复合状态。

### 9.活动图

活动图 (Activity Diagram) 将进程或其他计算的结构展示为计算内部一步步的控制流和数据流。活动图专注于系统的动态视图。它对于系统的功能建模特别重要，并强调对象间的控制流程。

例如：图9-5就是某机票预订系统的活动图。

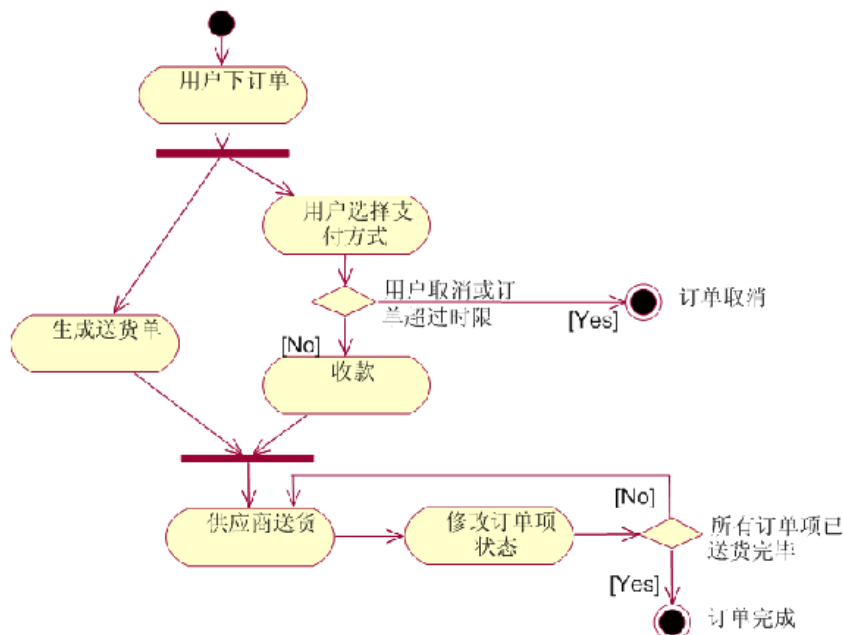


图9-5 某机票预订系统活动图

从图9-5中可以看出，活动图与结构化分析与设计中经常使用的系统流程图十分的相近。

活动图是由状态图变化而来的，它们各自用于不同的目的。活动图依据对象状态的变化来捕获动作（将要执行的工作或活动）与动作的结果。活动图一个活动结束后将立即进入下一个活动（在状态图中状态的变迁可能需要事件的触发）。

活动图包括了初始状态、终止状态，以及中间的活动状态，每个活动之间，也就是一种状态的变迁。在活动图中，还引入了以下几个概念：

（1）判定：说明基于某些表达式的选择性路径，在UML中使用菱形表示。

（2）分叉与结合：由于活动图建模时经常会遇到并发流，因此在UML中引入了如图9-6所示的粗线来表示分叉和结合。

活动图通常应用于事件流较复杂的场合。由于它具有丰富的表现力，因此可以将基本事件流、扩展事件流图形化的表示出来，能够帮助读者更清晰准确地掌握整个用例的事件流程。

为了在简单活动图的基础上，有效地表示各个活动由谁负责的信息，我们可以通过泳道（swim lane）来实现。例如：在图9-5中，活动的主要负责人包括客户、系统、供应商，因此可以将其分成3个泳道，绘制出如图9-6所示的活动图。

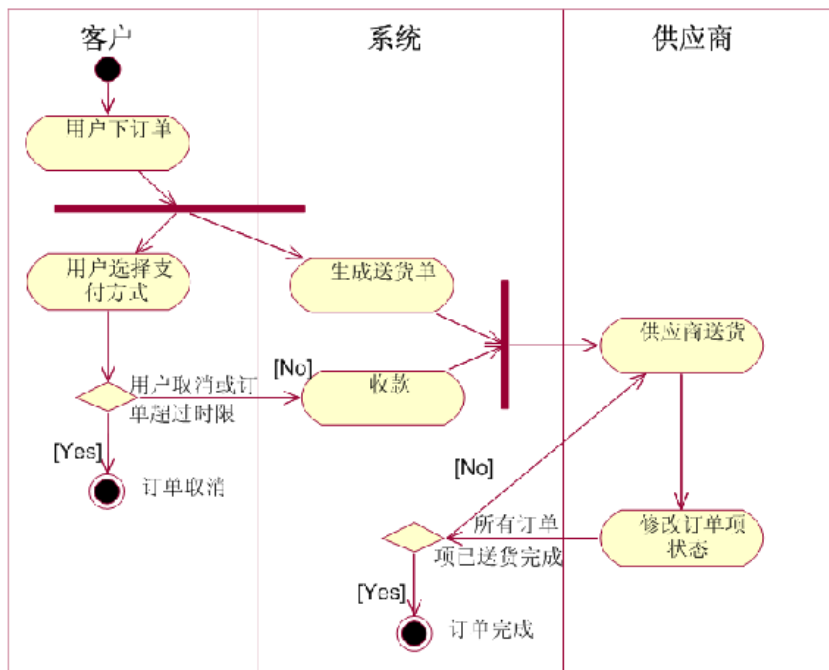


图9-6 带泳道的活动图

在图9-6中，泳道将活动图中的活动节点分成了几个小组，每个小组都显示出了负责实施这些操作的角色。在本图中，这些都是现实世界中的实体，而同样，你也可以用来表示不同的类。

每个泳道在视觉上是用一条垂直的线将它们分开，并且每个泳道都必须有一个唯一的名称，例如本图中的客户、系统、供应商。从图9-6中也可以看出，每个活动节点、分支是必须只属于一个泳道的，而转换、分岔与汇合是可以跨泳道的。通过泳道，我们不仅体现了整个活动控制流，还体现出了每个活动的实施者。

## 10.部署图

部署图（Deployment Diagram）描述了对运行时的处理结点以及在其中生存的构件的配置。部署图也称为实施图，给出了体系结构的静态部署视图，通常一个结点包含一个或多个制品。

我们通过构件图将能够理解系统的物理组成结构，但是它并没有办法体现出这些物理组成部分是如何反映在计算机硬件系统之上的。而部署图正是用来弥补这个不足的，它的关注点就在于系统

如何部署。相对来说，构件图是说明构件之间的逻辑关系，而部署图则是在此基础上更进一步，描述系统硬件的物理拓扑结构以及在此结构上执行的软件。部署图可以显示计算结点的拓扑结构和通信路径、结点上运行的软件构件，常常用于帮助理解分布式系统。部署图通常可以用于以下情况的建模工作：

(1) 对处理器和设备建模：通常包括对单机式、嵌入式、客户/服务器式和分布式系统的拓扑结构的处理器和设备进行建模。

(2) 对构件的分布建模：用来可视化地设定其构件的位置与协作关系。

例如：图9-7就是某系统部署图。

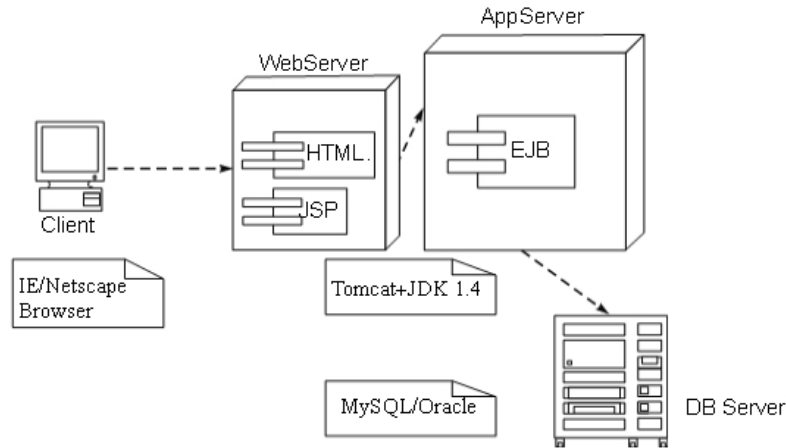


图9-7 某系统的部署图

开发团队通过构建和维护部署图，将可以为维护人员提供足够的技术信息支持，以保证部署、安装、维护工作的顺利实施。

#### 11.制品图

制品图 (Artifact Diagram) 描述计算机中一个系统的物理结构。制品包括文件、数据库和类似的物理比特集合。制品图通常与部署图一起使用。制品图也展现了它们实现的类和构件。

#### 12.包图

包图 (Package Diagram) 描述由模型本身分解而成的组织单元，以及它们的依赖关系。

#### 13.定时图

定时图 (Timing Diagram) 是一种交互图，它描述了消息跨越不同对象或角色的实际时间，而不仅仅是关心消息的相对顺序。

#### 14.交互概览图

交互概览图 (Interaction Overview Diagram) 是活动图和顺序图的混合物。

UML并不限定仅使用这些图，开发工具可以利用UML来提供其他种类的图，但到目前为止，这几种图在实际应用中是最常用的。

版权方授权希赛网发布，侵权必究

上一节 本书简介 下一节



### 9.4.3 用例之间的关系

两个用例之间的关系可以概括为两种情况：一种是用于重用的包含关系，用构造型《include》或《use》表示；另一种是用于分离出不同行为的扩展，用构造型《extend》表示。

#### 1. 包含关系

当可以从两个或两个以上的原始用例中提取公共行为，或者发现能够使用一个构件来实现某一个用例的部分功能很重要时，我们应该使用包含关系来表示它们。

#### 2. 扩展关系

如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种事情。我们可以断定将这个用例分为一个主用例和一个或多个辅用例描述可能更加清晰。

例如：某系统的用例图如图9-8所示，其中就包括了包含关系和扩展关系。

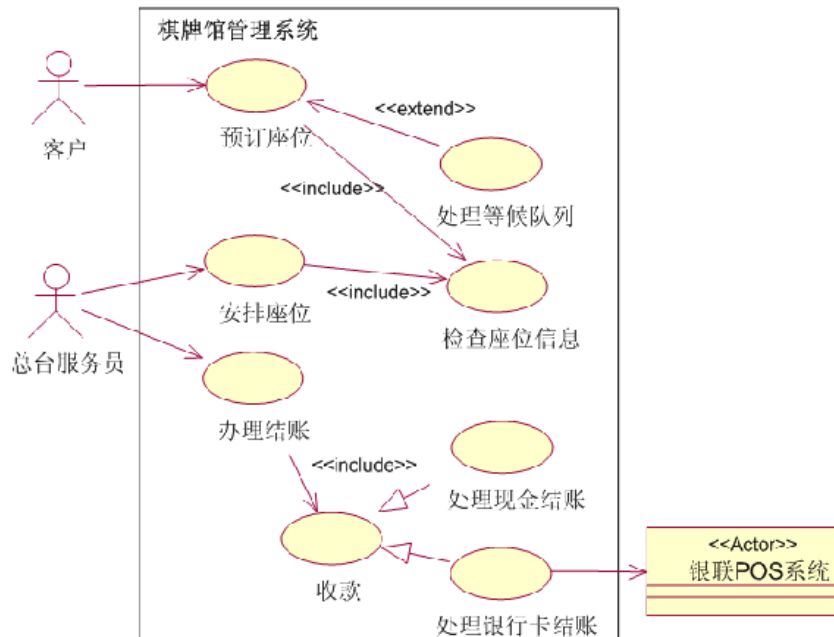


图9-8 某棋牌馆管理系统局部用例图

#### 3. 泛化关系

在必要的时候，也可在用例图中使用泛化关系，它可以用来表示参与者与参与者之间，用例与用例之间的特殊/一般化关系。在用例模型中，泛化关系和类图中的泛化关系是一样的。

对于参与者而言，泛化关系的引用可以有效地降低模型的复杂度，例如，在图9-8所示的用例图，我们希望引入一个“迎宾员”的角色，并且为了缓解总台服务员的压力，希望让迎宾员也能够完成“安排座位”的职责，那么我们就可以通过参与泛化来更有效地组织这个用例图，如图9-9所示。

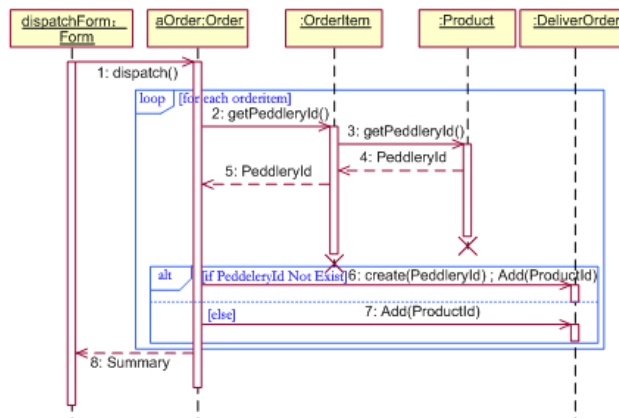


图9-9 通过参与者泛化来简化模型表示

图9-9表示，总台服务员是一种“特殊”的迎宾员，她不仅可以安排座位，还能够办理结账。而用例之间的泛化则表示子用例继承了父用例的行为和含义；子用例还可以增加或覆盖父用例的行为；子用例可以出现在父用例出现的任何位置。

版权方授权希赛网发布，侵权必究

上一节 本书简介 下一节

第9章：面向对象方法


作者：希赛教育软考学院 来源：希赛网 2014年05月21日

## 类之间的关系

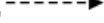
### 9.4.4 类之间的关系

类之间的关系主要有关联关系、依赖关系、泛化关系、聚合关系、组合关系、实现关系、流关系等。

#### 1. 关联关系


描述了给定类的单独对象之间语义上的连接。关联提供了不同类之间的对象可以相互作用的连接。其余的关系涉及类元自身的描述，而不是它们的实例。用“”表示。

#### 2. 依赖关系


有两个元素X、Y,如果修改元素X的定义可能会引起对另一个元素Y的定义的修改，则称元素Y依赖于元素X.在UML中，使用带箭头的虚线“”表示依赖关系。

在类中，依赖由各种原因引起，例如：一个类向另一个类发送消息；一个类是另一个类的数据成员；一个类是另一个类的某个操作参数。如果一个类的接口改变，它发出的任何消息可能不再合法。


#### 3. 泛化关系

泛化关系描述了一般事物与该事物中的特殊种类之间的关系，也就是父类与子类之间的关系。继承关系是泛化关系的反关系，也就是说子类是从父类继承的，而父类则是子类的泛化。在UML中，使用带空心箭头的实线“”表示泛化关系，箭头指向父类。


#### 4. 聚合关系

聚合是一种特殊形式的关联，它是传递和反对称的。聚合表示类之间的关系是整体与部分的关系。例如：一辆轿车包含四个车轮、一个方向盘、一个发动机和一个底盘，就是聚合的一个例子。在UML中，使用一个带空心菱形的实线“”表示聚合关系，空心菱形指向的是代表“整体”的类。

#### 5. 组合关系

如果聚合关系中的表示“部分”的类的存在与否，与表示“整体”的类有着紧密的关系，例如：“公司”与“部门”之间的关系，那么就应该使用“组合”关系来表示这种关系。在UML中，使用带有实心菱形的实线“”表示组合关系。

#### 6. 实现关系

实现关系将说明和实现联系起来。接口是对行为而非实现的说明，而类之中则包含了实现的结构。一个或多个类可以实现一个接口，而每个类分别实现接口中的操作。用“”表示。

#### 7. 流关系

流关系将一个对象的两个版本以连续的方式连接起来表示一个对象的值、状态和位置的转换。流关系可以将类元角色在一次相互作用中连接起来。流的种类包括变成（同一个对象的不同版本）和拷贝（从现有对象创造出一个新的对象）两种，用" $\text{-----}\rightarrow$ "表示。

### 8.多重度

与关联关系相关的概念是多重度（Multiplicity,多重性、重复度）。多重度定义了某个类的一个实例可以与另一个类的多少个实例相关联。通常把它写成一个表示取值范围的表达式或者一个具体的值。

识别关联的多重度是面向对象建模过程中的一个重要步骤。多重度的概念与E-R图中的实体之间的联系有点类似，多重度表示为一个整数范围 $n\dots m$ ，整数 $n$ 定义所连接的最少对象的数目，而 $m$ 则为最多对象数（当不知道确切的 $m$ 时， $m$ 用 $*$ 号表示）。最常见的多重性有 $0\dots 1$ 、 $0\dots *$ 、 $1$ 、 $1\dots *$ 、 $*$ 等。例如：

（1）书与借书记录之间的关系，就应该是1对 $0\dots 1$ 的关系，也就是一本书可以有0个或1个借书记录。

（2）经理与员工之间的关系，则应为1对 $0\dots *$ 的关系，也就是一个经理可以领导0个或多个员工。

（3）学生与选修课程之间的关系，就可以表示为 $0\dots *$ 对 $1\dots *$ 的关系，也就是一个学生可以选择1门或多门课程，而一门课程有0个或多个学生选修。

在解答这类题目时，最关键的是要根据题意来理解它们之间的多重度关系，而不是从概念上，因为多重度是描述类之间的关联的，而类则是对现实对象的抽象。在实际表达时，如果用 $n$ 号代替 $*$ ，也是正确的。例如：图9-10就是一个带重复度的类图。

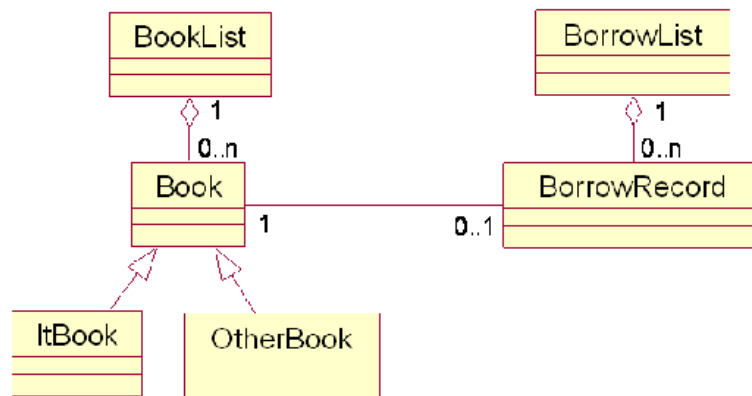


图9-10 类图的实例

在类图中，如果没有指定2个类之间的多重度，则默认为 $*$ 。

版权方授权希赛网发布，侵权必究

上一节 本书简介 下一节

我们可将设计模式 ( Design Pattern ) 简单地理解成设计面向对象的软件开发的经验总结。每个设计模式都系统的命名, 并解释和评价了面向对象系统中一个重要的设计。设计模式的目标是将设计经验收集成人们可以有效利用的模型, 并以目录形式表现出来。

利用设计模式可方便地重用成功的设计和结构。把已经证实的技术表示为设计模式, 使它们更加容易被新系统的开发者所接受。设计模式帮助设计师选择可使系统重用的设计方案, 避免选择危害到可重用性的方案。设计模式还提供了类和对象接口的明确的说明书和这些接口的潜在意义, 来改进现有系统的记录和维护。

版权方授权希赛网发布, 侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

第 9 章：面向对象方法

作者：希赛教育软考学院    来源：希赛网    2014年05月21日

## 设计模式的概念

### 9.5.1 设计模式的概念

在介绍设计模式的具体定义之前, 我们先看一个例子: 模型-视图-控制器 ( Model-View-Controller, MVC ), 在开发人机界面软件时需考虑这种模式。用户界面承担着向用户显示问题模型、与用户进行操作、输入/输出交互的作用。用户希望保持交互操作界面的相对稳定性, 但更希望根据需要改变和调整显示的内容和形式。例如, 要求支持不同的界面标准或得到不同的显示效果, 适应不同的操作需求, 这就要求界面结构能够在不改变软件功能的情况下, 支持用户对界面结构的调整。要做到这一点, 从界面构成的角度看, 困难在于: 在满足对界面要求的同时, 如何使软件的计算模型独立于界面的构成。MVC正是这样的一种交互界面的结构组织模型。

对于界面设计可变性的需求, MVC把交互系统的组成分解成模型、视图、控制三种构件。其中模型构件独立于外在显示内容和形式, 是软件所处理的问题逻辑的内在抽象, 它封装了问题的核心数据、逻辑和功能的计算关系, 独立于具体的界面表达和输入/输出操作; 视图构件把表示模型数据及逻辑关系和状态的信息以特定形式展示给用户, 它从模型获得显示信息, 对于相同的信息可以有多个不同的显示形式或视图; 控制构件处理用户与软件的交互操作, 其职责是决定软件的控制流程, 确保用户界面与模型间的对应联系, 接受用户的输入, 将输入反馈给模型, 进而实现对模型的计算控制, 是使模型和视图协调工作的部件。

模型、视图与控制器的分离, 使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据, 所有其他依赖于这些数据的视图都应反映出这些变化。因此, 无论何时发生了何种数据变化, 控制器都会将变化通知所有的视图, 导致显示的更新。

图9-11的对象模型技术类图描述了MVC解决方案。

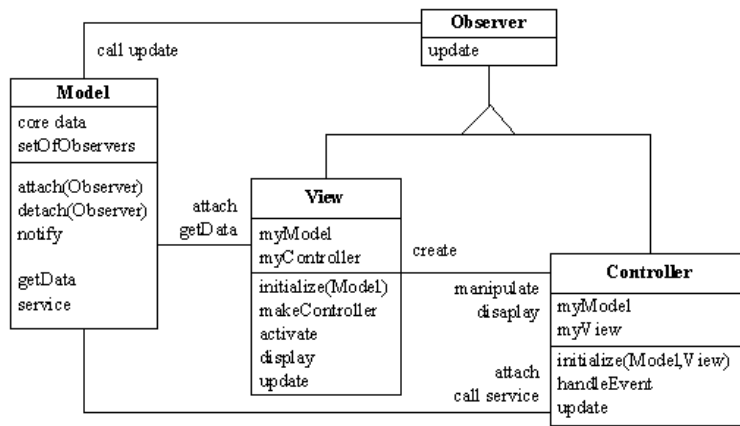


图9-11 MVC解决方案

从上面的例子中，我们可以看出：一个模式关注一个在特定设计环境中出现的重现设计问题，并为它提供一个解决方案，而且为此案提供了一个经过充分验证的通用图示。解决方案图示通过描述其组成构件及其责任和相互关系以及它们的协作方式来具体指定。因此，设计模式不但能重用成功的设计和架构，提高软件质量，而且能适应需求动态的变化。在上面的例子中，问题是支持用户界面的可变性，比如开发人机交互软件系统时，这个问题就会出现。

一个好的模式必须做到以下几点：

- (1) 解决一个问题：从模式可以得到解，而不仅仅是抽象的原则或策略。
- (2) 是一个被证明了的概念：模式通过一个记录得到解，而不是通过理论或推测。
- (3) 解并不是显然的：许多解决问题的方法（例如软件设计范例或方法）是从最基本的原理得到解；而最好的方法是以非直接的方式得到解，对大多数比较困难的设计问题来说，这是必要的。
- (4) 描述了一种关系：模式并不仅仅描述模块，它给出更深层的系统结构和机理。
- (5) 模式有重要的人为因素：所有软件服务于人类的舒适或生活质量，而最好的模式是追求它的实用性和美学。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

## 设计模式的组成

### 9.5.2 设计模式的组成

一般地说，一个模式有以下4个基本组成部分。

#### 1. 模式名称

模式名称通常用来描述一个设计问题，以及它的解法和效果，由一到两个词组成。模式名称的产生使我们可以更高的抽象层次上进行设计并交流设计思想。因此寻找好的模式名称是一个很重也要也是很困难的工作。

#### 2. 问题

问题告诉我们设计模式针对什么背景提出的，解决什么困难。例如：模型-视图-控制器模式关心用户界面经常变化的问题。它可能描述诸如如何将一个算法表示成一个对象这样的特殊设计问题。

在应用这个模式之前，也许还要给出一些该模式的适用条件。

模式的问题陈述用一个强制条件集来表示，说明问题要解决时应该考虑的各个方面，例如：

解决方案必须满足的需求。例如：对等进程间通信必须是高效的。

必须考虑的约束。例如：进程间通信必须遵循特定协议。

解决方案必须具有期望的特性。例如：软件更改应该是容易的。

模型-视图-控制器模式指出了两个强制条件：用户界面必须易于修改，且软件的功能核心不能被修改所影响。一般地，强制条件从多个角度讨论问题并有助于设计师了解它的细节。强制条件可以相互补充或相互矛盾。例如：系统的可扩展性与代码的最小化构成了两个相互矛盾的强制条件。如果希望系统可扩展，那么就应倾向于使用抽象超类；如果想使代码最小化（例如：用于嵌入式系统），就不能承受抽象超类的奢侈。但更重要的是，强制条件是解决问题的关键。它们平衡得越好，对问题的解决方案就越好。所以，强制条件的详细讨论是问题陈述的重要部分。

### 3. 解决方案

解决方案描述设计的基本要素及其关系、各自的任务，以及相互之间的合作。解决方案并不是针对某一个特殊问题而给出的。设计模式提供有关设计问题的一个抽象描述以及如何安排这些基本要素以解决问题。一个模式就像一个可以在许多不同环境下使用的模板，抽象的描述使我们可以把该模式应用于解决许多不同的问题。

模式的解决方案部分给出了如何解决再现问题，或者更恰当地说是如何平衡与之相关的强制条件。在软件体系结构中，这样的解决方案包括2个方面：

(1) 每个模式规定了一个特定的结构，即元素的一个空间配置。例如：MVC模式的描述包括以下语句：“把一个交互应用程序划分成三部分：处理、输入和输出”。

(2) 每个模式规定了运行期间的行为。例如：MVC模式的解决方案部分包括以下陈述：“控制器接收输入，而输入往往是鼠标移动、点击鼠标或键盘输入等事件。事件转换成服务请求，这些请求再发送给模型或视图”。

值得注意的是，解决方案不必解决与问题相关的所有强制条件，而可以集中于特殊的强制条件，对于剩下的强制条件进行部分解决或完全不解决，特别是强制条件相互矛盾时。

### 4. 效果

效果描述应用设计模式后的结果和权衡。比较与其他设计方法的异同，得到应用设计模式的代价和优点。对于软件设计来说，通常要考虑的是空间和时间的权衡，也会涉及语言问题和实现问题。对于一个面向对象的设计而言，可重用性很重要，评价其效果的标准还包括对系统灵活性、可扩充性及可移植性的影响，明确看出这些效果有助于理解和评价设计模式。

版权方授权希赛网发布，侵权必究

[上一节](#)

[本书简介](#)

[下一节](#)

## 设计模式的方法分类

### 9.5.3 设计模式的方法分类

#### 1. Coad的面向对象模式

1992年，美国的面向对象技术的大师Peter Coad从MVC的角度对面向对象系统进行了讨论，

设计模式由最底层的构成部分（类和对象）及其关系来区分。他使用了一种通用的方式来描述一种设计模式：

- (1) 模式所能解决问题的简要介绍与讨论。
- (2) 模式的非形式文本描述及图形表示。
- (3) 模式的使用方针：在何时使用，以及能够与哪些模式结合使用。

我们可以将Coad的模式划分为以下三类：

(1) 基本的继承和交互模式：主要包括面向对象程序设计语言所提供的基本建模功能，继承模式声明了一个类能够在其子类中被修改或被补充，交互模式描述了在有多个类的情况下消息的传递。

(2) 面向对象软件系统的结构化模式：描述了在适当情况下，一组类如何支持面向对象软件系统结构的建模。主要包括条目描述模式、为角色变动服务的设计模式和处理对象集合的模式。

(3) 与MVC框架相关的模式。

几乎所有Coad提出的模式都指明如何构造面向对象软件系统，有助于设计单个的或者一小组构件，描述了MVC框架的各个方面。但是，他没有重视抽象类和框架，没有说明如何改造框架。

## 2.代码模式

代码模式的抽象方式与面向对象程序设计语言中的代码规范很相似，该类模式有助于解决某种面向对象程序设计语言中的特定问题。代码模式的主要目标在于：

- (1) 指明结合基本语言概念的可用方式。
- (2) 构成源码结构与命名规范的基础。
- (3) 避免面向对象程序设计语言（尤其是C++语言）的缺陷。

代码模式与具体的程序设计语言或者类库有关，它们主要从语法的角度对于软件系统的结构方面提供一些基本的规范。这些模式对于类的设计不适用，同时也不支持程序员开发和应用框架，命名规范是类库中的名字标准化的基本方法，以免在使用类库时产生混淆。

## 3.框架应用模式

在应用程序框架“菜谱”（Application Framework Cookbook Recipes）中有很多“菜谱”，它

## 设计模式的方法分类



定的问题。程序员将框架作为应用程序开发并不讲解框架的内部设计实现，只讲如何使用。

用。

不同的框架有各自的“菜谱”，例如：Glenn E. Krasner和Stephen T. Pope在1988年出版的《A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80》书中，提出了如何使用MVC框架的“菜谱”，苹果公司在1989年提出的“菜谱”说明如何利用MacApp的GUI应用程序框架在Macintosh机器上开发应用系统，还有其他一些学者提出了建立图形编辑器框架的“菜谱”等。

实践证明：“菜谱”的概念非常适合于框架的应用，它覆盖了大部分典型应用，但是这些“菜谱”基本上都是不完全的。在“菜谱”中说明的应用情况越多，就越不容易找到相应的“菜谱条”，并且有的应用可以用数种方案来解决，或者要用数种方案的结合来解决，这种模糊性的不清晰性使程序员很容易糊涂。为了避免这样的问题，“菜谱”应该由那些对框架本身有相当深入的理解的人来撰写，最理想的情况是由框架的开发者来撰写。

超文本系统能够很好地支持这种“菜谱”方法，更高级的超文本系统（如HTML+）已经超出了简

单的应用“菜谱”的范畴，它们还可以基于设计模式方法（例如：设计模式目录，元模式等）来对框架的设计做文档。

#### 4.形式合约

形式合约（Formal Contracts）也是一种描述框架设计的方法，强调组成框架的对象间的交互关系。有人认为它是面向交互的设计，对其他方法的发展有启迪作用。但形式化方法由于其过于抽象，而有很大的局限性，仅仅在小规模程序中使用。

Richard Helm等人是形式合约模式的倡导者，他们最先在面向对象系统领域内探索用抽象的方法来描述被他们称为行为合成（Behavioral Composition）的内容。他们所使用的规范符号有如下优点：

（1）符号所包含的元素很少，并且其中引入的概念能够被映射成为面向对象程序设计语言中的概念。例如：参与者映射成为对象。

（2）形式合约中考虑到了复杂行为是由简单行为组成的事实，合约的修订和扩充操作使得这种方法很灵活，易于应用。

形式合约模式的缺点有以下三点：

（1）在某些情况下很难用，过于繁琐。若引入新的符号，则又使符号系统复杂化。

（2）强制性地要求过分精密，从而在说明其中可能发生隐患（例如冗余）。

（3）形式合约的抽象程度过低，接近面向对象的程序设计语言，不易分清主次。

版权方授权希赛网发布，侵权必究

[上一节](#)    [本书简介](#)    [下一节](#)

## 设计模式目录的内容

### 9.5.4 设计模式目录的内容

Gamma在他的博士论文中总结了一系列的设计模式，做出了开创性的工作。他用一种类似分类目录的形式将设计模式记载下来。我们称这些设计模式为设计模式目录。根据模式的目标（所做的事情），可以将它们分成创建性模式（creational）、结构性模式（structural）和行为性模式（behavioral）。创建性模式处理的是对象的创建过程，结构性模式处理的是对象/类的组合，行为性模式处理类和对象间的交互方式和任务分布。根据它们主要的应用对象，又可以分为主要应用于类的和主要应用于对象的。

表9-3是Gamma总结的23种设计模式。

表9-3 设计模式目录的分类



目的	设计模式	简要说明	可改变的方面
创建性	Abstract Factory	提供创建相关的或相互信赖的一组对象的接口，使我们不需要指定类	产品对象族
	Builder	将一个复杂对象的结构与它的描述隔离开来，使我们使用相同的结构可以得到不同的描述	如何建立一种组合对象
	Factory Method*	定义一个创建对象的接口，但由于子类决定需要实例化哪一个类	实例化子类的对象
	Prototype	使用一个原型来限制要创建的类的类型，通过拷贝这个原型得到新的类	实例化类的对象
	Singleton	保证一个类只有一个实例，并提供一个全局性的访问点	类的单个实例
结构性	Adapter*	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	与对象的接口
	Bridge	将类的抽象概念和它的实现分离开来，使它们可以相互独立地变化	对象的实现
	Composite	将对象组成树结构来表示局部和整体的层次关系。客户可以统一处理单个对象和对象组合	对象的结构和组合
	Decorator	给对象动态地加入新的职责。它提供了用子类扩展功能的一个灵活的替代	无子类对象的责任
	Facade	给一个子系统的所有接口提供一个统一的接口。它定义了更高层的接口，使该系统更便于使用	与子系统的接口
	Flyweight	提供支持大量细粒度对象共享的有效方法	对象的存储代价
	Proxy	给另一个对象提供一个代理或定位符号，以控制对它的访问	如何访问对象，对象位置
行为性	Chain of Responsibility	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求	可满足请求的对象
	Command	将一个请求封装为一个对象，从而将不同的请求对象化并进行排队或登记，以支持撤销操作	何时及如何满足一个请求

续表

目的	设计模式	简要说明	可改变的方面
行为性	Interpreter*	给定一种语言，给出它的语法的一种描述方法和一个解释器，该解释器用这种描述方法解释语言中的句子	语言的语法和解释
	Iterator	提供一种顺序性访问一个聚集对象中元素的方法，而不需要暴露它的底层描述	如何访问、遍历聚集的元素
	Mediator	定义一个对象来封装一系列对象的交互。它保持对象间避免显式地互相联系，而消除它们间的耦合，还可以独立地改变对象间的交互	对象之间如何交互及哪些对象交互
	Memento	在不破坏封装的条件下，获得一个的内部状态并将它外部化，从而可以在以后使对象恢复到这个状态	何时及哪些私有信息存储在对象之外
	Observer	定义一个对象间一对多的信赖关系，当一个对象改变状态时，所有与它有信赖关系的对象都得到通知并自动更新	信赖于另一对象的对象数量，信赖对象如何保持最新数据
	State	允许一个对象在内部状态改变时的行为，对象看起来似乎能改变自己的类	对象的状态
	Strategy	定义一族算法，对每一个都进行封装，使它们互相可交换。它使算法可以独立于它的用户而变化	算法
	Template Method*	在方法中定义算法的框架，而将算法中的一些操作步骤延迟到子类中实现	算法的步骤
	Visitor	描述在一个对象结构中对某个元素需要执行的一个操作。它使我们在不改变被操作的元素类的条件下定义新操作	无需改变其类而可应用于对象的操作

其中带\*为关于类的，其他是关于对象的。

版权方授权希赛网发布，侵权必究

上一节 本书简介 下一节