第 8 章　面向对象开发方法

本章内容

8.1 面向对象方法概述

面向对象技术考虑问题的基本原则是，尽可能模拟人类习惯的思维方式。面向对象使描述问题的问题空间（也称为问题域）与实现解法的解空间（也称为求解域）在结构上尽可能一致。

8.1.1 面向对象方法的主要优点

与人类习惯的思维方式一致

稳定性好

可重用性好

较易开发大型软件产品

可维护性好

8.1.2 面向对象方法的主要概念

01 OPTION

对象

1 对象的定义

- 对象可以是具体物理实体的抽象、人为的概念、任何有明确边界和意义的事物等
- 由一组属性和对这组属性进行操作的一组方法（服务）组成

2 对象的特点

- 以数据为核心
- 主动性
- 数据封装
- 本质上具有并行性
- 模块独立性好

8.1.2 面向对象方法的主要概念

02 OPTION 类

类（Class）是具有相同属性和相同方法的一组对象的集合。它为属于该类的全部对象提供了统一的抽象描述。同类对象具有相同的属性和方法，属性的定义形式相同，每个对象的属性值不同。

8.1.2 面向对象方法的主要概念

03 OPTION 实例

实例（Instance）是由某个特定的类描述的一个具体对象。一个对象是类的一个实例。例如学生是一个类，某位学生张三就是学生类的一个实例，即对象。

8.1.2 面向对象方法的主要概念

04 OPTION 属性

属性（Attribute）是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。类的每个实例具有自己特定的属性值。

例如学生类的实例—每位学生有自己特定的姓名、学号、性别、年龄等，所以根据系统的需求，可以定义学生类的属性包括姓名、学号、性别、年龄等。

8.1.2 面向对象方法的主要概念

05 OPTION 消息

消息（Message）就是向对象发出的服务请求，包含提供服务的对象标识、服务（方法）标识、输入信息、回答信息等。

消息可分为同步消息和异步消息。同步消息的发送者要等待接收者的返回。异步消息的发送者在发送消息后继续自己的活动，不等待消息接收者返回信息。面向对象的消息和函数调用是不同的，函数调用往往是同步的，调用者要等待接收者返回信息。

8.1.2 面向对象方法的主要概念

06 OPTION 方法

方法（Method）是对象所能执行的操作，也就是类中所定义的服务。方法描述了对象执行操作的算法，以及响应消息的方法。

07 OPTION 封装

封装（Encapsulation）就是把对象的属性和方法结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节。封装使对象形成接口部分和实现部分两个部分。

8.1.2 面向对象方法的主要概念

08 OPTION

继承

1 单继承

- 一个类只允许有一个父类，即类等级的数据结构为树形结构时，类的继承是单继承。

2 多继承

- 当一个类有多个父类时，类的继承是多继承，此时类的数据结构为网状结构或图

8.1.2 面向对象方法的主要概念

09 OPTION

多态性

多态性（Polymorphism）就是指有多种形态。在面向对象技术中，多态是指一个实体在不同条件下具有不同意义或用法的能力。

不同层次的类可以共享一个行为的名字，当对象接收到消息时，根据对象所属的类动态选用该类所定义的算法。

8.1.2 面向对象方法的主要概念

10 OPTION

重载

1 函数重载

- 在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名

2 运算符重载

- 同一运算符可以施加于不同类型的操作数

本章内容

8.2.1 UML 的发展

UML 采用了面向对象的概念，引入了各种独立于语言的表示符号。UML 通过建立用例模型、静态模型和动态模型完成对整个系统的建模，所定义的概念和符号可用于软件开发过程的分析、设计和实现的全过程，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换。

OOSE 方法的最大特点是面向用例。

8.2.2 UML 的设计目标

设计人员为UML 设定了以下目标。

为所有建立模型的人员提供通用的建模语言

尽可能简洁，但又有足够的表达能力

对良好设计实践的支持，如封装、框架、目标捕获、分布、并发、模式及协作等

支持现代迭代构造方法，如用例驱动、建立强壮结构

包含所有面向对象概念

8.2.2 UML 的设计目标

UML 现在已经做到了以下几点。

- 运用面向对象概念来构造任何系统模型。
- 对人和计算机都适用的可视化建模语言。
- 支持独立于编程语言和开发过程的规格说明。
- 建立概念模型与可执行体之间的对应关系。
- 提供可扩展机制和特殊化机制。
- 支持更高级的开发概念，如组件、协作、模式、框架等。

8.2.3 UML 的内容

01
OPTION

UML 语义

UML 的语义是定义在一个建立模型的框架中的，建模框架有如下4 个层。

基本元素层
01

由UML 的基本元素组成，代表要定义的所有事物

元模型层
02

由UML 的基本元素层组成，包括面向对象和面向构件的概念

静态模型层
03

由UML 静态模型组成，静态模型描述系统的元素及元素间的关系，常称为类模型

用例模型层
04

由UML 用例模型组成，每个概念是静态模型层的一个实例，也是元模型层的一个实例

8.2.3 UML 的内容

02 OPTION UML 表示法

- 图

用例图 01

用于表示系统的功能，并指出各功能的执行者

静态图 02

包括类图、对象图及包，表示系统的静态结构

行为图 03

包括状态图和活动图，用于描述系统的动态行为和对象之间的交互关系

交互图 04

包括顺序图和协作图，用于描述系统的对象之间的动态合作关系

实现图 05

包括构件图和部署图，用于描述系统的物理实现

8.2.3 UML 的内容

- 视图

视图由若干张图构成，从不同的目的或角度描述系统。

- 模型元素

图中使用的概念，例如用例、类、对象、消息和关系，统称为模型元素。模型元素在图中用相应的图形符号表示。一个模型元素可以在多个不同的图中出现，但它的含义和符号是相同的。

8.2.3 UML 的内容

- 通用机制

字符串：用于表示有关模型的信息

名字：用于表示模型元素

标号：用于表示附属于图形符号的字符

特定字符串：用于表示附属于模型元素的特性

类型表达式：用于声明属性变量和参数

8.2.3 UML 的内容

- 扩展机制

UML 的扩展机制使它能够适应一些特殊方法或满足用户的某些特殊需要。扩展机制用标签、约束、版型来表示。

8.2.3 UML 的内容

03
OPTION

UML 模型

UML 可以建立系统的用例模型、静态模型和动态模型，每种模型都由适当的UML图组成。

用例模型描述用户所理解的系统功能

静态模型描述系统内的对象、类、包以及类与类、包与包之间的相互关系等

动态模型描述系统的行为，描述系统中的对象通过通信相互协作的方式、对象在系统中改变状态的方式等

8.2.4 UML 的扩展机制

01 OPTION 标签

利用标签（标记）可以增加模型元素的信息。每个标签代表一种性质，能应用于多个元素。可把性质定义成一个标签名和标签值。标签名和标签值都用字符串表示，且用花括号标注。标签值是布尔值true（真）时，可以省略不写。

8.2.4 UML 的扩展机制

02 OPTION 约束

{abstract}：用于类的约束，表明该类是一个抽象类

{complete}：用于关系的约束，表明该关系是一个完全分类

{hierarchy}：用于关系的约束，表明该关系是一个分层关系

{ordered}：用于多重性的约束，表明目标对象是有序的

{bag}：用于多重性的约束，表明目标对象多次出现、无序

8.2.4 UML 的扩展机制

03 OPTION 版型

版型（Stereotype）能把UML 已经定义的元素的语义专有化或扩展。UML 中预定义了40多种版型，与标签和约束一样，用户可以自定义版型。版型的图形符号是《》，符号中间为版型名，如图所示。

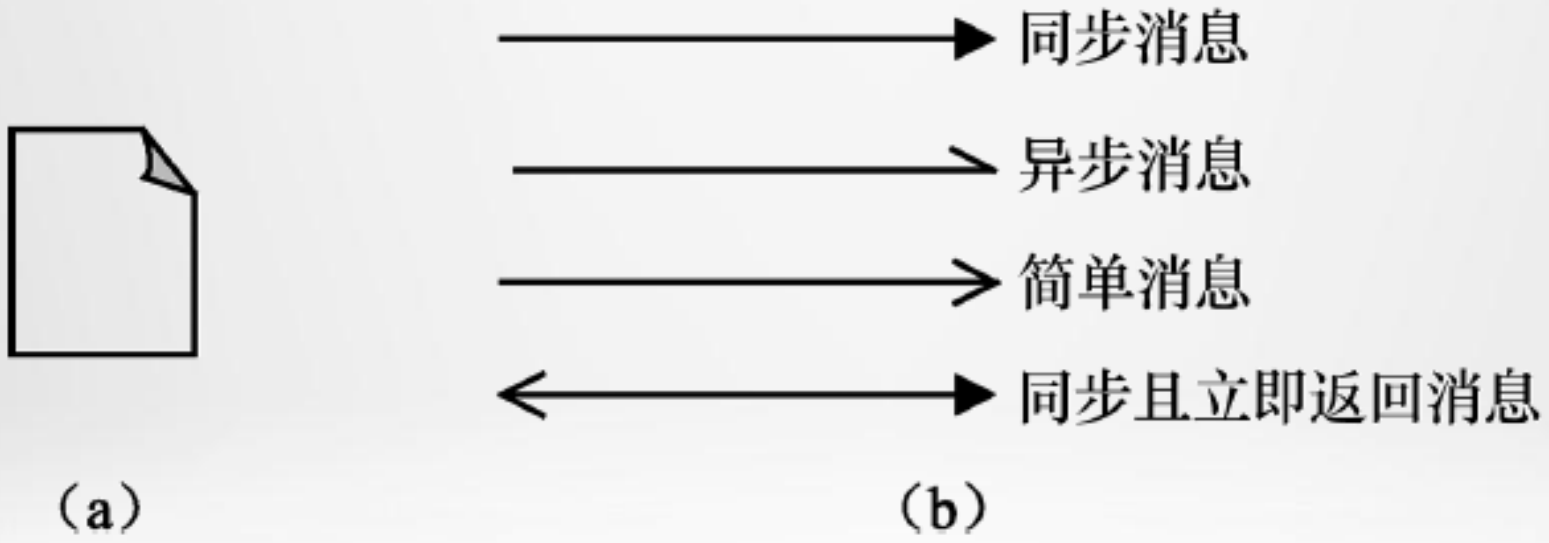
（a）　　（b）

本章内容

8.3 UML 图

01 OPTION 注释

折角矩形是注释的符号，折角矩形中的文字是注释内容，如图所示。



8.3 UML 图

02 消息

OPTION

对象之间的交互是通过传递消息完成的。UML 图中的消息都用从发送者连接到接收者的箭头线表示。

1 简单消息

表示简单的控制流，只表示消息从一个对象传给另一个对象，没有描述通信的任何细节

2 同步消息

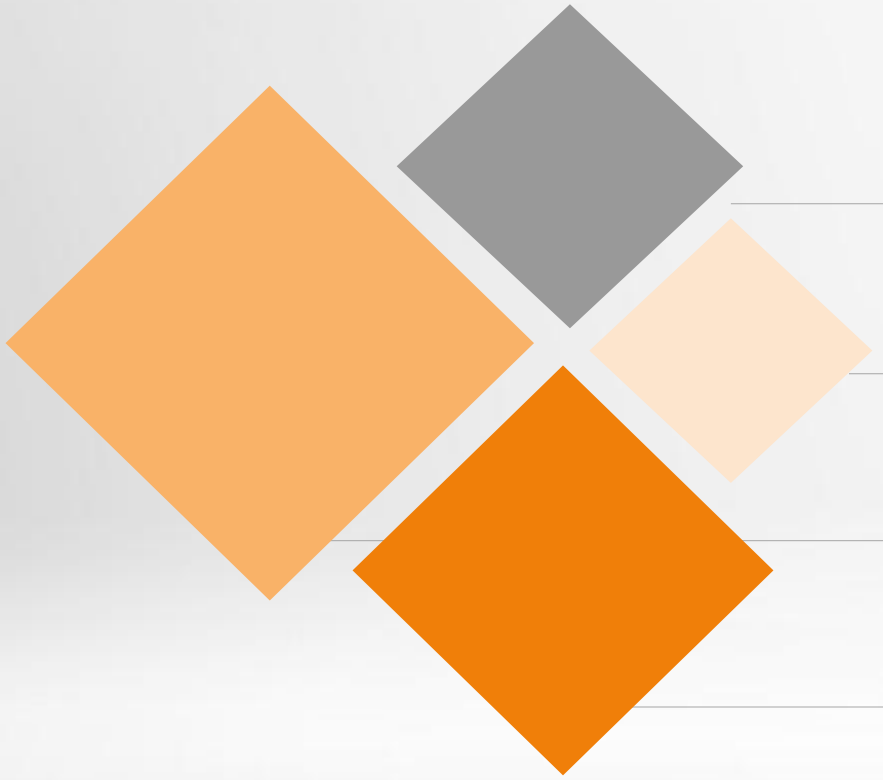
表示调用者发出消息后必须等待消息返回，只有当处理消息的操作执行完毕后，调用者才可以继续执行自己的操作

3 异步消息

发送者发出消息后，不用等待消息处理完就可以继续执行自己的操作

8.3.1 用例图

01 OPTION 用例



用例是一个类，它代表一类功能而不是使用该功能的某一具体实例

用例代表某些用户可见的功能，实现一个具体的用户目标

用例由执行者激活，并提供确切的值给执行者

用例可大可小，但必须是对一个具体用户目标实现的完整描述

8.3.1 用例图

02 OPTION 执行者

执行者也称为角色，用一个小人图形表示

执行者是与系统交互的人或物

执行者是能够使用某个功能的一类人或物

8.3.1 用例图

03 OPTION 通信联系

执行者和用例之间要交换信息，称为通信联系。执行者与用例之间用线段连接，表示两者之间进行通信联系。

执行者不一定是一个具体的人，可能是使用该系统的其他系统或设备等，但都用小人图形来表示。执行者激活用例，并与用例交换信息。单个执行者可与多个用例联系；反过来，一个用例也可与多个执行者联系。对于同一个用例，不同执行者起的作用也可不同。

8.3.1 用例图

04 OPTION 脚本（用例描述）

用例的实例是系统的一种实际使用方法，称为脚本，是系统的一次具体执行过程。用例图中应尽可能包含所有的脚本，这样才能较完整地从用户使用的角度来描述系统的功能。

8.3.2 类图

01 OPTION 类图的符号

1 类的名称

类的名称是名词，应当含义明确、无歧义。

2 类的属性

类的属性描述该类对象的共同特性。类的属性的值应能描述并区分该类的每个对象。

3 类的操作

类的操作用于修改、检索类的属性或执行某些动作。操作只能用于该类的对象上。

8.3.2 类图

02 OPTION 类的关联关系

普通关联

两个类之间的普通关联关系用直线连接来表示。类的关联关系有方向时，用黑三角形表示方向，可在方向上起名字，也可不起名字

关联类

为了详细说明类与类之间的关联，可以用关联类来记录关联的一些附加信息，关联类与一般类一样可以定义其属性和操作

限定关联

在一对多或多对多的关联关系中，可以用限定词将关联变成一对一的限定关联，限定词放在关联关系末端的一个小方框内

聚集

聚集表示类与类之间的关系是整体与部分的关系

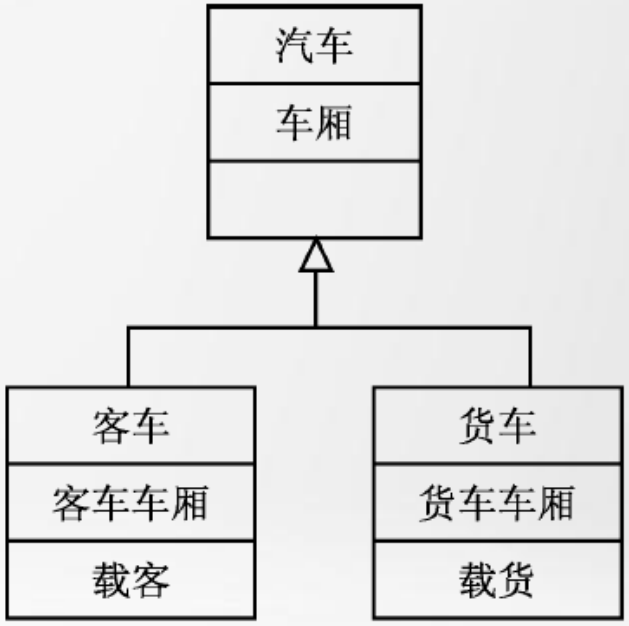
8.3.2 类图

03 OPTION

类的一般- 特殊关系

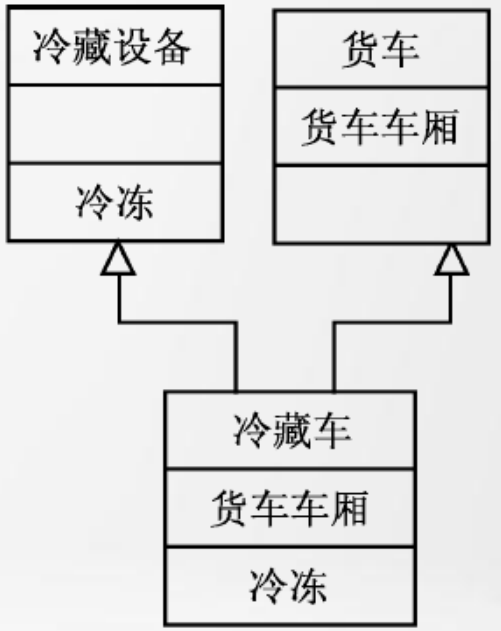
类与若干个互不相容的子类之间的关系称为一般- 特殊关系，或称为泛化关系。事物往往既有共同性，也有特殊性。同样，一般化类中有时也有特殊类。

如果类B 具有类A 的全部属性和全部服务，而且具有自己的特性或服务，则B叫作A的特殊类，A 叫作B 的一般化类。



8.3.2 类图

特殊类的对象拥有其一般化类的全部属性和服务，称作特殊类对一般化类的继承。继承就是“自动地拥有”，因而特殊类不必重新定义一般化类中已定义过的属性和服务，只需要声明它是某个类的特殊类，定义它自己特殊的属性和服务。特殊类中可能还存在下一层的特殊类。



8.3.2 类图

04 OPTION 类的依赖关系

有依赖关系的两个类用带箭头的虚线连接，箭头指向独立的类。如图所示，类A是独立的，类B 以某种方式依赖于类A，如果类A 改变了，将影响依赖于它的类B 中的元素。如果一个类向另一个类发送消息，一个类使用另一个类的对象作为操作的参数或者作为它的数据成员等，这样的两个类之间都存在依赖关系。表示依赖关系的虚线可以带一个版型标签，版型名写在《》内，具体说明依赖的种类。

8.3.2 类图

05 OPTION

类的细化关系

1 概念层类图

在需求分析阶段用概念层类图描述应用领域中的概念

2 说明层类图

在设计阶段用说明层类图描述软件的接口部分

3 实现层类图

在实现阶段用实现层类图描述软件系统中类的实现

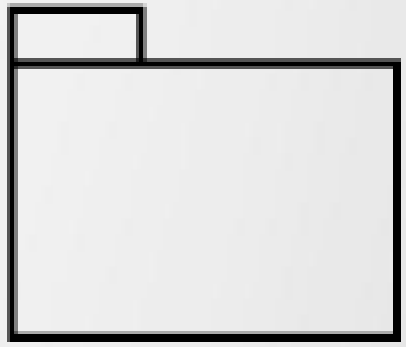
8.3.2 类图

06 包

OPTION

包（Package）是一种组合机制，像一个“容器”，可以组织模型中的相关元素，是把各种各样的模型元素通过内在的语义关系连接在一起形成的一个高内聚、低耦合的整体。包通常用于对模型的管理，有时可把包称为子系统。

包的图示符号由两个矩形组成，小矩形位于大矩形的左上方，如图所示。包的名称可以写在小矩形内，也可以写在大矩形内。



8.3.2 类图

设计包时必须遵守以下原则。

重用等价原则 01

把包作为可重用的单元

共同闭包原则 02

把需要同时改变的类放在一个包中

共同重用原则 03

不会一起使用的类或包不要放在同一包中

非循环依赖原则 04

包和包之间的依赖关系不要形成循环

8.3.3 对象图

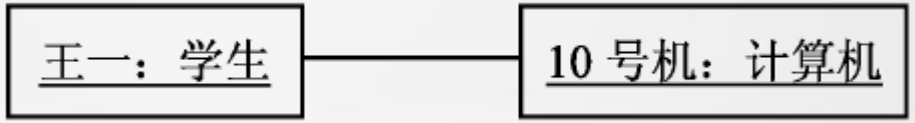
对象是类的实例。因此，对象图（Object Diagram）可以看作类图的实例，能帮助人们理解比较复杂的类图。类图与对象图之间的区别是，对象图中对象的名字下面要加下画线。对象名与类名之间用冒号连接，一起加下画线。如果只有类名没有对象名，类名前一定要加冒号，冒号和类名同时要加下画线。另外，也可以只写对象名并加下画线，类名及冒号省略。

对象有以下3 种表示方式。

（1）对象名：类名。

（2）：类名。

（3）对象名。



8.3.4 状态图

状态转换（转移）是指两个状态之间的关系，它描述了对象从一个状态进入另一个状态的情况，并执行了所包含的动作。

- 椭圆或圆角矩形：表示对象的一种状态，椭圆或圆角矩形内部填写状态名。
- 箭头：表示从箭头出发的状态可以转换到箭头所指向的状态。
- 事件：箭头线上方可标出引起状态转换的事件名。
- 方括号（[ ]）：事件名后面可加方括号，方括号内填写状态转换的条件。
- 实心圆（●）：指出该对象被创建后所处的初始状态。
- 内部实心的同心圆（◉）：表示对象的最终状态。

8.3.5 顺序图

顺序图（Sequence Diagram）描述对象之间动态交互的情况，着重表示对象间消息传递的时间顺序。顺序图中的对象用矩形框表示，矩形框内标有对象名。顺序图有以下两个方向。

1 从上到下

- 从表示对象的矩形框开始，从上到下代表时间的先后顺序，并表示某段时间内该对象是存在的

2 水平方向

- 水平方向的箭头指示了不同对象之间传递消息的方向

8.3.6 活动图

活动图（Activity Diagram）是状态图的一种特殊情况，不需指明任何事件，只要动作被执行，活动图中的状态就自动开始转换。当状态转换的触发事件是外部事件时，常用状态图来表示。如果状态转换的触发事件是内部动作的完成，即可用活动图描述。

在活动图中，用例和对象的行为中的各个活动之间通常具有时间顺序，活动图描述了这种顺序，展示出对象执行某种行为时或者在业务过程中所要经历的各个活动和判定点。每个活动用一个圆角矩形表示，判定点用菱形表示。

8.3.7 协作图

协作图（Collaboration Diagram）用于描述系统中相互协作的对象之间的交互关系和关联链接关系。协作图和顺序图都是描述对象间的交互关系，但它们的侧重点不同，顺序图着重表示交互的时间顺序，协作图着重表示交互对象的静态链接关系。

协作图中对象图示与顺序图相同。对象之间的连线代表对象之间的关联和消息传递，每个消息箭头都带有一个消息标签。消息标签的语法如下：

前缀 [条件] 序列表达式 返回值 ： = 消息说明

8.3.8 构件图

01 OPTION 构件的类型

1 源构件

- 实现类的源代码文件

2 二进制构件

- 一个对象代码文件、一个静态库文件或一个动态库文件

3 可执行构件

- 一个可执行的程序文件，是链接所有二进制构件所得到的结果

8.3.8 构件图

02 OPTION 构件图的表示符号

- 构件的图示符号是左边带有两个小矩形的大矩形，构件的名称写在大矩形内。
- 构件的依赖关系用一条带箭头的虚线表示，箭头的形状表示消息的类型。
- 构件的接口：从代表构件的大矩形框画出一条线，线的另一端为小空心圆，接口的名字写在小空心圆附近。

8.3.9 部署图

01 OPTION 结点和连接

结点（Node）是一种代表运行时计算资源的分类器。一般来说，结点至少要具备存储功能，而且常常具有处理能力。运行时对象和构件可驻留在结点上。

结点可代表一个物理设备以及在该设备上运行的软件系统，例如一个服务器、一台计算机、一台打印机、一台传真机等。结点用一个长方体表示，结点名放在长方体的左上角。

结点间的连线表示系统之间进行交互的通信线路，在UML 中称为连接。通信的类型写在表示连接的线旁，以指定所用的通信协议或网络类型。结点的连接是关联，可以加约束、版型、多重性等符号。

8.3.9 部署图

02 构件和接口

OPTION

部署图中的构件代表可执行的物理代码模块（可执行构件的实例），在逻辑上可以和类图中的包或类对应。因此，部署图显示运行时各个包或类在结点中的分布情况。

在面向对象方法中，类和构件的操作和属性对外并不都是可见的，类和构件等元素对外提供的可见操作和属性称为接口。接口用一端是小空心圆的直线来表示。

8.3.9 部署图

03 OPTION 对象

部署图中的构件是包或类对应的物理代码模块，因此，构件中应包含一些运行的对象。部署图中的对象与对象图中的对象的表示方法相同。

本章内容

8.4.1 面向对象分析过程

面向对象分析过程的第一步是要分析得到完整、准确的用户需求。分析用户需求陈述时，要发现和改正原始陈述中有二义性和不一致性的内容，补充、修改遗漏的内容。通过反复与用户讨论、协商和交流，并通过深入地调查研究，得到更完整、更准确的用户需求陈述。第二步是根据用户需求陈述对事物进行抽象，并用模型准确地表达系统需求。

8.4.2 面向对象分析原则

面向对象分析的基础是对象模型。对象模型由问题域中的对象及其相互的关系组成。最重要的是，一定要把在应用领域中有意义的、与所要解决的问题有关系的所有事物作为对象。既不能遗漏所需的对象，也不能定义与问题无关的对象。

包含原则 对现实世界中的事物进行抽象时，强调对象的各个实例的相似方面。

排斥原则 对不能抽象成某一对象的实例，要明确地排斥。

本章内容

8.5.1 确定对象和类

对象

对象是系统中用来描述客观事物的一个实体，是构成系统的一个基本单位，由一组属性和对这组属性进行操作的一组服务构成。

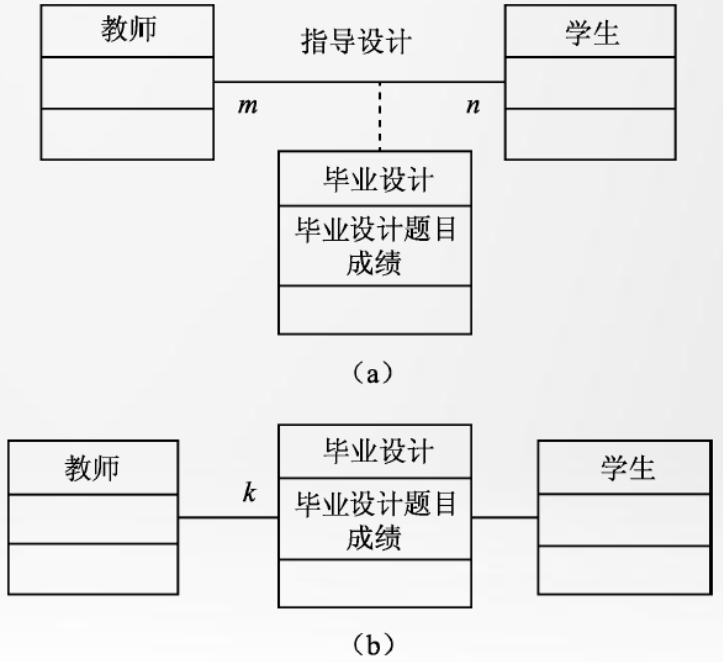
类

类是具有相同属性和服务的一组对象的集合，为属于它的全部对象提供了统一的抽象描述（属性和服务）。

8.5.2 确定类的相互关系

01 OPTION 关联关系

类的关联关系反映了对象之间相互依赖、相互作用的关系。

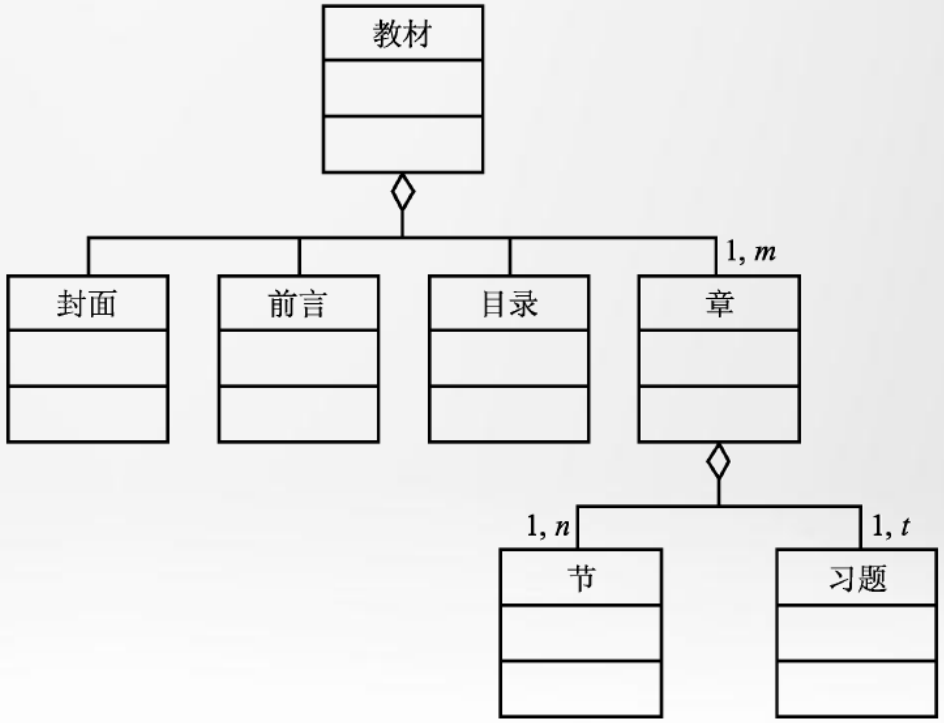


8.5.2 确定类的相互关系

02 OPTION 整体- 部分关系

整体- 部分关系就是聚集关系，它反映了对象之间的构成关系。聚集关系最重要的性质是传递性。

当聚集关系有多个层次时，可以用一棵简单的聚集树来表示它。



8.5.2 确定类的相互关系

02 OPTION 一般- 特殊关系

前文已提到，类与若干个互不相容的子类之间的关系称为一般- 特殊关系。继承是面向对象方法中一个十分重要的概念，是面向对象技术可以提高软件开发效率的一个重要原因。子类继承父类所定义的属性和操作，又可定义本身的特殊属性和操作。

8.5.3 划分主题

01 OPTION 主题

主题（Subject）是把一组具有较强联系的类组织在一起而得到的类的集合，有以下几个特点。

- 主题是由一组类构成的集合，但其本身不是一个类。
- 一个主题内部的对象具有某种意义上的内在联系。
- 主题的划分有一定的灵活性，强调的重点不同可以得到不同的主题划分。

8.5.3 划分主题

02 OPTION 主题图

面向对象分析时，可将问题域中的类图划分成若干个主题。主题的划分无论采用自顶向下还是自底向上方式，最终结果都是一个完整的类图和一个主题图。

主题图有展开方式、压缩方式和半展开方式3 种表示方式。关系较密切的对象画在一个框内，框的每个角标上主题号，框内是详细的类图，标出每个类的属性和服务以及类之间的详细关系，就可得到展开方式的主题图；如果将每个主题号及主题名分别写在一个框内，就可得到压缩方式的主题图；每个框内将主题号、主题名及该主题中所含的类全部列出，就可得到半展开方式的主题图。

压缩方式的主题图可表明系统的总体情况，展开方式的主题图则可用于了解系统的详细情况。

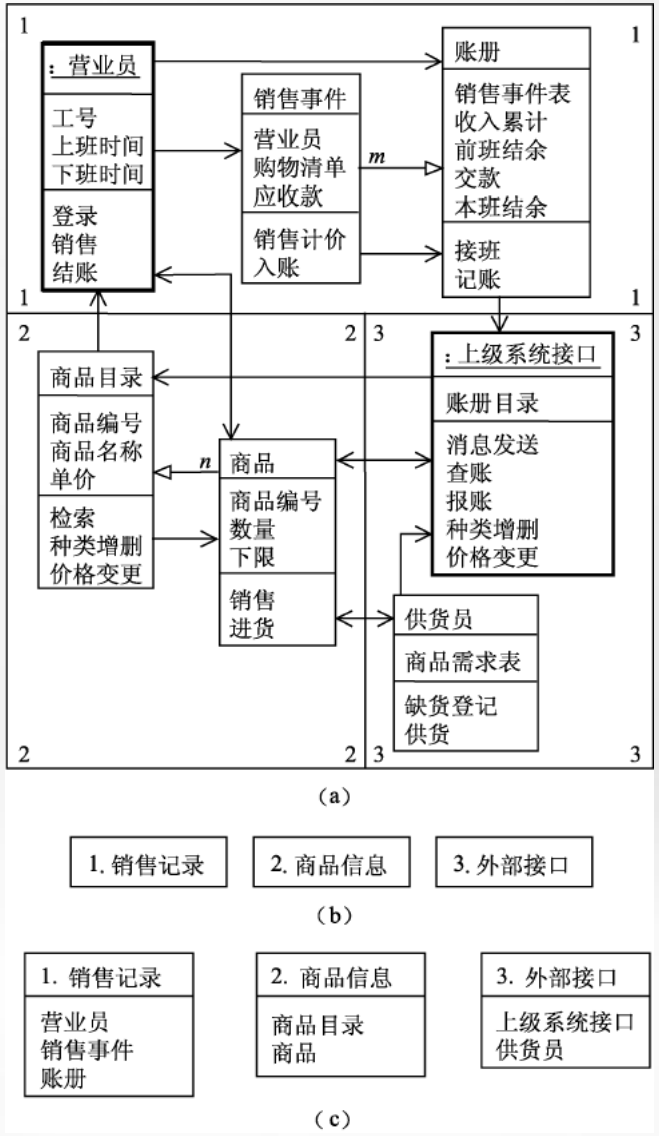
8.5.3 划分主题

03 主动对象

OPTION

主动对象（Active Object）的概念、作用和意义最近几年开始受到重视。按照通常理解，对象的每个服务是在消息的驱动下执行的操作，所有这样的对象都是被动对象（PassiveObject）。

主动对象的作用是描述问题域中具有主动行为的事物以及在系统设计时识别的任务，主动服务描述相应的任务所完成的操作，在系统实现阶段应被实现为一个能并发执行的、主动的程序单位，如进程或线程。



本章内容

8.6.1 编写脚本

脚本的原意是指表演戏剧、曲艺及摄制影视剧等所依据的本子，里面记载台词、故事情节等。在建立动态模型的过程中，脚本是系统执行某个功能的一系列事件。

脚本通常起始于一个系统外部的输入事件，结束于一个系统外部的输出事件，它可以包括发生在此期间的系统的所有内部事件。编写脚本的目的是保证不遗漏系统功能中重要的交互步骤，有助于确保整个交互过程的正确性和清晰性。

8.6.2 设计用户界面

大多数交互行为都可以分为应用逻辑和用户界面两部分。通常，系统分析员首先集中精力考虑系统的信息流和控制流，而不是首先考虑用户界面。

但是，用户界面的美观、方便、易学及效率，是用户使用系统时首先感受到的。用户界面的美观与否往往对用户是否喜欢、是否接受一个系统起很重要的作用。在分析阶段不能忽略用户界面的设计，应该快速建立用户界面原型，供用户试用与评价。

8.6.3 画顺序图或活动图

顺序图中，一条竖线代表应用领域中的一个类，每个事件用一条水平的箭头线表示，箭头方向从事件的发送对象指向接收对象，时间从上向下递增。

8.6.4 画状态图

如果对象的属性值不相同，对象的行为规则有所不同，则称对象处于不同的状态。由于对象在不同状态下呈现不同的行为方式，因此应分析对象的状态，才可正确地认识对象的行为，并定义它的服务。

本章内容

8.7 建立功能模型

01 OPTION 确定输入、输出值

数据流图中的输入、输出值是系统与外部之间进行交互的事件的参数。

8.7 建立功能模型

02 OPTION 画数据流图

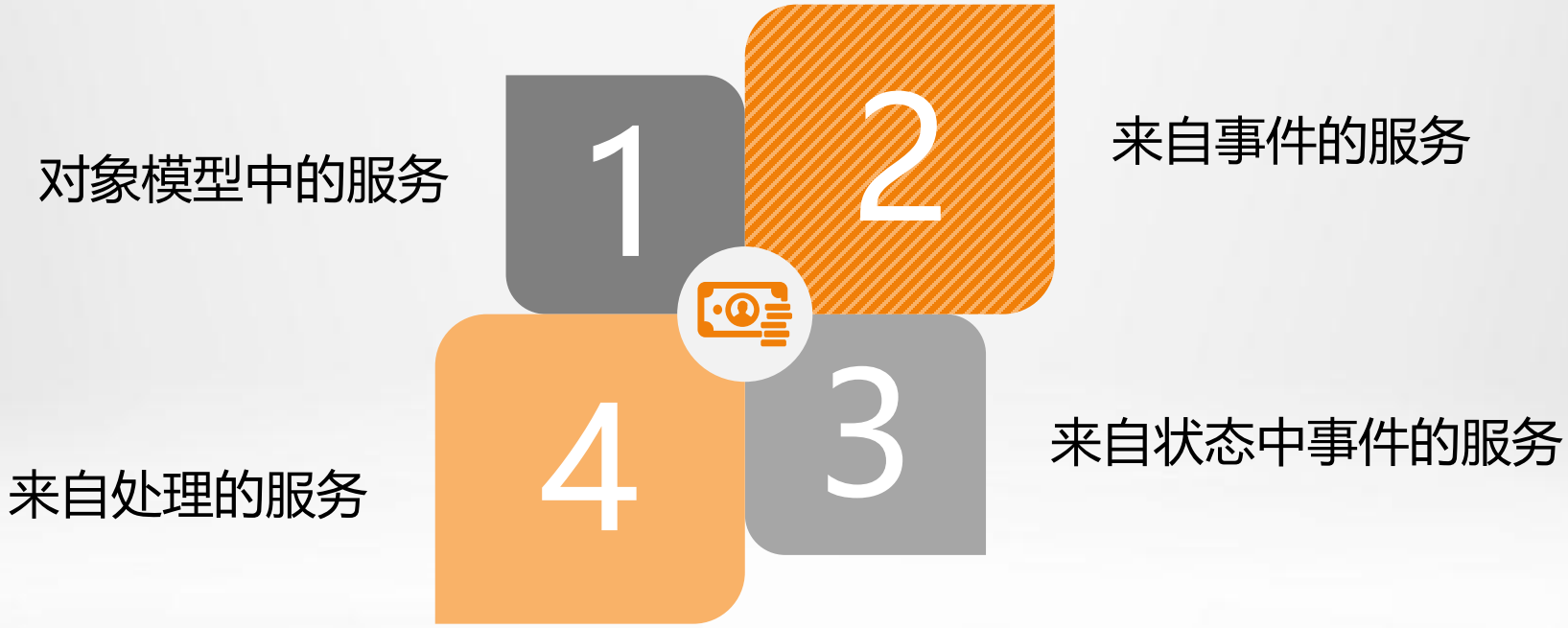
功能模型可用多张数据流图、程序流程图来表示。

数据流或处理流程	处理	数据存储	数据源或数据终点
用带箭头的直线表示	用圆角矩形或圆形表示	用两端被同方向的圆弧封口的平行线或一端用线段封口、一端开口的平行线表示	用矩形或长方体表示

8.7 建立功能模型

03 OPTION 定义服务

在建立对象模型时，确定了类、属性、关联、结构后，还要确定类的服务（操作）。在建立动态模型和功能模型后，类的服务才能最终确定。



本章内容

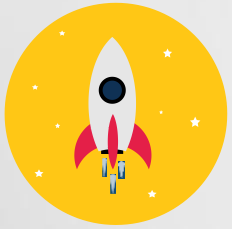
8.8.1 系统设计

系统设计确定实现系统的策略和目标系统的高层结构。系统设计要将系统分解为若干子系统，定义和设计子系统时应使其具有良好的接口，通过接口和系统的其余部分通信。

- 将系统分解为子系统，设计系统的拓扑结构
- 设计问题域子系统
- 设计人机交互子系统
- 设计任务管理子系统
- 设计数据管理子系统

8.8.2 对象设计

01 OPTION 对象描述



协议描述

协议描述是一组消息和对消息的注释。对于有很多消息的大型系统，可能要创建消息的类别。通过定义对象可以接收的每个消息和当对象接收到消息后完成的相关操作来建立对象的接口。

实现描述

实现描述由传送给对象的消息所蕴含的每个操作的实现细节组成，包括对象名字的定义和类的引用、关于描述对象属性的数据结构的定义及操作过程的细节。

8.8.2 对象设计

02 OPTION 设计类中的服务

1 确定类中应有的服务

- 需要综合考虑对象模型、动态模型和功能模型来确定类中应有的服务，如状态图中对象对事件的响应、数据流图中的处理、输入流对象、输出流对象及存储对象等。

2 设计实现服务的方法

- 设计实现服务应先设计实现服务的算法，考虑算法的复杂度，考虑如何使算法容易理解、容易实现并容易修改；
- 其次是选择数据结构，要选择能方便、有效地实现算法的数据结构；
- 最后是定义类的内部操作，可能需要添加一些用来存放中间结果的类。

8.8.2 对象设计

03 OPTION 设计类的关联

在应用系统中，使用关联有两种可能的方式：只需单向遍历的单向关联和需要双向遍历的双向关联。单向关联用简单指针来实现，双向关联用指针集合来实现。

8.8.2 对象设计

04 OPTION 链属性的实现

链属性的实现要根据具体情况分别处理，如果是一对一关联关系，链属性可作为其中一个对象的属性而存储在该对象中；如果是一对多关联关系，链属性可作为“多”端对象的一个属性。至于多对多关联关系，可使用一个独立的类来实现链属性。

8.8.2 对象设计

05 OPTION 设计的优化

设计的优化需要先确定优先级，设计人员必须确定各项质量指标的相对重要性，才能确定优先级，以便在优化设计时制定折中方案。通常在效率和设计清晰性之间寻求折中，有时可以增加冗余的关联关系以提高访问效率，或调整查询次序，或保留派生的属性等方法来优化设计。

8.8.3 面向对象设计的准则和启发式规则

01 OPTION 面向对象设计的准则

模块化

对象就是模块，把数据结构和操作数据的方法紧密地结合在一起，构成模块

抽象

类是一种抽象数据类型，对外开放的公共接口构成了类的规格说明（协议）

信息隐蔽

对于类的用户来说，属性的表示方法和操作的实现算法都应该是隐蔽的。

低耦合（弱耦合）

对象之间的耦合主要有交互耦合和继承耦合两种

高内聚（强内聚）

- 服务内聚
- 类内聚
- 一般- 特殊内聚

重用性

尽量使用已有的类；确实需要创建新类时，应考虑将来可重复使用

8.8.3 面向对象设计的准则和启发式规则

02 OPTION 面向对象设计的启发式规则

- 设计结果应该清晰、易懂。
- 一般- 特殊结构的深度应适当。
- 设计简单的类。
- 使用简单的协议。
- 使用简单的服务。
- 把设计变动减到最小。

本章内容

8.9.1 选择程序设计语言

01 OPTION 选择程序设计语言的关键因素

与面向对象分析和面向对象设计有一致的表示方法

具有可重用性

可维护性强

8.9.1 选择程序设计语言

02 OPTION 面向对象程序设计语言的技术特点

- 具有支持类与对象的概念的机制。
- 实现整体- 部分结构的机制。
- 实现一般- 特殊结构的机制。
- 实现属性和服务的机制。
- 类型检查的机制。
- 建立类库。
- 持久保存对象的机制。
- 将类参数化的机制。
- 运行效率。
- 开发环境。

8.9.1 选择程序设计语言

03 OPTION 选择面向对象程序设计语言的实际因素

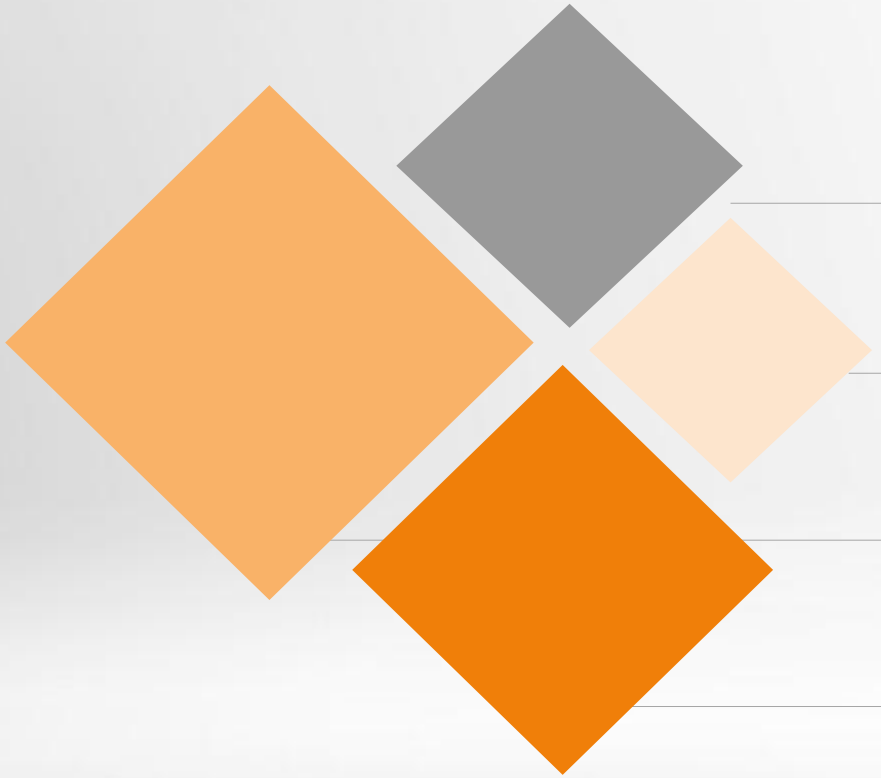
8.9.2 面向对象程序设计

01 OPTION 提高软件的可重用性

- 提高类的操作（服务）的内聚。类的一个操作应只完成单个功能，如果涉及多个功能，
- 应把它分解成几个更小的操作。
- 减小类的操作的规模。类的某个操作的规模如果太大，应把它分解成几个更小的操作。
- 保持操作的一致性：功能相似的操作应有一致的名字、参数特征、返回值类型、使用条件及出错条件等。
- 把提供决策的操作与完成具体任务的操作分开设计。
- 全面覆盖所有条件组合。
- 尽量不使用全局量。
- 利用继承机制。

8.9.2 面向对象程序设计

02 OPTION 提高软件的可扩充性



把类的实现封装起来

一个操作应只包含对象模型中的有限内容，不要包含多种关联的内容

避免使用多分支语句

精心确定公有的属性、服务或关联

8.9.2 面向对象程序设计

03
OPTION

提高软件的健壮性

预防用户的操作错误

检查参数的合法性

不预先设定数据结构的限制条件

本章内容

8.10.1 面向对象测试策略

传统的单元测试集中在最小的可编译程序单位中，即子程序（模块）中，一旦这些单元都测试完，就把它们集成到程序结构中，这时要进行一系列的回归测试，以发现模块的接口错误和新单元加入程序中所带来的副作用。最后，系统被作为一个整体来测试，以发现软件需求中的错误。

对象和类的认定

面向对象的单元测试

面向对象的集成测试

面向对象的确认测试

8.10.2 面向对象的测试步骤

1．制定测试计划

根据用例模型、分析模型、设计模型、实现模型、构架描述和补充需求来制定测试计划

2．设计测试用例

设计类的测试用例、设计集成测试用例、设计系统测试用例、设计回归测试用例

3．实现测试构件

依赖于测试自动化工具、软件工程师开发测试构件

4．集成测试

根据测试用例、测试规程、测试构件和实现模型执行集成测试

5．系统测试

根据测试用例、测试规程、测试构件和实现模型对所开发的系统进行测试

6．测试评估

根据测试计划、测试用例、测试规程、测试构件和测试执行者反馈的测试缺陷，对一系列的测试工作做出评估

本章内容

8.11.1 UML 模型

01 OPTION 用例模型

- 将用例按优先级分类：优先级高的、必须首先实现的功能最先开发。
- 区分用例在体系结构方面的风险大小：如果某个用例暂时不实现会导致以后的迭代渐增式开发时有大量的改写工作，这样的用例要先开发。
- 对用例所需的工作量进行估算，合理安排工作计划。

8.11.1 UML 模型

02 OPTION 静态模型

UML 对可重用性的支持，在设计的前期体现在支持可重复使用的类和结构上，后期则体现在组件的装配上。

静态模型主要描述类、接口、用例、组件、结点等体现系统结构的事物。静态模型使用的图包括用例图、类图、对象图、构件图和部署图等。

8.11.1 UML 模型

03 动态模型

OPTION

动态模型主要描述消息交互和状态机两种动作。

消息交互

对象之间为达到特定目的而进行的一系列消息交换，从而组成一系列动作，通过消息通信相互协作等，可用顺序图、活动图和协作图等描述。

状态机

状态机由对象的一系列状态和激发这些状态转换的事件组成，用状态图描述。

8.11.1 UML 模型

04 实现模型

OPTION

实现模型包括构件图和部署图，它们描述了系统实现时的一些特性。

构件图

显示代码本身的逻辑结构：构件图描述系统中的软件构件以及它们之间的相互依赖关系，构件图的元素有构件、依赖关系和接口。

部署图

显示系统运行时的结构：部署图显示系统硬件的拓扑结构和通信路径、系统结构结点上执行的软件构件所包含的逻辑单元等。

8.11.2 UML 视图

01 OPTION 结构分类

1 静态视图

由类图组成，主要概念为类、关联、继承、依赖关系、实现和接口。

2 用例视图

由用例图组成，主要概念为用例、执行者、关联、扩展，包括用例继承。

3 实现视图

由构件图组成，主要概念为构件、接口、依赖关系和实现。

8.11.2 UML 视图

02 OPTION 动态行为

部署视图：由部署图组成，主要概念为结点、构件、依赖关系和位置

状态视图：由状态图组成，主要概念为状态、事件、转换和动作

活动视图：由活动图组成，主要概念为状态、活动、转换、分叉和结合

交互视图：由顺序图、协作图组成，主要概念为交互、对象、消息、激活、协作及角色

8.11.2 UML 视图

03 OPTION 模型管理

模型管理说明了模型的分层组织结构。模型管理根据系统开发和部署来组织视图，主要概念为包、子系统和模型。

UML 的所有视图和所有图都具有可扩展性，扩展机制用约束、版型和标签来实现。

8.11.3 UML 使用准则

UML 可以有多种模型、多种视图，都用图来描述。UML 共有用例图、类图、对象图、状态图、顺序图、活动图、协作图、构件图和部署图9 种图。下面介绍UML 的一些使用准则。

8.11.4 UML 的应用领域

UML 是一种通用的标准建模语言，可以为任何具有静态结构和动态行为的系统建立模型。UML 适用于系统开发的全过程，应用于需求分析、设计、编码和测试等所有阶段。