

# UML 基础教程

## 1. 前言

- 1.1前言
- 1.2UML概述
- 1.3UML事物
- 1.4UML关系
- 1.5各UML图及特征
- 1.6各UML图的关系
- 1.7UML语法
- 1.8习题

## 2. 用例图

- 2.1用例图概要
- 2.2用例图中的事物及解释
- 2.3用例图中的关系及解释
- 2.4例子
- 2.5习题

## 3. 类图

- 3.1类图概要
- 3.2类图中的事物及解释
- 3.3类图中的关系及解释
- 3.4类图与代码的映射
- 3.5类图例子
- 3.6习题

## 4. 顺序图

- 4.1概要
- 4.2顺序图中的事物及解释
- 4.3顺序图与用例图和类图的关系
- 4.4顺序图例子
- 4.5 练习题

## 5. 协作图

- 5.1概要
- 5.2协作图中的事物及解释
- 5.3协作图中的关系及解释

5.4对消息标签的详细讲解

5.5协作图例子

5.6协作图与顺序图的区别和联系

5.7练习题

## 6. 状态图

6.1状态图概要

6.2状态图的组成

6.3状态图中的事物及解释

6.4状态的可选活动表

6.5简单的例子:对象的状态图

6.6复杂的例子:网上银行登陆系统

6.7练习

## 7. 活动图

7.1活动图概要

7.2活动图事物

7.3活动图关系

7.4活动图实例

7.5活动图练习

## 8. 构件图

8.1构件图概要

8.2构件图中的事物及解释

8.3构件图中的关系及解释

8.4构件图的例子

8.5习题

## 9. 部署图

9.1部署图概要

9.2部署图中的事物及解释

9.3部署图中的关系及解释

9.4部署图的例子

9.5关于部署图与构件图

9.6习题

附录

# 1. 前言

## 1.1 前言

本资料对UML1.5各种模型图的构成和功能进行说明，通过本资料的学习达到可以读懂UML模型图的目的。本资料不涉及模型图作成的要点等相关知识。

## 1.2 UML概述

### 1.2.1 UML简介

UML (Unified Modeling Language)为面向对象软件设计提供统一的、标准的、可视化的建模语言。适用于描述以用例为驱动，以体系结构为中心的软件设计的全过程。

UML的定义包括UML语义和UML表示法两个部分。

**(1) UML语义：**UML对语义的描述使开发者能在语义上取得一致认识，消除了因人而异的表达方法所造成的影响。

**(2) UML表示法：**UML表示法定义UML符号的表示法，为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。

### 1.2.2 UML模型图的构成

**事物(Things)：**UML模型中最基本的构成元素，是具有代表性的成分的抽象

**关系(Relationships)：**关系把事物紧密联系在一起

**图(Diagrams )：**图是事物和关系的可视化表示

# 1. 前言

## 1.3 UML事物

UML包含4种事物：构件事物 行为事物 分组事物 注释事物

### 1.3.1 构件事物： UML模型的静态部分，描述概念或物理元素

它包括以下几种：

**类**：具有相同属性相同操作 相同关系相同语义的对象的描述

**接口**：描述元素的外部可见行为，即服务集合的定义说明

**协作**：描述了一组事物间的相互作用的集合

**用例**：代表一个系统或系统的一部分行为，是一组动作序列的集合

**构件**：系统中物理存在，可替换的部件

**节点**：运行时存在的物理元素

另外，参与者、信号应用、文档库、页表等都是上述基本事物的变体

### 1.3.2 行为事物： UML模型图的动态部分，描述跨越空间和时间的行为

**交互**：实现某功能的一组构件事物之间的消息的集合，涉及消息、动作序列、链接

**状态机**：描述事物或交互在生命周期内响应事件所经历的状态序列

### 1.3.3 分组事物： UML模型图的组织部分，描述事物的组织结构

**包**：把元素组织成组的机制

### 1.3.4 注释事物： UML模型的解释部分，用来对模型中的元素进行说明，解释

**注解**：对元素进行约束或解释的简单符号

# 1. 前言

## 1.4 UML关系

### 1.4.1 依赖

依赖(dependency)是两个事物之间的语义关系，其中一个事物(独立事物)发生变化，会影响到另一个事物(依赖事物)的语义

### 1.4.2 关联

关联(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间的联系

### 1.4.3 泛化

泛化(generalization)是一种特殊/一般的关系。也可以看作是常说的继承关系

### 1.4.4 实现

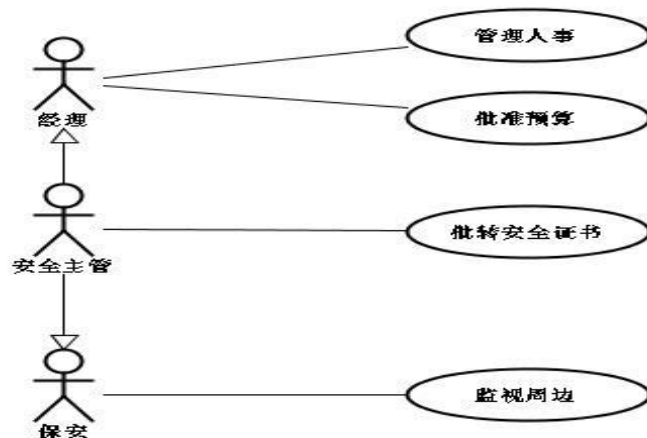
实现(realization)是类元之间的语义关系，其中的一个类元指定了由另一个类元保证执行的契约

# 1. 前言

## 1.5 各UML图及特征

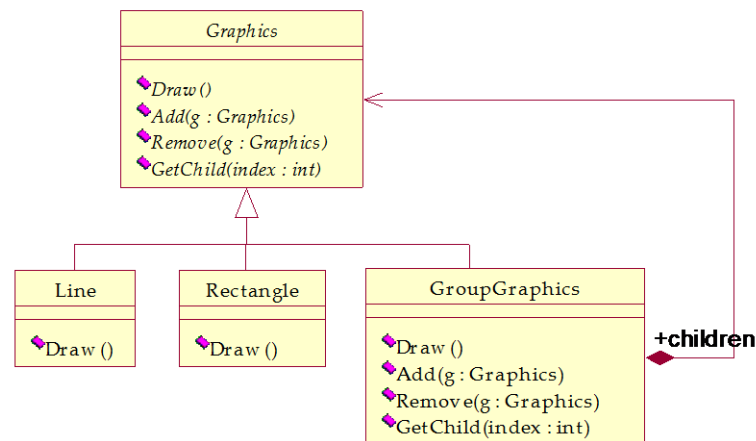
### 1.5.1 用例图( Use Case Diagram )

- ※ 用例图是从用户角度描述系统功能，是用户所能观察到的系统功能的模型图，用例是系统中的一个功能单元



### 1.5.2 类图(Class Diagram)

- ※ 类图描述系统中类的静态结构。不仅定义系统中的类，表示类之间的联系如关联、依赖、聚合等，也包括类的内部结构(类的属性和操作)
- ※ 类图是以类为中心来组织的，类图中的其他元素或属于某个类或与类相关联

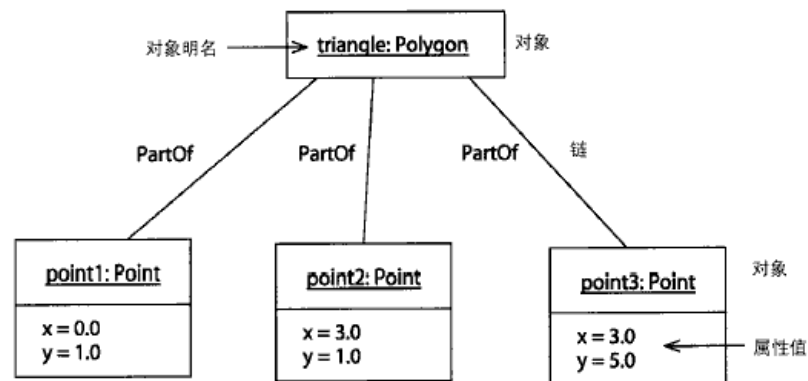


# 1. 前言

## 1.5 各UML图及特征

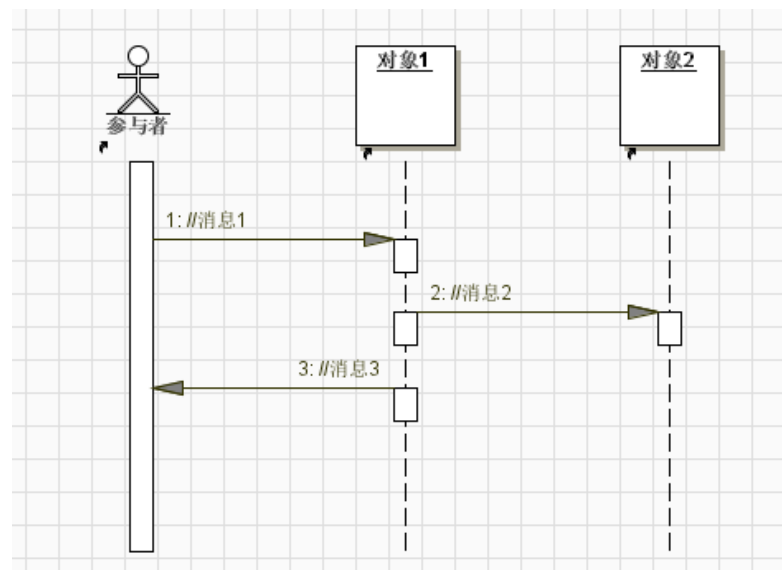
### 1.5.3 对象图( Object Diagram )

- ※ 对象图是类图的实例，几乎使用与类图完全相同的标识。他们的不同点在于对象图显示类的多个对象实例，而不是实际的类



### 1.5.4 顺序图(Sequence Diagram)

- ※ 顺序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互
- ※ 顺序图的一个用途是用来表示用例中的行为顺序。当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或引起状态转换的触发事件



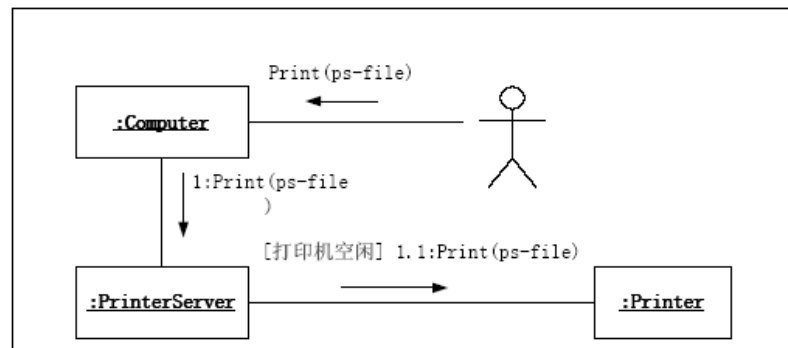


# 1. 前言

## 1.5 各UML图及特征

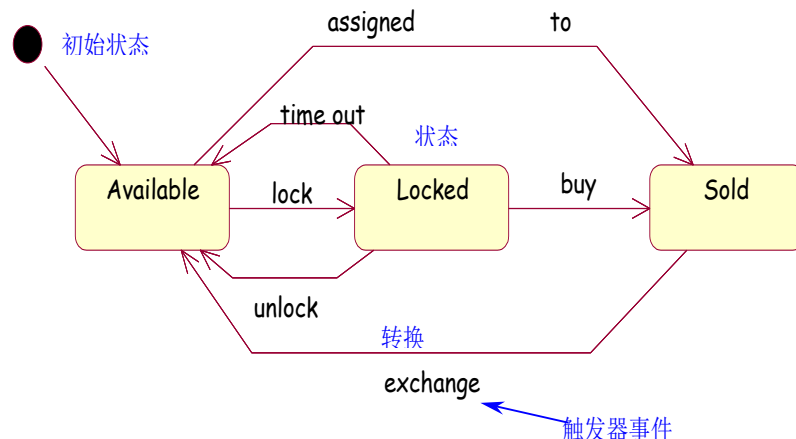
### 1.5.5 协作图(Collaboration Diagram)

- ※ 协作图描述对象间的协作关系，协作图跟顺序图相似，显示对象间的动态合作关系。除显示信息交换外，协作图还显示对象以及它们之间的关系。
- ※ 协作图的一个用途是表示一个类操作的实现



### 1.5.6 状态图(State Chart Diagram)

- ※ 状态图是一个类对象所可能经历的所有历程的模型图。状态图由对象的各个状态和连接这些状态的转换组成

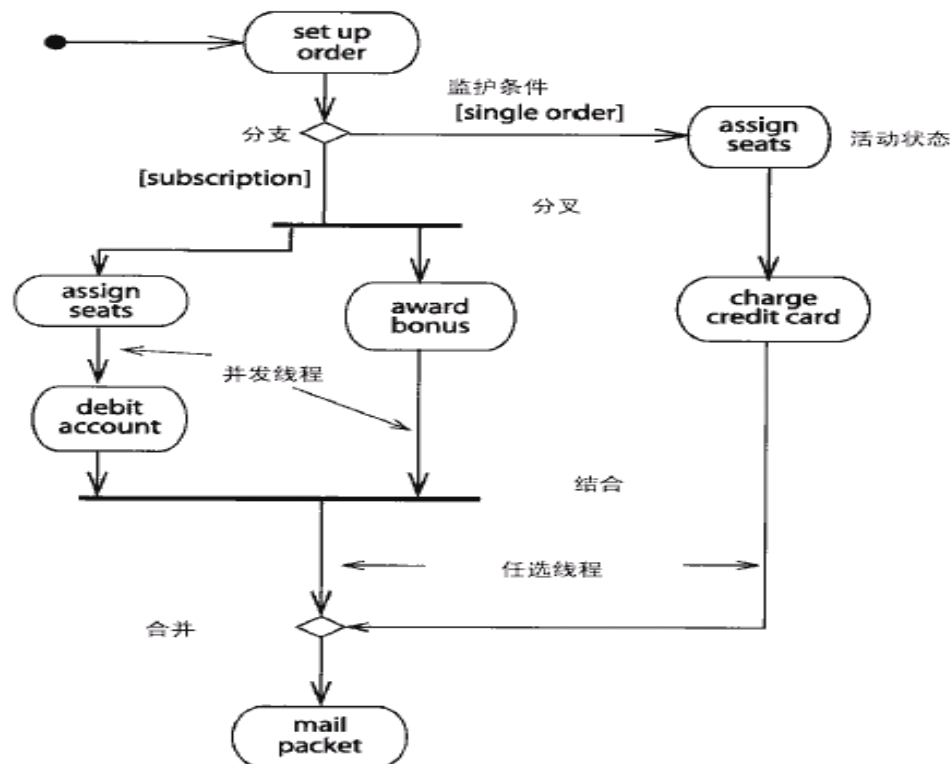


# 1. 前言

## 1.5 各UML图及特征

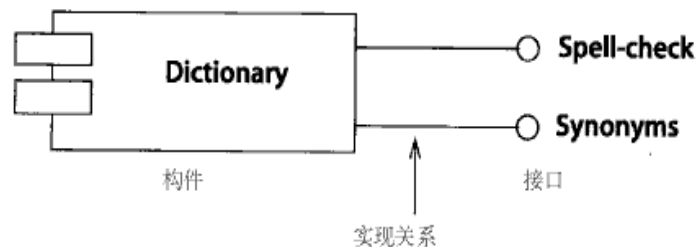
### 1.5.7 活动图(Activity Diagram)

- ※ 活动图是状态图的一个**变体**，用来描述**执行算法的工作流程**中涉及的活动
- ※ 活动图描述了一组**顺序的或并发的**活动



### 1.5.8 构件图(Component Diagram)

- ※ 构件图为系统的构件建模—构件即构造应用的软件单元—还包括各构件之间的依赖关系，以便通过这些依赖关系来估计对系统构件的修改给系统可能带来的影响

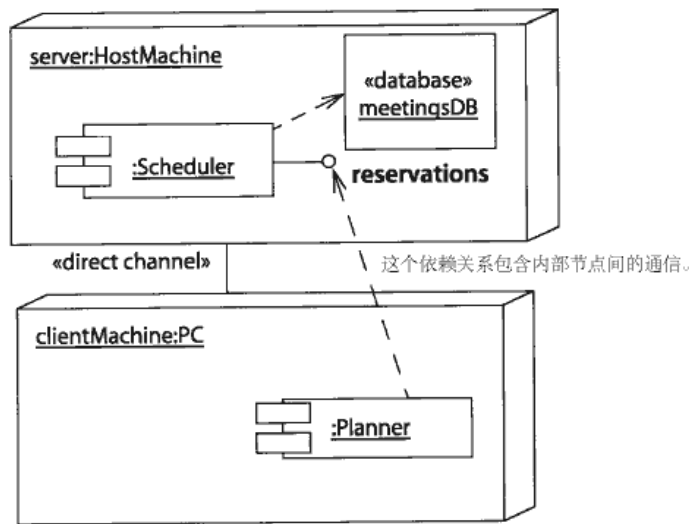


# 1. 前言

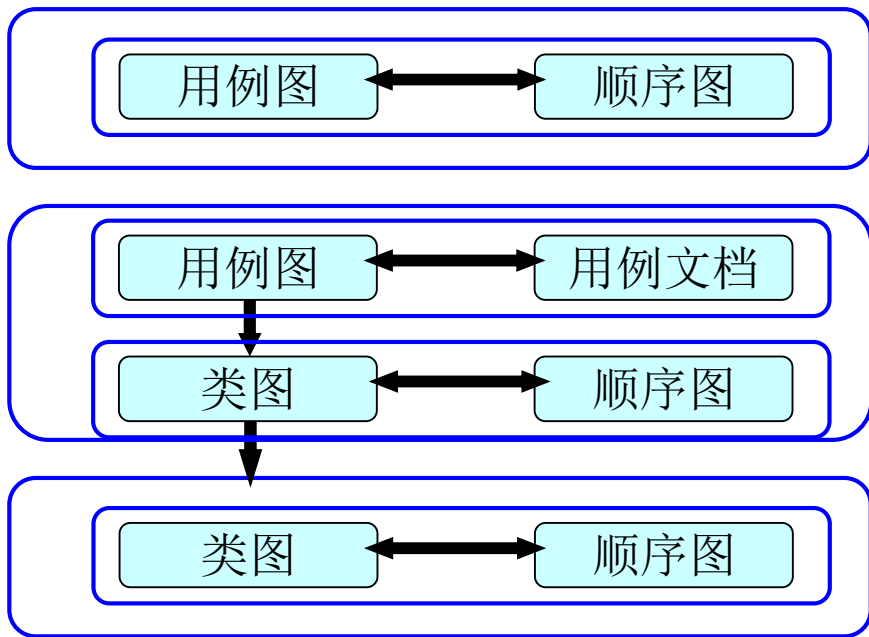
## 1.5 各UML图及特征

### 1.5.9 部署图(Deployment Diagram)

部署视图描述位于节点实例上的运行构件实例的安排。节点是一组运行资源，如计算机、设备或存储器。这个视图允许评估分配结果和资源分配



## 1.6 各UML图的关系



需求分析

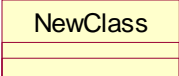
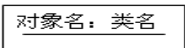
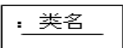




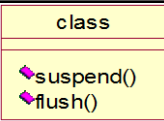
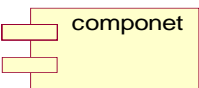

BD Base Design 基本设计



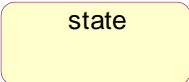
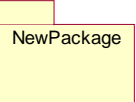




FD Functional Design 功能设计  
DD Detailed Design 详细设计

主要图之间的关系

# 1. 前言

## 1.7 UML语法描述

类	是对一组具有相同属性、相同操作、相同关系和相同语义的对象的描述	
对象	  	
接口	是描述了一个类或构件的一个服务的操作集	
协作	定义了一个交互，它是由一组共同工作以提供某种协作行为的角色和其他元素构成的一个群体	
用例	是对一组动作序列的描述	
主动类	对象至少拥有一个进程或线程的类	
构件	是系统中物理的、可替代的部件	
参与者	在系统外部与系统直接交互的人或事物	

节点	是在运行时存在的物理元素	
交互	它由在特定语境中共同完成一定任务的一组对象间交换的消息组成	
状态机	它描述了一个对象或一个交互在生命期内响应事件所经历的状态序列	
包	把元素组织成组的机制	
注释事物	是UML模型的解释部分	
依赖	一条可能有方向的虚线	
关联	一条实线，可能有方向	
泛化	一条带有空心箭头的实线	
实现	一条带有空心箭头的虚线	

# 1. 前言

## 1.8习题

### 判断题

- 1、UML中一共有九种图：它们是用例图、类图、对象图、顺序图、协作图、状态图、活动图、构件图、部署图 **OK**
- 2、用例图是从程序员角度来描述系统的功能 **NO**
- 3、类图是描述系统中类的静态结构，对象图是描述系统中类的动态结构 **NO**
- 4、活动图 and 状态图用来描述系统的动态行为 **OK**
- 5、协作图的一个用途是表示一个类操作的实现 **OK**

### 选择题

- 6、请在下面选项目中选出两种可以互相转换的图 **ab**  
(a) 顺序图 (b) 协作图 (c) 活动图 (d) 状态图 **解释：协作图与顺序图类似；活动图是状态图的一个变体**
- 7、下面哪些图可用于BD阶段 **acd**  
(a) 用例图 (b) 构件图 (c) 类图 (d) 顺序图



答案：1.正确 2.错误 3.错误 4.正确 5.正确 6. (a)(b) 7.(a)(c)(d)

# 2. 用例图

## 2.1 用例图概要





- ◇ 用例图是被称为参与者的外部用户所能观察到的系统功能的模型图。(《UML参考手册》)
- ◇ 用例图列出系统中的用例和系统外的参与者，并显示哪个参与者参与了哪个用例的执行(或称为发起了哪个用例)。
- ◇ 用例图多用于静态建模阶段(主要是业务建模和需求建模)。

## 2.2 用例图中的事物及解释

事物名称	解释	UML表示
参与者(Actor)	<p>在系统外部与系统直接交互的人或事物(如另一个计算机系统或一些可运行的进程)。我们需要注意的:</p> <ol style="list-style-type: none"><li>1.参与者是角色(role)而不是具体的人,它代表了参与者在与系统打交道的过程中所扮演的角色。所以在系统的实际运作中,一个实际用户可能对应系统的多个参与者。不同的用户也可以只对应于一个参与者,从而代表同一参与者的不同实例。</li><li>2.参与者作为外部用户(而不是内部)与系统发生交互作用,是它的主要特征。</li><li>3.在后面的顺序图等中出现的“参与者”,与此概念相同,但具体指代的含义,视具体情况而定。</li></ol>	
用例(Use Case)	<p>系统外部可见的一个系统功能单元。系统的功能由系统单元所提供,并通过一系列系统单元与一个或多个参与者之间交换的消息所表达。创建新用例,确认候选用例和划分用例范围的优秀法则----“WAVE”测试(见附录)</p>	

## 2. 用例图

### 2.3 用例图中的关系及解释

关系		解释	图
参与者与用例之间的关系	关联	表示参与者与用例之间的交互，通信途径。 (关联有时候也用带箭头的实线来表示，这样的表示能够显示地表明发起用例的是参与者。)	
用例之间的关系	包含	箭头出发的用例为基用例； 箭头指向的用例为被包含的用例，称为包含用例； 包含用例是 <b>必选</b> 的，如果缺少包含用例，基用例就不完整；包含用例 <b>必须</b> 被执行，不需要满足某种条件； 其执行并 <b>不会</b> 改变基用例的行为。	《include》 
	扩展	箭头出发的用例为基用例； 箭头指向的用例为被扩展的用例，称为扩展用例； 扩展用例是 <b>可选</b> 的，如果缺少扩展用例，不会影响到基用例的完整性；扩展用例在 <b>一定条件下</b> 才会执行，并且其执行 <b>会</b> 改变基用例的行为。	《extend》 
参与者之间的关系	泛化	发出箭头的事物“ <b>is a</b> ”箭头指向的事物。泛化关系是一般和特殊关系，发出箭头的一方代表特殊的一方，箭头指向的一方代表一般一方。特殊一方继承了一般方的特性并增加了新的特性。	

## 2. 用例图

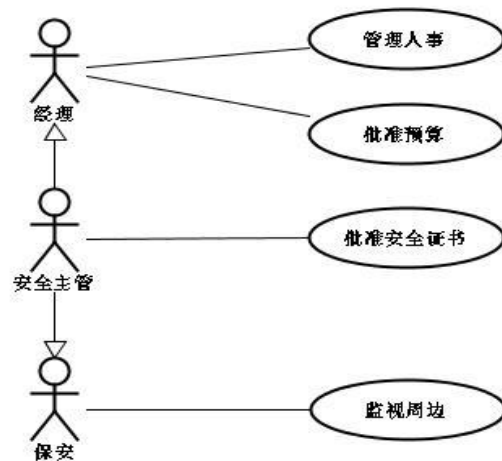
### 2.4 例子

#### 实例1 参与者之间的泛化关系

**参与者：**经理，安全主管，保安

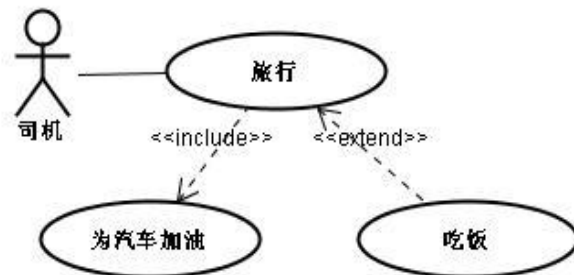
**用例：**管理人事，批准预算，批准安全证书，监视周边

在参与者之间不存在泛化关系的情况下，各个参与者参与用例的情况分别是：经理参与用例管理人事和批准预算；安全主管参与用例批准安全证书；保安参与用例监视周边。由于安全主管与经理，安全主管与保安之间泛化关系的存在，意味着安全主管可以担任经理和保安的角色，就能够参与经理和保安参与的用例。这样，安全主管就可以参与全部4个用例。但经理或者保安却不能担任安全主管的角色，也就不能参与用例批准安全证书。



#### 实例2 用例之间扩展和包含关系

用例的上下文是：短途旅行但汽车的油不足以应付全部路程。那么为汽车加油的动作在旅行的每个场景(事件流)中都会出现，不加油就不会完成旅行。吃饭则可以由司机决定是否进行，不吃饭不会影响旅行的完成。

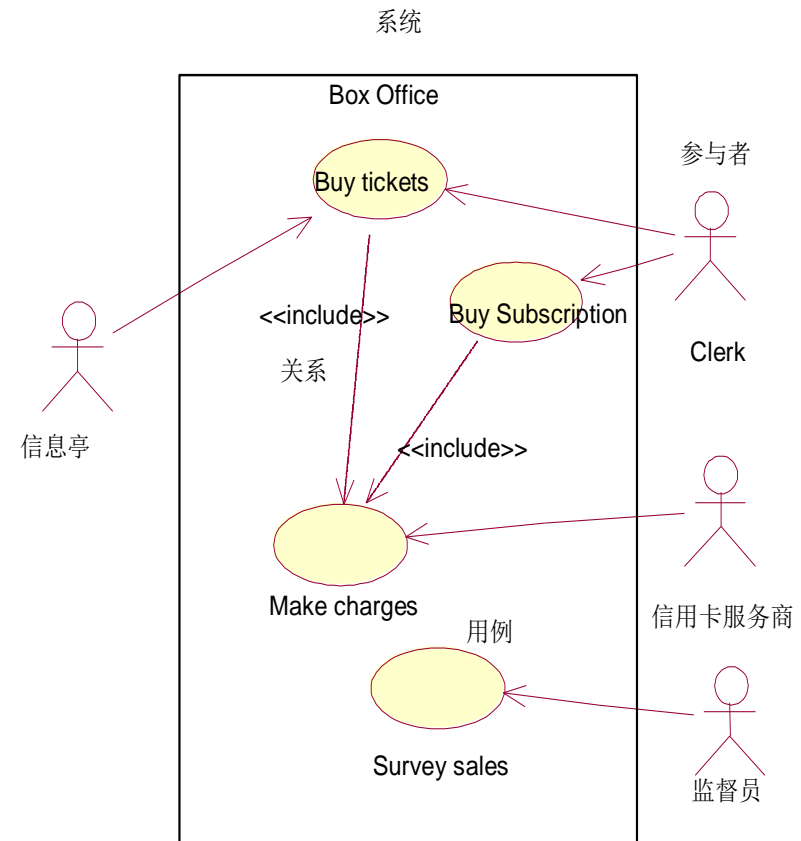




## 2. 用例图

### 实例3. 航空售票的用例图

- ✧ 参与者(actor): clerk, 监督员, 信用卡服务商, 信息亭
- ✧ 用例(use case): Buy tickets, Buy Subscription, Make charges, Survey sales
- ✧ 参与者Clerk参与(或称发起)Buy tickets和Buy Subscription两个用例(关联关系)。这两个用例的事件流都包含Make charges用例(包含关系)。
- ✧ 系统由: Buy tickets, Buy Subscription, Make charges, Survey sales组成。
- ✧ 该系统主要包含: Buy tickets, Buy Subscription, Make charges, Survey sales这几个功能。
- ✧ 该系统主要面向的用户(参与者): clerk, 监督员, 信用卡服务商, 信息亭。



# 2. 用例图

## 2.5 习题

1. 右图中的参与者有？ **ad**

- (a) 1
- (b) 2
- (c) 3
- (d) 4

2. 右图中的用例有？ **bc**

- (a) 1
- (b) 2
- (c) 3
- (d) 4

3. 2和3之间是什么关系？ 5和6呢？ **b**

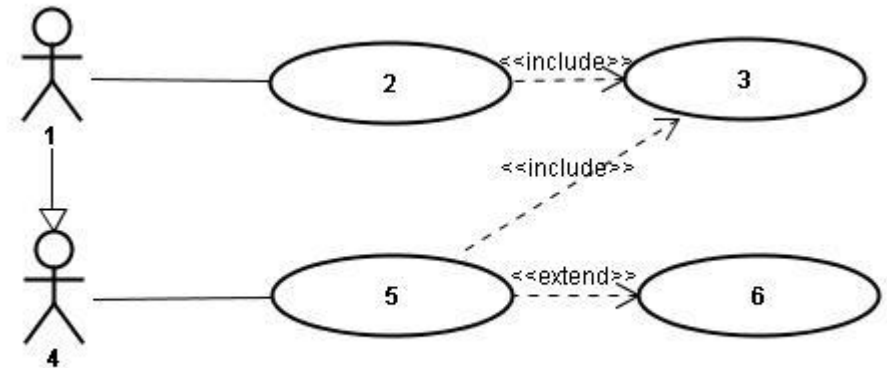
- (a) 扩展，包含
- (b) 包含，扩展

4. 5缺少了3仍然是个完整的用例？ **b**

- (a) 是的
- (b) 不是

5. 4能够参与2吗？ 1能够参与5吗？ **b**

- (a) 可以，不可以
- (b) 不可以，可以



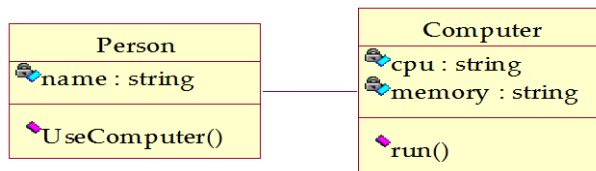
习题答案：

1、(a)(d) 2、(b)(c) 3、(b) 4、(b) 5、(b)

# 3. 类图

## 3.1 类图概要

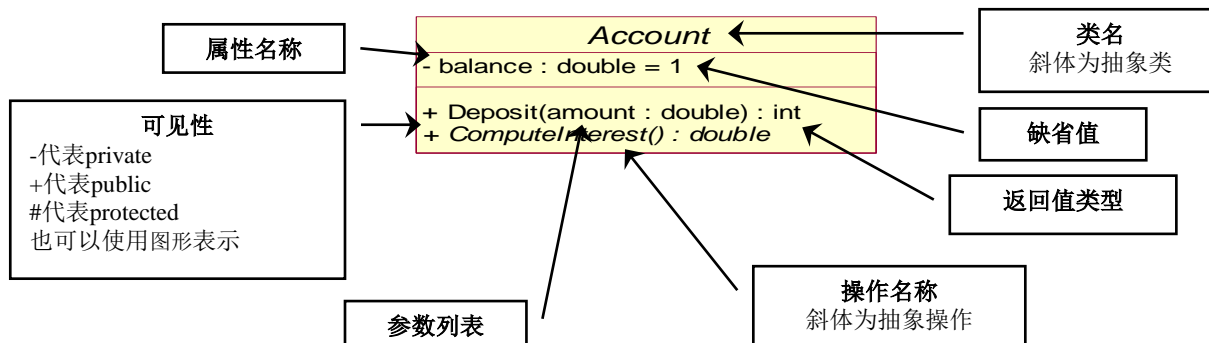
- ※ 类图以反映类的结构(属性、操作)以及类之间的关系为主要目的, 描述了软件系统的结构, 是一种静态建模方法
- ※ 类图中的“类”与面向对象语言中的“类”的概念是对应的, 是对现实世界中的事物的抽象



## 3.2 类图中的事物及解释

### 3.2.1 类

- ※ 从上到下分为三部分, 分别是类名、属性和操作。类名是必须有的
- ※ 类如果有属性, 则每一个属性都必须有一个名字, 另外还可以有其它的描述信息, 如可见性、数据类型、缺省值等
- ※ 类如果有操作, 则每一个操作也都有一个名字, 其它可选的信息包括可见性、参数的名字、参数类型、参数缺省值和操作的返回值的类型等



# 3. 类图

## 3.2 类图中的事物及解释

### 3.2.2 接口

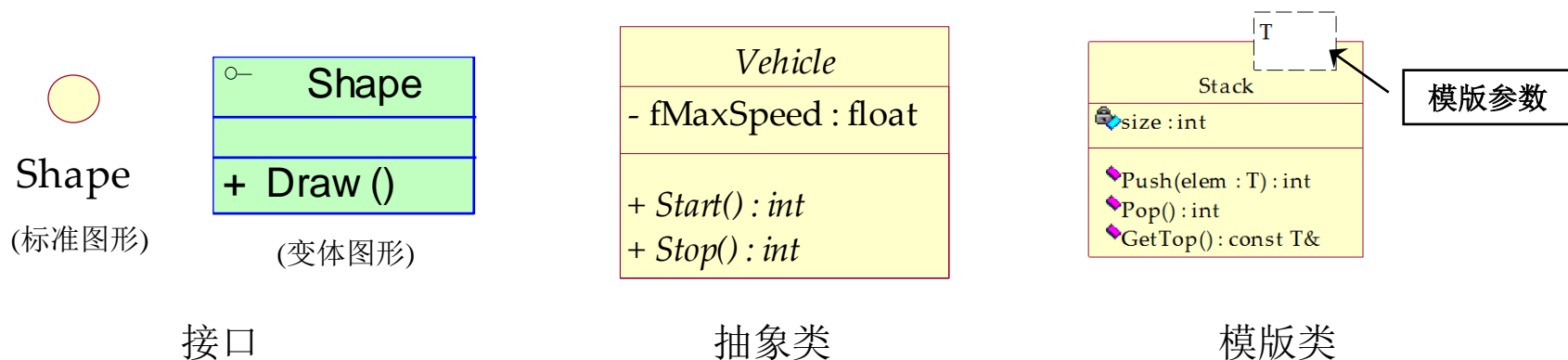
※ 一组操作的集合，只有操作的声明而没有实现

### 3.2.3 抽象类

※ 不能被实例化的类，一般至少包含一个抽象操作

### 3.2.4 模版类

※ 一种参数化的类，在编译时把模版参数绑定到不同的数据类型，从而产生不同的类



# 3. 类图

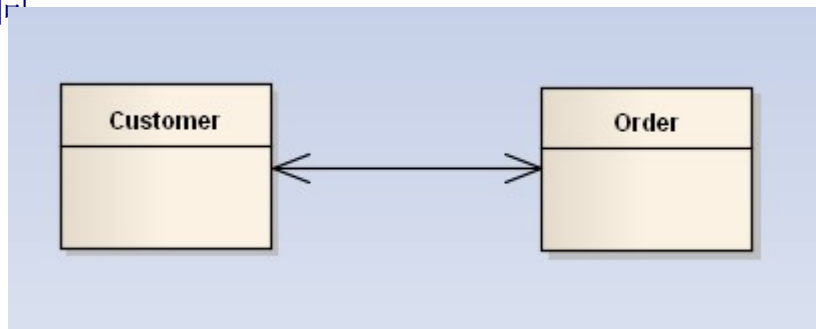
## 3.3 类图中的关系及解释

### 3.3.1 关联关系

- ※ 1、关联(Association): 对象之间一种引用关系, 比如客户类与订单类之间的关系。这种关系通常使用类的属性表达。关联又分为一般关联、聚合关联与组合关联。后两种在后面分析。在类图使用带箭头的实线表示, 箭头从使用类指向被关联的类。可以是单向和双向。



UML表示法

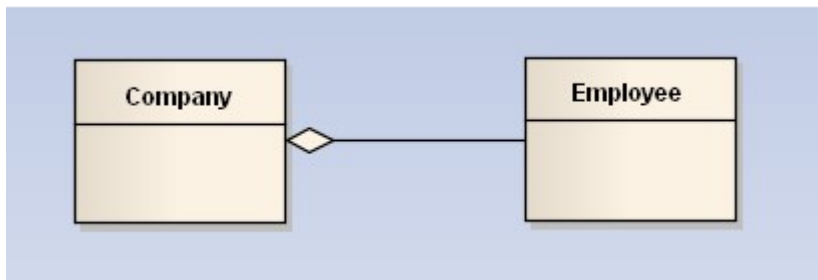


### 2、聚合关系

- 聚合(Aggregation): 表示has-a的关系, 是一种不稳定的包含关系。较强于一般关联, 有整体与局部的关系, 并且没有了整体, 局部也可单独存在。如公司和员工的关系, 公司包含员工, 但如果公司倒闭, 员工依然可以换公司。在类图使用空心的菱形表示, 菱形从局部指向整体。

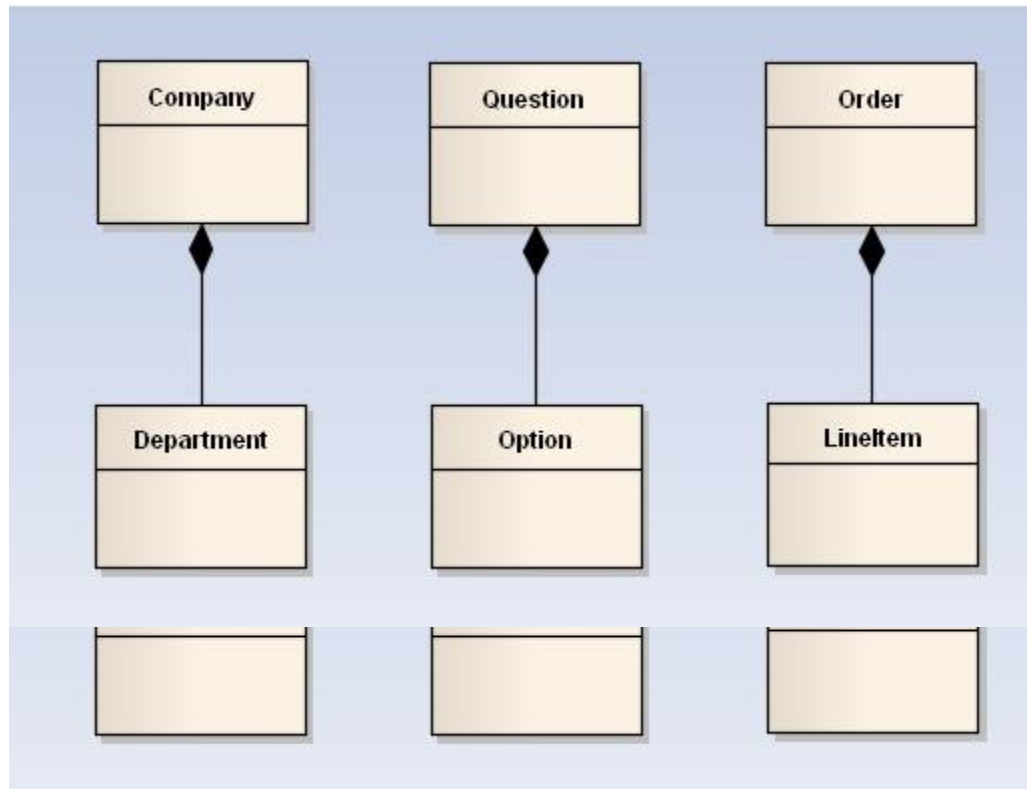


UML表示法



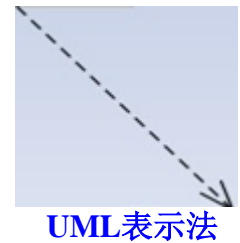
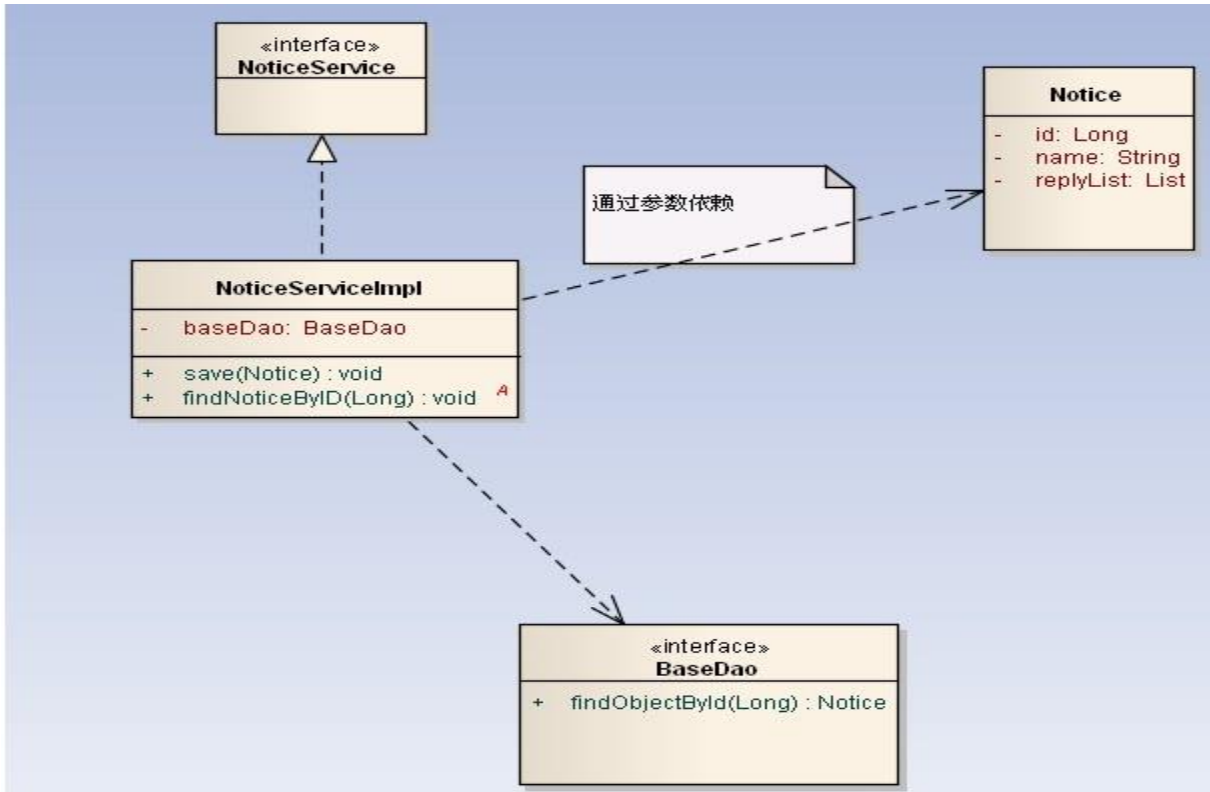
### 3、组合关系

- 组合(Composition): 表示contains-a的关系，是一种强烈的包含关系。组合类负责被组合类的生命周期。是一种更强的聚合关系。部分不能脱离整体存在。如公司和部门的关系，没有了公司，部门也不能存在了；调查问卷中问题和选项的关系；订单和订单选项的关系。在类图使用实心的菱形表示，菱形从局部指向整体。



### 3.3.2 依赖关系

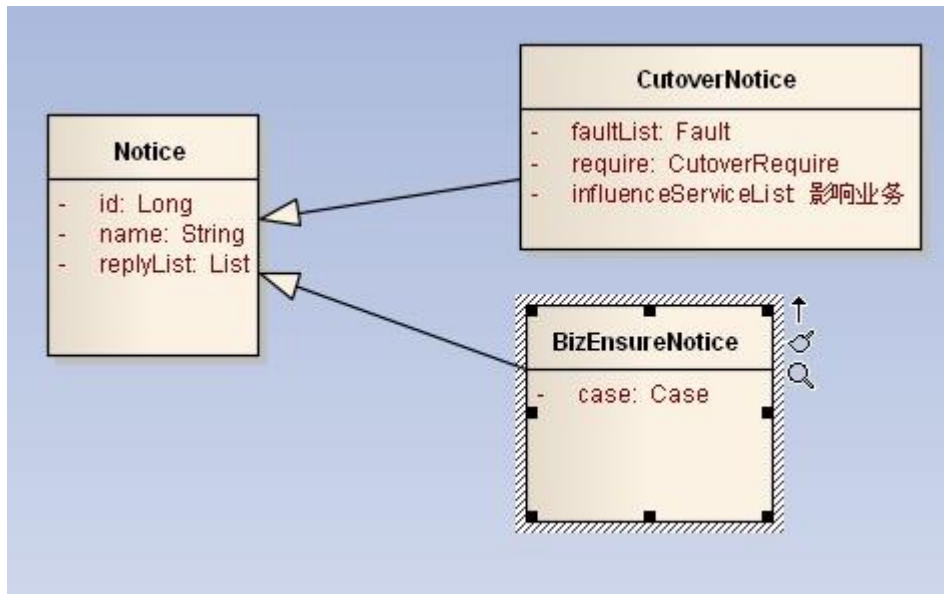
依赖 (Dependency)：对象之间最弱的一种关联方式，是临时性的关联。代码中一般指由局部变量、函数参数、返回值建立的对于其他对象的调用关系。一个类调用被依赖类中的某些方法而得以完成这个类的一些职责。在类图使用带箭头的虚线表示，箭头从使用类指向被依赖的类。



# 3. 类图

## 3.3.3 泛化关系

※ 泛化(generalization): 表示is-a的关系, 是对象之间耦合度最大的一种关系, 子类继承父类的所有细节。直接使用语言中的继承表达。在类图中使用带三角箭头的实线表示, 箭头从子类指向父类。

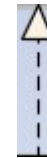
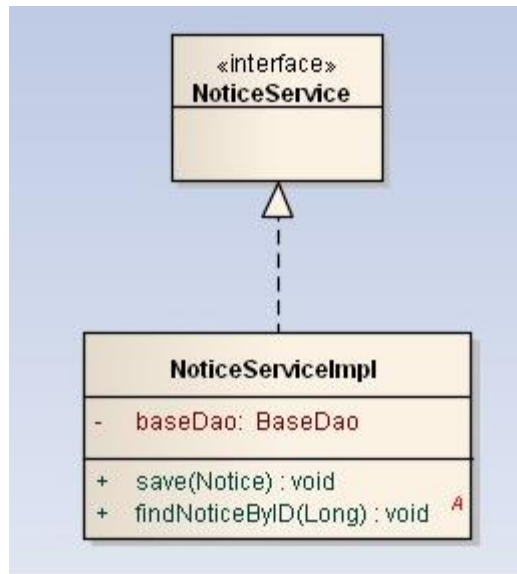


UML表示法



### 3.3.4 实现关系

实现（Realization）：在类图中就是接口和实现的关系。在类图中使用带三角箭头的虚线表示，箭头从实现类指向接口。

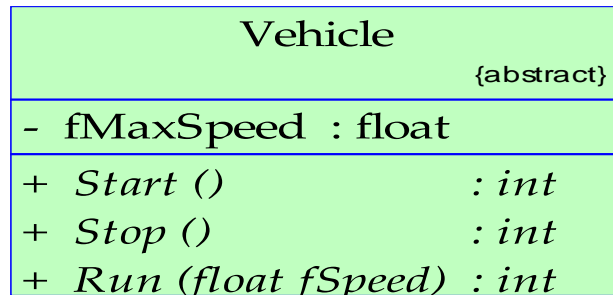


UML表示法

# 3. 类图

## 3.4 类图与代码的映射

### 3.4.1 类的映射



C++代码

```
class Vehicle
{
public:
    virtual int Start() = 0;
    virtual int Stop() = 0;
    virtual int Run(float
fSpeed) = 0;
private:
    float fMaxSpeed;
};
```



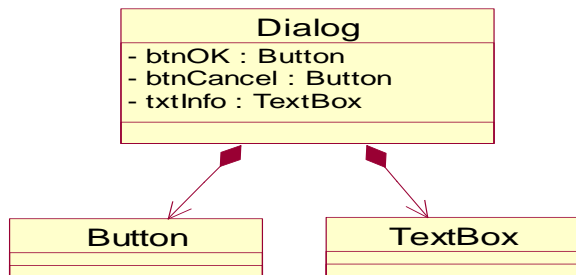
Java代码

```
public abstract class Vehicle
{
    public abstract int Start();
    public abstract int Stop();
    public abstract int Run(float
fSpeed);

    private float fMaxSpeed;
}
```

# 3. 类图

## 3.4.2 关联关系的映射



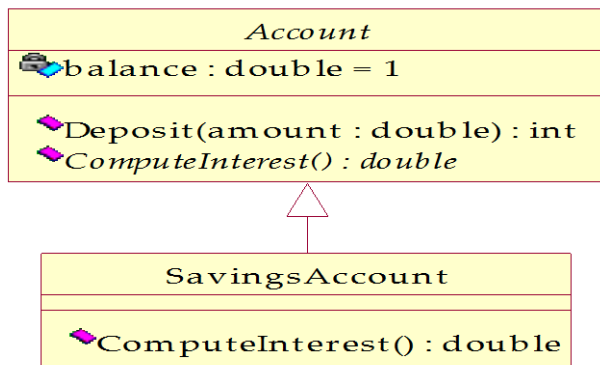
组合关系，代码表现为Dialog的属性有Button和TextBox的对象



C++代码

```
class Dialog
{
private:
    Button btnOK;
    Button btnCancel;
    TextBox txtInfo;
};
class Button
{};
class TextBox
{};
```

## 3.4.3 泛化关系的映射



C++代码

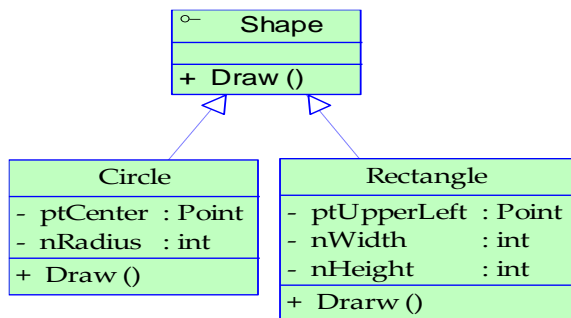
```
class SavingsAccount : public Account
{ };
```

Java代码

```
public class SavingsAccount extends Account
{ }
```

# 3. 类图

## 3.4.4 实现关系的映射



在C++语言里面，使用抽象类代替接口，  
使用泛化关系代替实现关系  
在Java语言里面，有相应的关键字  
interface、implements



C++代码

```
class Shape
{
public:
    virtual void Draw() = 0;
};

class Circle : public Shape
{
public:
    void Draw();
private:
    Point ptCenter;
    int nRadius;
};
```

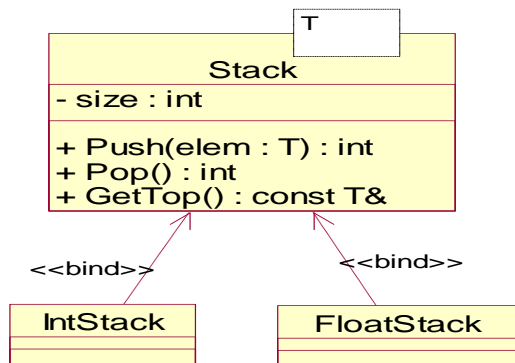
Java代码

```
public interface Shape
{
    public abstract void Draw();
}

public class Circle implements Shape
{
    public void Draw();

    private Point ptCenter;
    private int nRadius;
}
```

## 3.4.5 依赖关系的映射



绑定依赖



C++代码

```
template<typename T>
class Stack
{
private:
    int size;
public:
    int Push(T elem);
    int Pop();
    const T& GetTop();
};

typedef Stack<float> FloatStack;
```

C++代码(编译器生成)

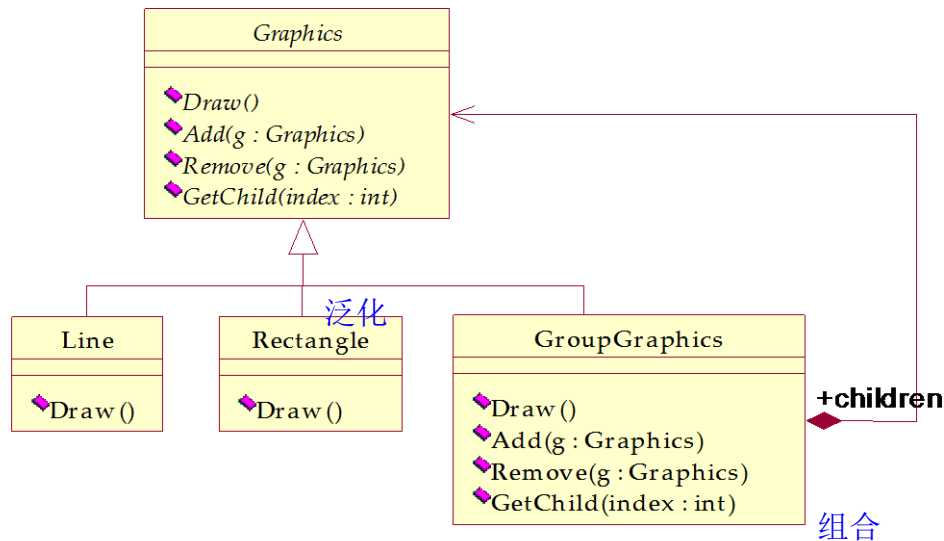
```
class FloatStack
{
private:
    int size;
public:
    int Push(float elem);
    int Pop();
    const float& GetTop();
};
```

# 3. 类图

## 3.5 类图例子

### 3.5.1 图形编辑器

- ※ 图形编辑器一般都具有一些基本图形，如直线、矩形等，用户可以直接使用基本图形画图，也可以把基本图形组合在一起创建复杂图形
- ※ 如果区别对待基本图形和组合图形，会使代码变得复杂，而且多数情况下用户认为二者是一样的
- ※ 组合模式可以用相同的方式处理两种图形



组合模式

**Graphics:** 基本图形和组合图形的父类，声明了所有图形共同的操作，如**Draw**；也声明了专用于组合图形管理子图形的操作，如**Add**、**Remove**

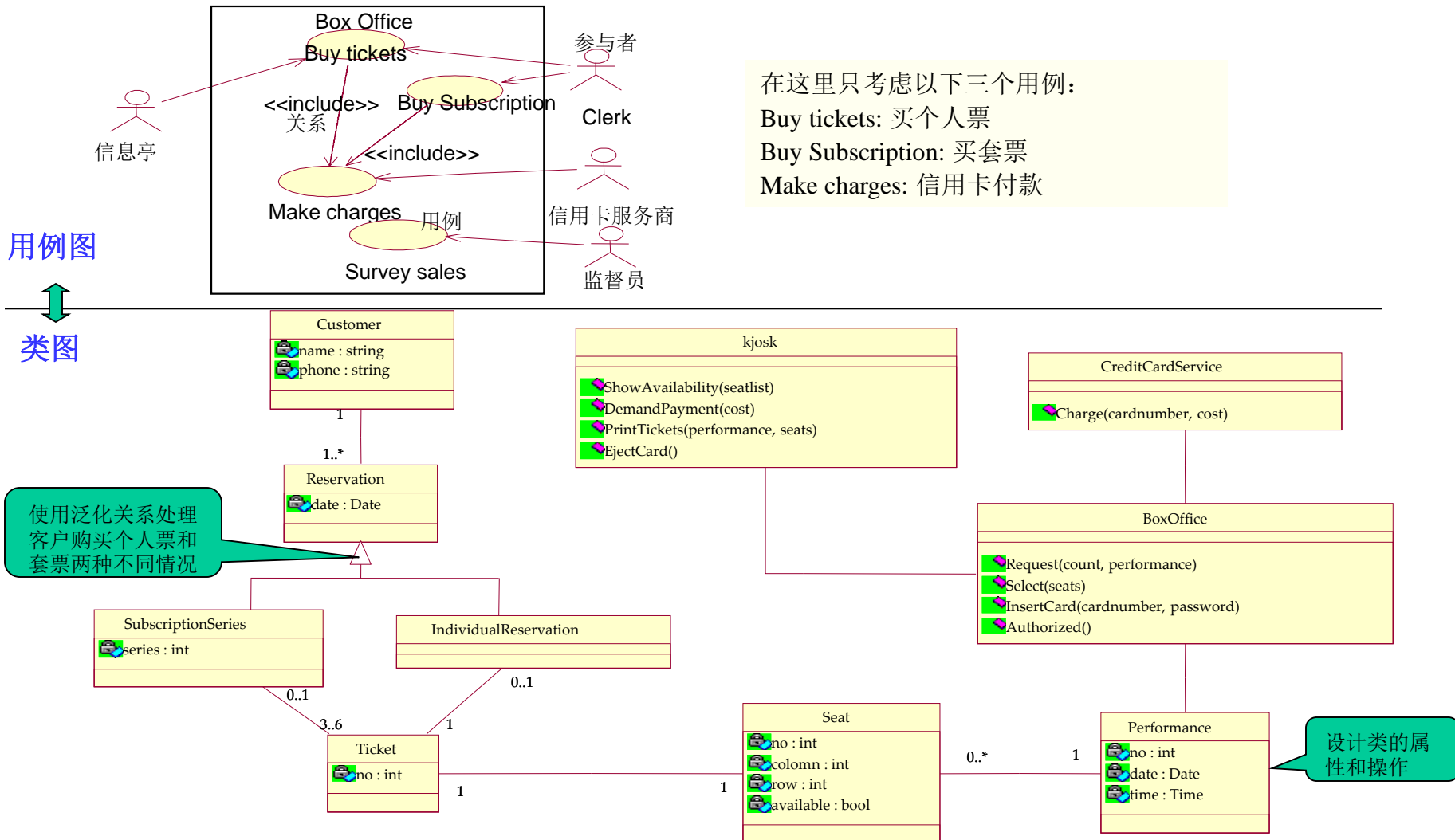
**Line、Rectangle:** 基本图形类

**GroupGraphics:** 组合图形类，与父类有组合关系，从而可以组合所有图形对象(基本图形和组合图形)

# 3. 类图

## 3.5.2 演出售票系统

在用例驱动的开发过程中，通过分析各个用例及参与者得到类图。分析用例图的过程中需要根据面向对象的原则设计类和关系，根据用例的细节设计类的属性和操作



# 3. 类图

## 3.6 习题

※ 右图描述了菜单(Menu)、菜单项(MenuItem)、抽象命令类(Command)和具体命令类(OpenCommand, PasteCommand)之间的关系, 完成1-4题

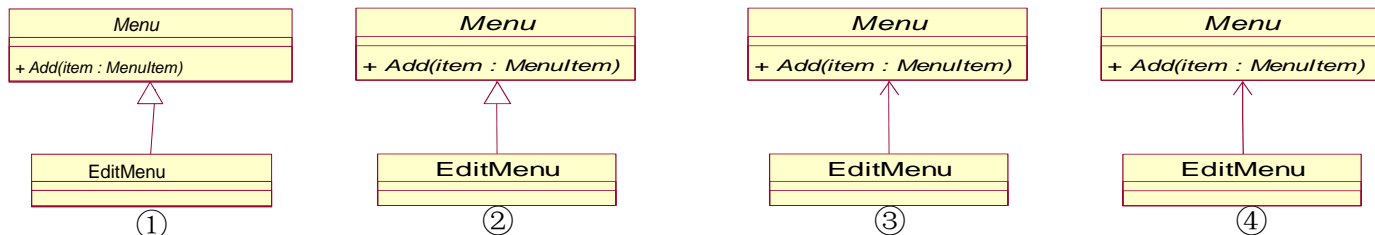
(1)哪两个类之间存在组合关系

- ① Menu、MenuItem
- ② MenuItem、Command
- ③ Command、OpenCommand
- ④ Command、PasteCommand

(2)OpenCommand和PasteCommand是什么关系

- ① 组合
- ② 泛化
- ③ 聚合
- ④ 没关系

(3)编辑菜单(EditMenu)是一种菜单, 下面哪个图较好的描述了二者之间的关系



(4)下面哪份代码(C++)最接近于图中对MenuItem的描述

```
class MenuItem
{
private:
    virtual void Click() =0;
public:
    Command* command;
};
```

①

```
class MenuItem
{
public:
    virtual void Click() = 0;
private:
    Command* command;
};
```

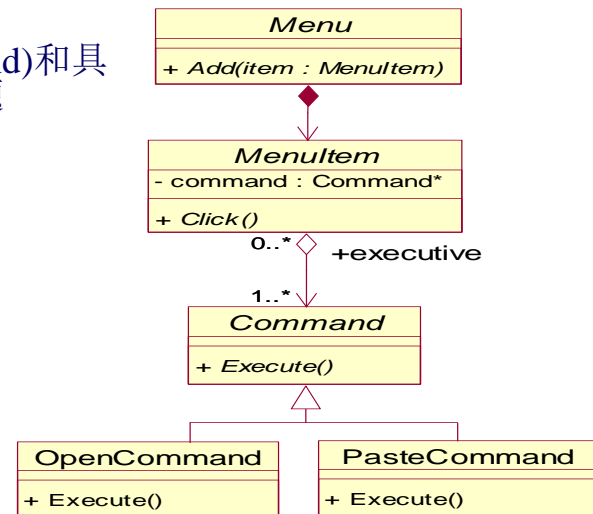
②

```
class MenuItem
{
private:
    virtual void Click() = 0;
    void undo();
public:
    Command* command;
};
```

③

```
class menuItem
{
public:
    virtual void Click() = 0;
private:
    Command* command;
};
```

④



# 3. 类图

※右图描述了图形接口(Graphics)、线段(Segment)、矩形(Rectangle)、点(Point)和三维点(Point3D)之间的关系，完成5-7题

(5)下面哪个关系没有在图中出现

- ①关联    ②泛化    ③实现    ④依赖

(6)下面对图中①②③④处的多重性的描述哪个不正确

- ① 0...\*    ② 1    ③ 0...\*    ④ 1

(7)下面哪份代码(Java)最接近于图中对Segment的描述

```
public class Segment implements Graphics
{
    private void Draw();
    public Point ptStart;
    public Point ptEnd;
}
```

①

```
public class Segment extends Graphics
{
    public void Draw();
    private Point ptStart;
    private Point ptEnd;
}
```

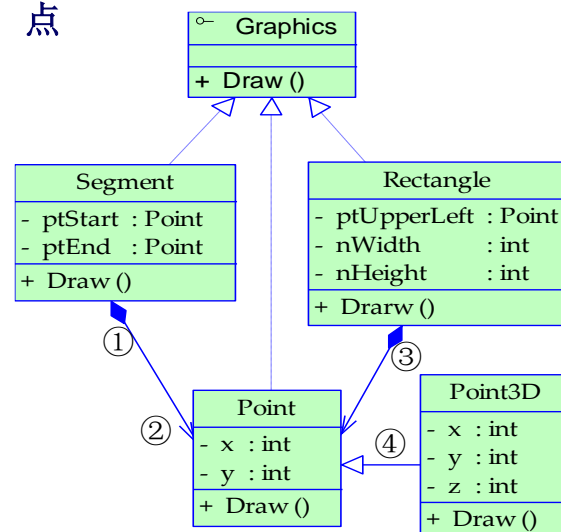
②

```
public class Segment implements Graphics
{
    private Point ptStart;
    private Point ptEnd;
    public void Draw();
}
```

③

```
public class segment implements graphics
{
    public void Draw();
    private Point ptStart;
    private Point ptEnd;
}
```

④




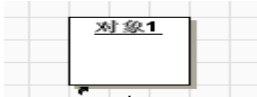

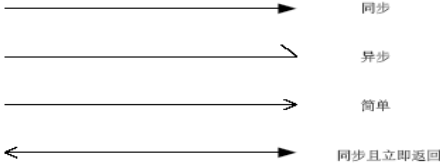


# 4. 顺序图

## 4.1 概要

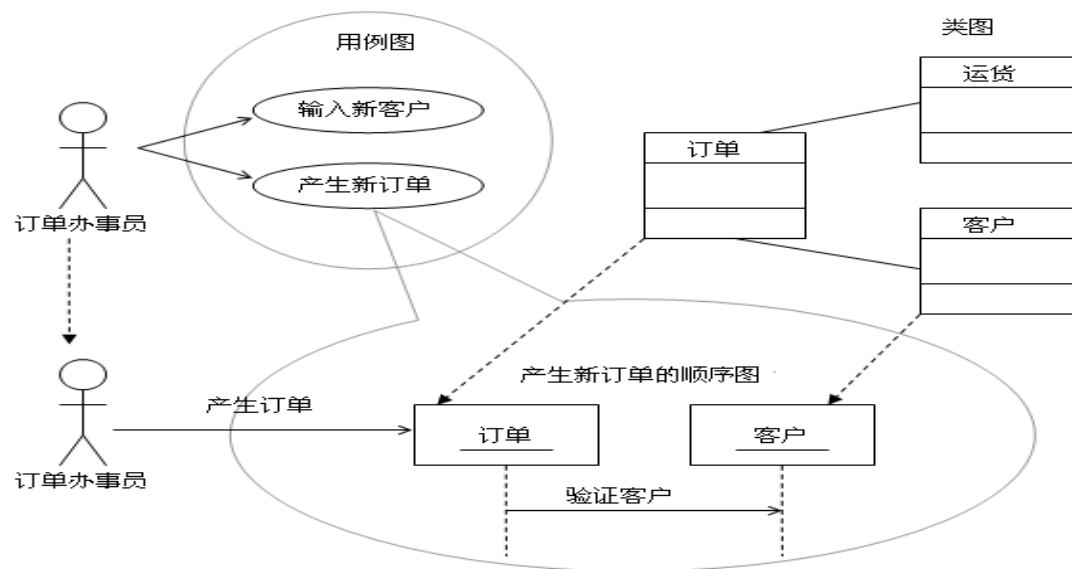
- ✧ 顺序图用来表示用例中的行为顺序。当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的事件。
- ✧ 顺序图展示对象之间的交互，这些交互是指在场景或用例的事件流中发生的。顺序图属于动态建模。
- ✧ 顺序图的重点在消息序列上，也就是说，描述消息是如何在对象间发送和接收的。表示了对象之间传递消息的时间顺序。
- ✧ 浏览顺序图的方法是：从上到下查看对象间交换的消息。

## 4.2 顺序图中的事物及解释

事物名称	解释	图
参与者	与系统、子系统或类发生交互作用的外部用户(参见用例图定义)。	
对象	顺序图的横轴上是与序列有关的对象。对象的表示方法是：矩形框中写有对象或类名，且名字下面有下划线。	
生命线	坐标轴纵向的虚线表示对象在序列中的执行情况(即发送和接收的消息，对象的活动)这条虚线称为对象的“生命线”。	
消息符号	消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头以时间顺序在图中从上到下排列。	

# 4. 顺序图

## 4.3 顺序图与用例图和类图的关系



# 4. 顺序图

## 4.4 顺序图例子

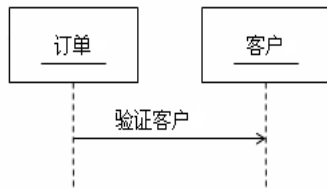
### 简单的例子

消息格式:

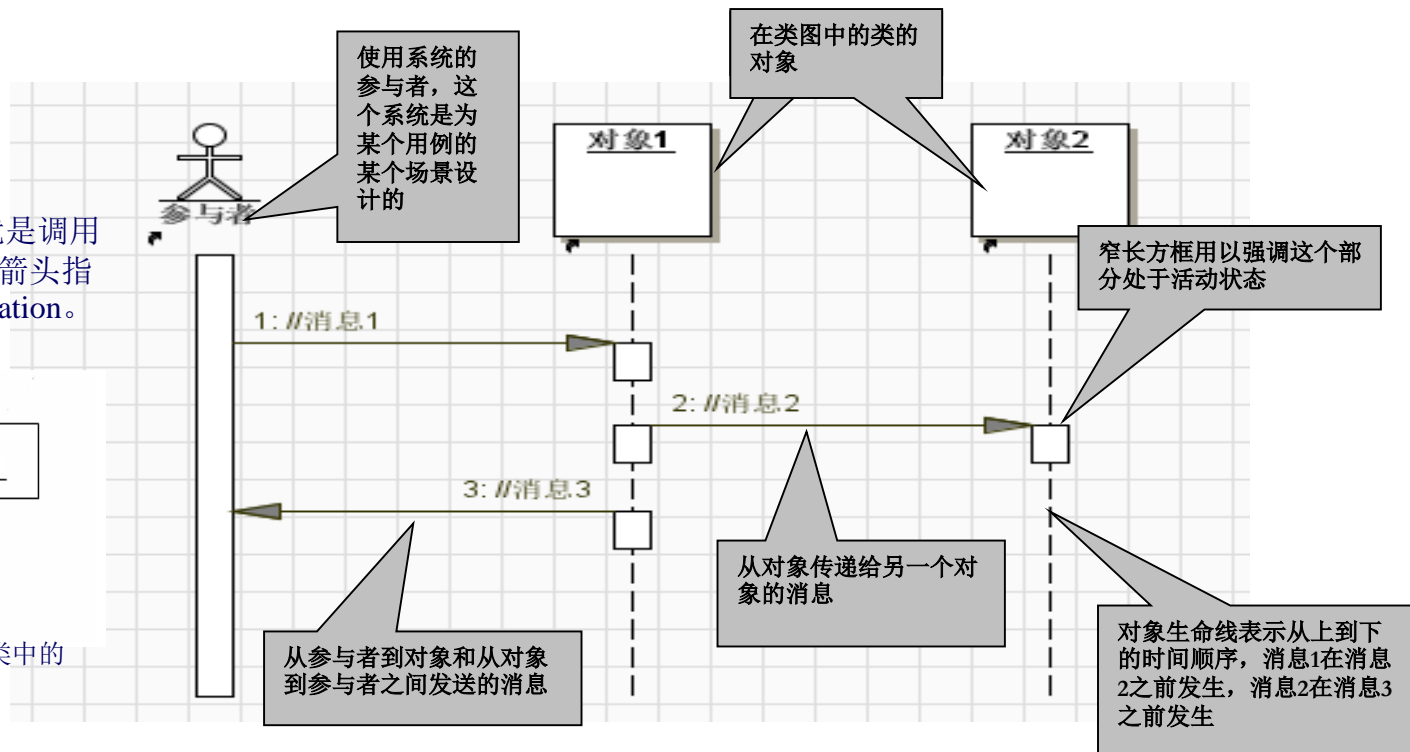
operation (parameter list)

向哪个对象发消息实际上就是调用它的类中的操作，就是调用箭头指向的对象所在类的一个operation。

例:

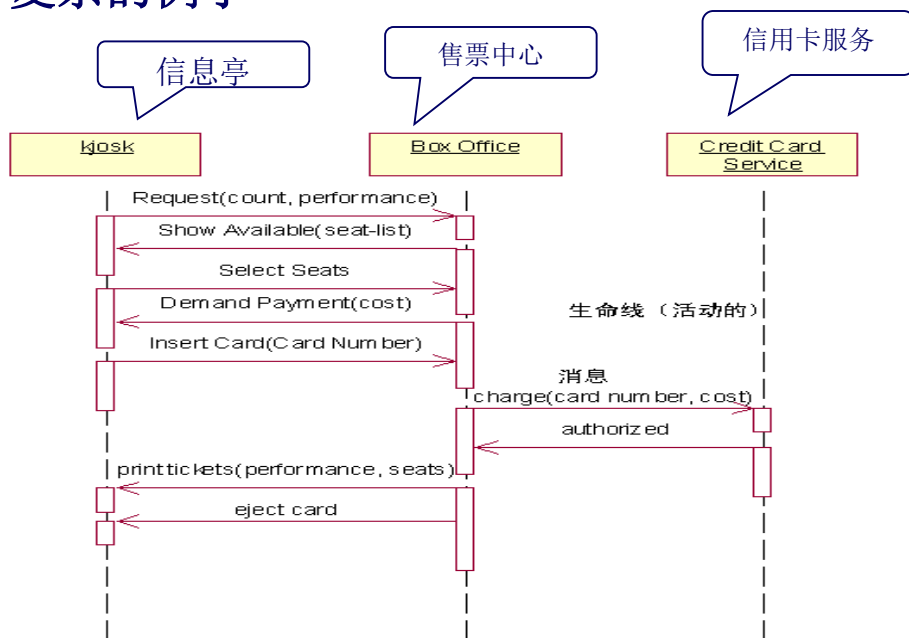


订单类发消息给客户类调用客户类中的“验证客户”操作



# 4. 顺序图

## 复杂的例子



从这个例子中可以看出:

Kiosk类中的操作有

Show Available (seat-list)

Demand Payment (cost)

printtickets (performance, seats)

eject card

Box Office中的操作有

Request (count, performance)

Select Seats

Insert Card (Card Number)

authorized

Credit Card Service类中的操作有

charge(card number, cost)

此图是描述购票这个用例的顺序图。顾客在信息亭与售票中心通话触发了这个用例的执行。顺序图中付款这个用例包括售票中心与信息亭和信用卡服务处使用消息进行通信过程。

此图中存在的事物有:

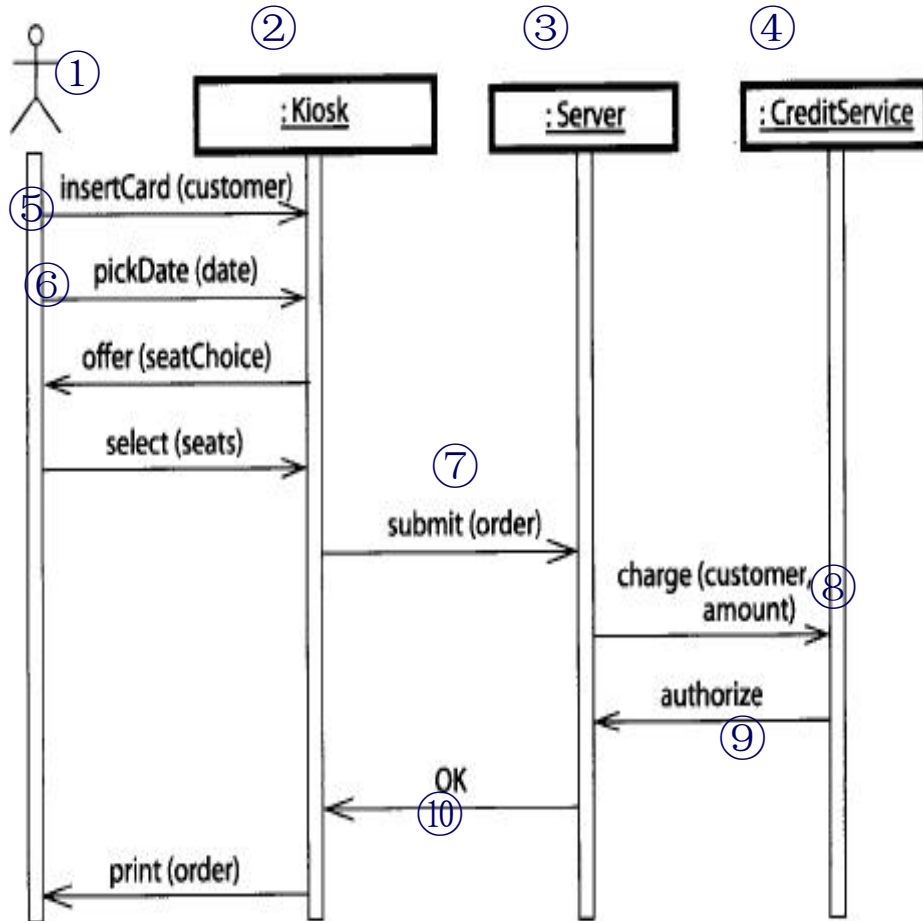
对象(信息亭 Kiosk, 售票中心 Box Office, 信用卡服务 Credit Card Service), 生命线, 消息符号。

信息亭发Request (count, performance)消息给售票中心, 表示调用售票中心类的Request (count, performance)操作, 来查询演出的信息。

售票中心发Show Available(seat-list)消息给信息亭, 表示调用信息亭类中的Show Available(seat-list)操作, 给出可用的座位表。

# 4. 顺序图

## 4.5 练习题



1 指出左图中的参与者? 1

A① B② C③ D④

2 哪些是对象? 234

A① B②③④ C④ D⑤⑥⑦⑧⑨⑩

3 Server类调用了CreditService类中的什么操作? 8

A⑦ B⑧ C⑦⑧ D⑧⑨

1. A 2. B 3. B

# 5. 协作图


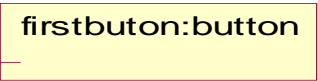

## 5.1 概要

协作图是一种交互图，强调的是发送和接收消息的对象之间的组织结构，使用协作图来说明系统的动态情况。

协作图主要描述协作对象间的交互和链接，显示对象、对象间的链接以及对象间如何发送消息。

协作图可以表示类操作的实现。

## 5.2 协作图中的事物及解释

事物名称	解释	图
参与者	发出主动操作的对象，负责发送初始消息，启动一个操作。	 Actor
对象	对象是类的实例，负责发送和接收消息，与顺序图中的符号相同，冒号前为对象名，冒号后为类名。	 firstbuton:button
消息流 (由箭头和标签组成)	箭头指示消息的流向，从消息的发出者指向接收者。标签对消息作说明，其中，顺序号指出消息的发生顺序，并且指明了消息的嵌套关系；冒号后面是消息的名字。	 标签

## 5.3 协作图中的关系及解释

关系名称	解释	关系实例
链接	用线条来表示链接，链接表示两个对象共享一个消息，位于对象之间或参与者与对象之间	

# 5. 协作图

## 5.4 消息标签

消息标签的Format: [前缀] [守卫条件] 序列表达式: [返回值: =] 消息名

✧ 前缀的语法规则: 序列号, 序列号, ..., 序列号 '/'

(前缀用来同步线程, 意思是在发送当前消息之前指定序列号的消息被处理.例: 1.1a, 1.1b/)

✧ 守卫条件的语法规则: [条件短句]

说明: 条件短句通常用伪代码或真正的程序语言来表示。 例: [x>=0]

✧ 返回值和消息名: 返回值表示一个消息的返回结果, 消息名指出了消息的名字和所需参数。

例: x:=calc ( n )

✧ 下面是一个完整的消息标签:

1.1a, 1.1b, 1.1c/	[x>=0]	1.2 *[i:=1..n] :	x	:=	calc(n)
↓	↓	↓	↓	↓	↓
前缀	守卫条件	序列表达式	返回值	:=	消息名

## 5.5 协作图与顺序图的区别和联系

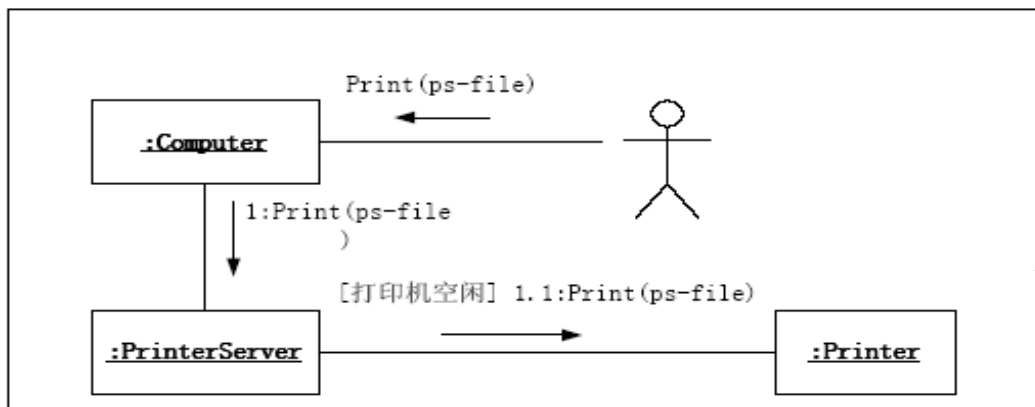
协作图和顺序图都表示出了对象间的交互作用, 但是它们侧重点不同。

- ✧ 顺序图清楚地表示了交互作用中的时间顺序(强调时间), 但没有明确表示对象间的关系。
- ✧ 协作图清楚地表示了对象间的关系(强调空间), 但时间顺序必须从顺序号获得。
- ✧ 协作图和顺序图可以相互转化。  
(进行协作图和顺序图的比较, 请参考练习题2)

# 5. 协作图

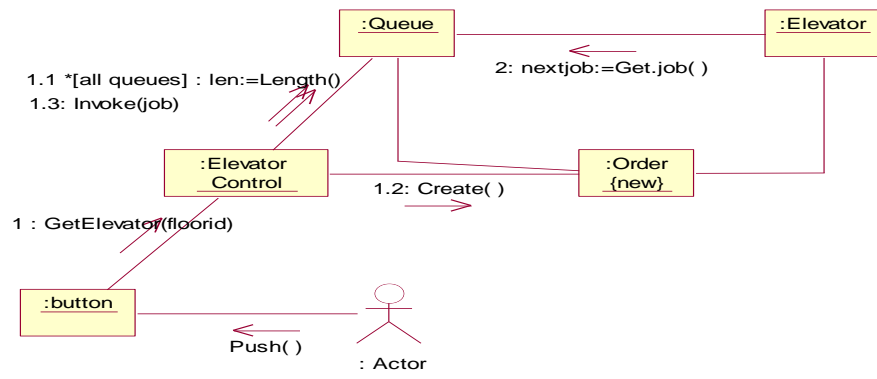
## 5.6 协作图例子

### 1. 打印操作的协作图



actor发送Print消息给Computer, Computer发送Print消息给PrinterServer, 如果打印机空闲, PrinterServer发送Print消息给printer

### 2. 乘坐电梯的协作图



图中存在的事物有:

参与者  
按钮对象  
电梯控制对象  
命令对象  
工作队列  
电梯对象

图中存在的关系有:

链接

参与者需要乘坐电梯, 他从系统外部按下按钮, 让电梯到达他想去的楼层。此时, 电梯系统的操作被启动, 电梯控制对象以循环的方式检查所有的电梯, 从中选择一个工作队列长度最短的。然后, 它创建一个作业命令, 并将该命令放入对应电梯的工作队列, 接着激活队列。电梯对象并发运行, 从它的队列中选择一个作业并执行。电梯是一个活动对象, 它与它的控制线程并发执行。



# 5. 协作图

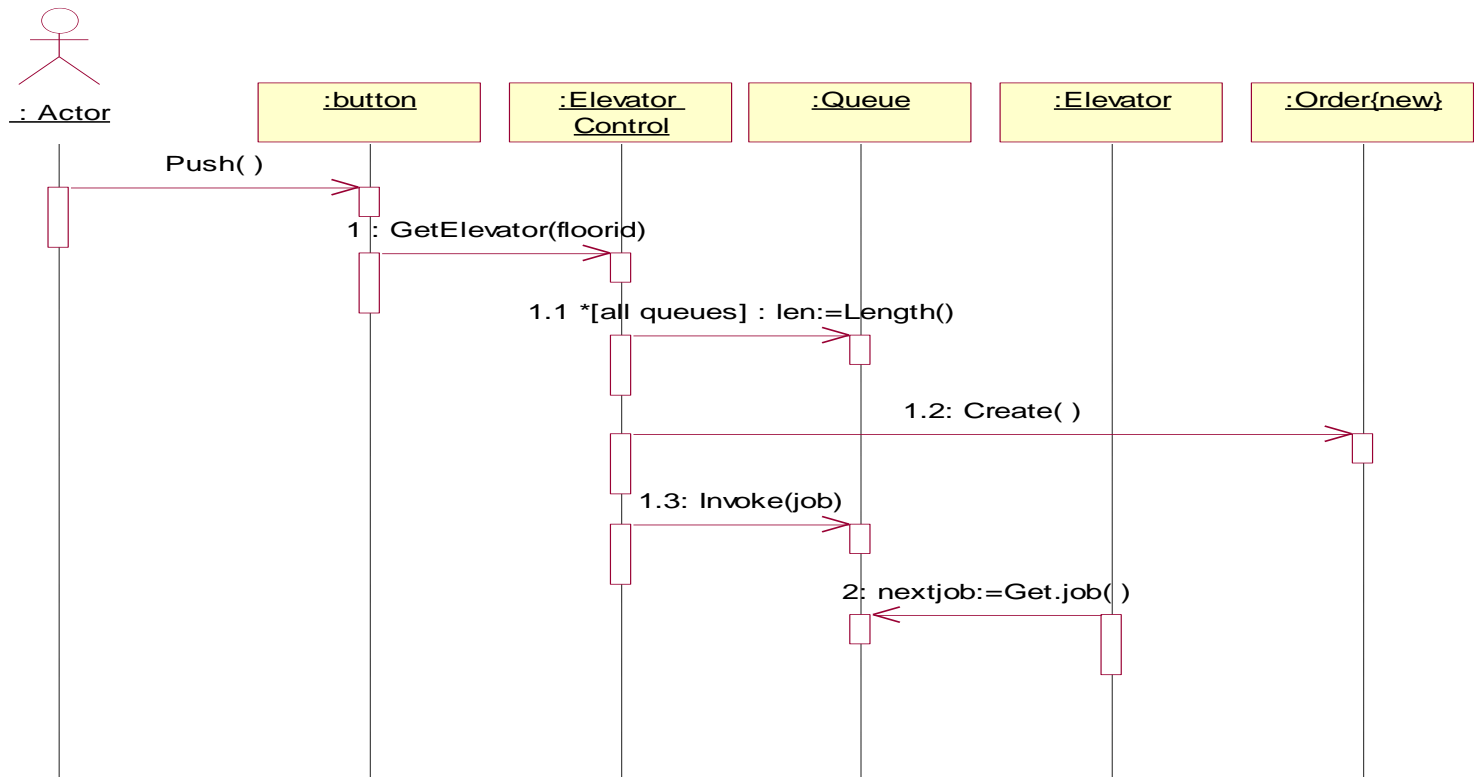
## 5.7 练习题

1. 请指出下面的消息标签各部分的内容。

- 1:display( A)  
A. 序列表达式:消息名 B. 返回值: 消息名 C. 序列表达式: 消息名
- [mode=display] 1.2.3.7: redraw( B)  
A. 序列表达式 返回值 消息名 B. 守卫条件 序列表达式 消息名
- 2 \* [n:=a . . z] : prim:=nextPrim(prim)B  
A. 守卫条件 序列表达式 消息名 B. 序列表达式 返回值 消息名
- 3.1 [x<0] : foo( B)  
A. 序列表达式 守卫条件 B. 守卫条件 消息名 C. 序列表达式 消息名
- 1.1a, 1.1b/1.2 : continue( A)  
A. 前缀 序列表达式 消息名 B. 后缀 守卫条件 消息名

# 5. 协作图

2. 请对比本章中的协作图与其相应的顺序图，做练习题。



①请在下图中指出，循环计算各个电梯的工作队列长度的消息的序号号。A

A 1.1      B1.2      C1.3      D2

②请说明消息1.3所进行的操作Invoke（job），应属于哪个类所具有的方法。A

A Queue类    B ElevatorControl类    C Elevator类    D Button类

3. 练习题答案

1. ①A      ②B      ③B      ④C      ⑤A

2. ①A      ②A

# 6. 状态图

## 6.1 状态图概要

### 6.1.1 状态图

说明对象在它的生命期中响应事件所经历的状态序列，以及它们对那些事件的响应。

### 6.1.2 状态图用于

揭示Actor、类、子系统和组件的复杂特性。  
为实时系统建模。

## 6.2 状态图的组成

### 6.2.1 状态

对象的状态是指在这个对象的生命期中的一个条件或状况，在此期间对象将满足某些条件、执行某些活动，或等待某些事件。

### 6.2.2 转移

转移是由一种状态到另一种状态的迁移。这种转移由被建模实体内部或外部事件触发。

对一个类来说，转移通常是调用了一个可以引起状态发生重要变化的操作的结果。

# 6. 状态图

## 6.3 状态图中的事物及解释

状态	上格放置名称，下格说明处于该状态时，系统或对象要做的工作(见可选活动表)	<div>Enter Password</div> <div>entry / set echo to star; password.reset() exit / set echo normal digit / handle character clear / password.reset() help / display help</div>
转移	转移上标出触发转移的事件表达式。如果转移上未标明事件，则表示在源状态的内部活动执行完毕后自动触发转移	消息(属性)[条件]/动作
开始	初始状态(一个)	
结束	终态(可以多个)	

## 6.4 状态的可选活动表

转换种类	描述	语法
入口动作	进入某一状态时执行的动作	entry/action
出口动作	离开某一状态时执行的动作	exit/action
外部转换	引起状态转换或自身转换，同时执行一个具体的动作，包括引起入口动作和出口动作被执行的转换	e(a:T)[exp]/action
内部转换	引起一个动作的执行但不引起状态的改变或不引起入口动作或出口动作的执行	e(a:T)[exp]/action

# 6. 状态图

## 6.5 例子

### (1) 对象的状态图

图中包含以下状态

初始状态

Available状态

Locked状态

Sold状态

状态间的转移

初始状态→Available状态

票被预订(lock): Available→Locked

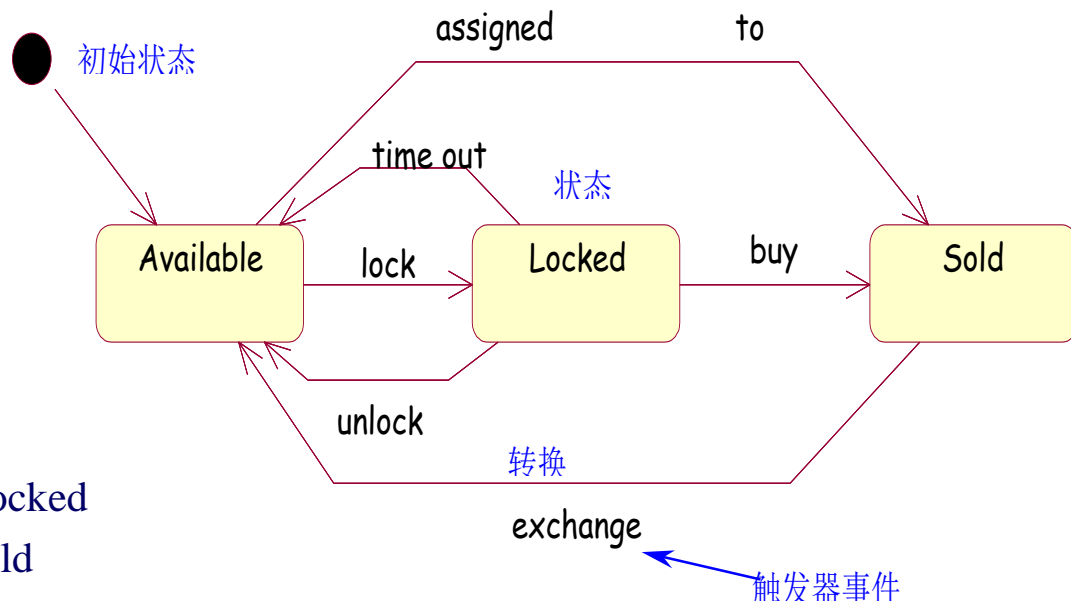
预定后付款(buy): Locked→Sold

预定解除(unlock): Locked→Available

预定过期(time out): Locked→Available

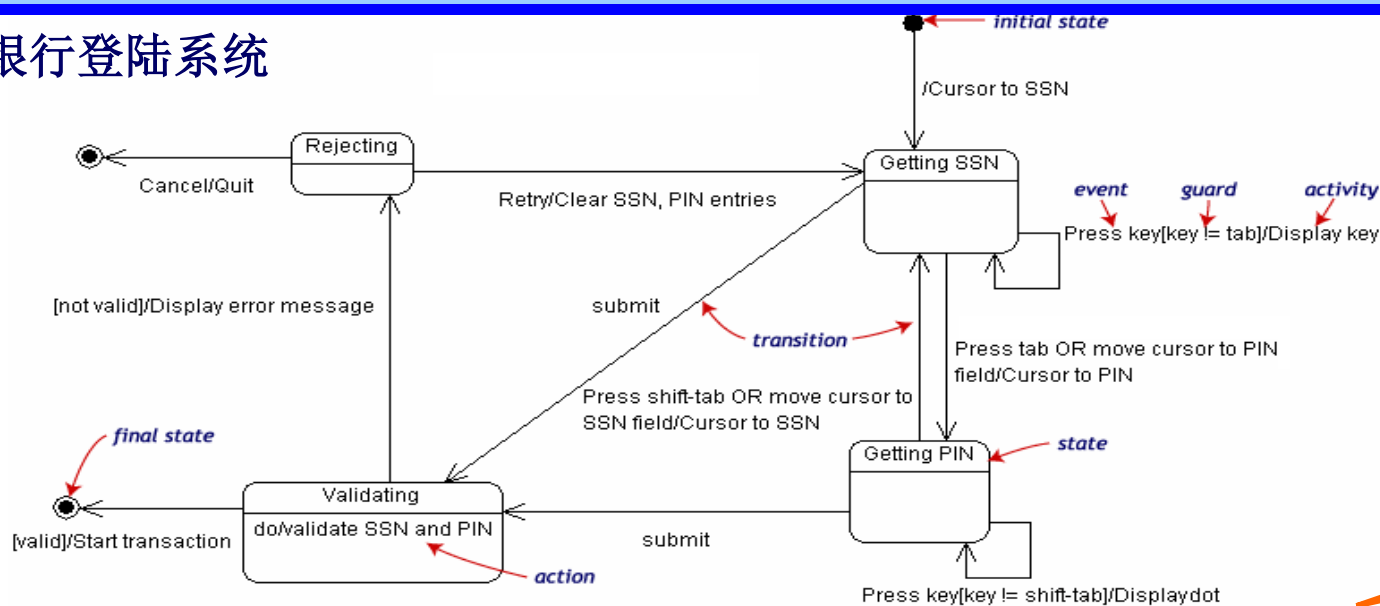
直接购买(assigned to): Available→Sold

换其它票(exchang) , 该票重有效: Sold→Available



# 6. 状态图

## (2) 网上银行登陆系统



状态转移的过程

登陆要求提交个人社会保险号(SSN)和密码(PIN)经验证有效后登陆成功。

登陆过程包括以下状态:

- ※ 初态(Initial state)
- ※ 获取社会保险号状态(Getting SSN)
- ※ 获取密码状态(Getting PIN)
- ※ 验证状态(Validating)
- ※ 拒绝状态(Rejecting)
- ※ 终态(Final state)

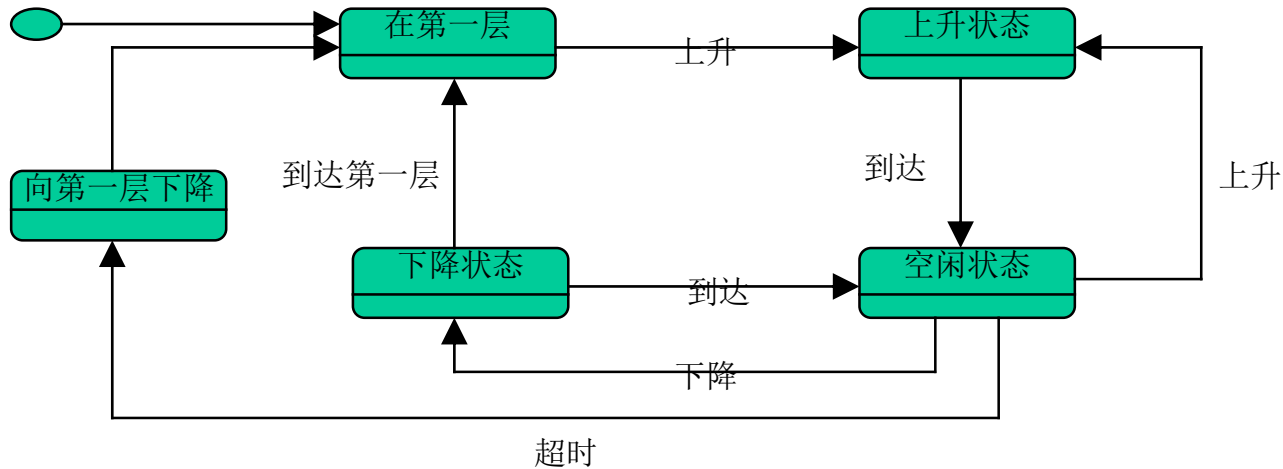
出发状态	动作	到达状态
Initial state	移动鼠标到 SSN	Getting SSN
Getting SSN	键入非tab键, 显示键入内容	Getting SSN
	键入tab键, 或移动鼠标到BIN	Getting PIN
	提交	Validating
Getting PIN	键入非shift-tab键, 显示 “ * ”	Getting PIN
	键入shift-tab键, 或移动鼠标到SSN	Getting SSN
	提交	Validating
Validating	验证提交信息有效, 状态转移	Final state
	验证提交信息无效, 显示错误信息	Rejecting
Rejecting	退出	Final state
	重试, 清除无效的SSN, PIN	Getting SSN

有两个不同的终态

# 6. 状态图

## 6.7 练习

分析下面的状态图，回答问题



(1) 以下那些图形元素是对状态的描述? c

(a) 超时 (b) 到达 (c) 在第一层

(2) 空闲状态 超时后转移到\_\_\_\_状态a

(a) 向第一层下降 (b) 上升状态 (c) 终态

习题答案

(1)

(c)

(2)

(a)

# 7. 活动图

## 7.1 活动图概要

- ※ 描述系统的动态行为。
- ※ 包含活动状态(ActionState)，活动状态是指业务用例的一个执行步骤或一个操作，不是普通对象的状态。
- ※ 活动图适合描述在没有外部事件触发的情况下的系统内部的逻辑执行过程；否则，状态图更容易描述。
- ※ 类似于传统意义上的流程图。
- ※ 活动图主要用于：
  - 业务建模时，用于详述业务用例，描述一项业务的执行过程；
  - 设计时，描述操作的流程。





## 7.2活动图事物

活动 (ActionState)	动作的执行	
起点 (InitialState)	活动图的开始	
终点(FinalState)	活动图的终点	
对象流(ObjectFlowState)	活动之间的交换的信息	
发送信号(signalSending)	活动过程中发送事件，触发另一活动流程	
接收信号(SignalReceipt)	活动过程中接收事件，接收到信号的活动流程开始执行	
泳道(SwimLane)	活动的负责者	



# 7. 活动图

## 7.3 活动图关系

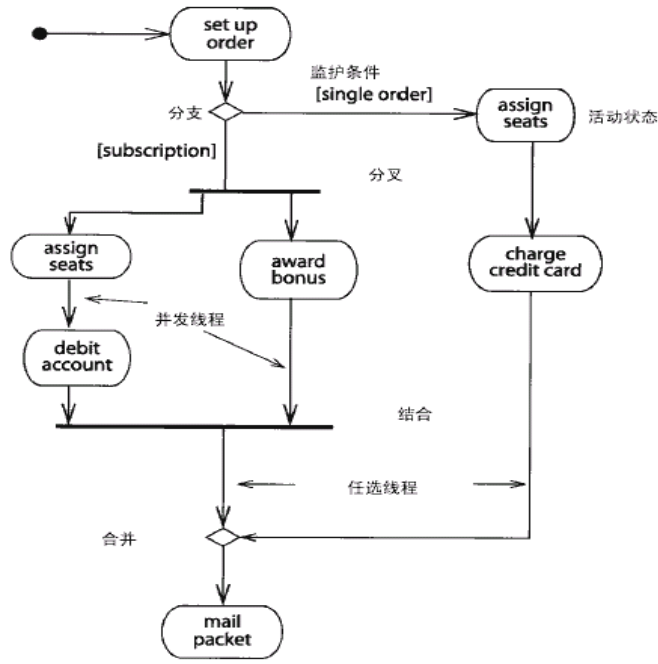
迁移(transition)	活动的完成与新活动的开始	
分支(junction point)	根据条件，控制执行方向	
分叉(fork)	以下的活动可并发执行	
结合(join)	以上的并发活动再此结合	

## 7.4 活动图实例

### 1. 一般的活动图

本活动图描述一个处理订单的用例执行过

- (1) 执行 setup order
- (2) 根据 order 的类型是执行不同的分支：
  - single order: 执行 assign seat、charge credit card
  - subscription: 同时执行 assignseats、debit account或 award bonus
- (3) 最后 mail packet。



# 7. 活动图

## 2. 带泳道的活动图

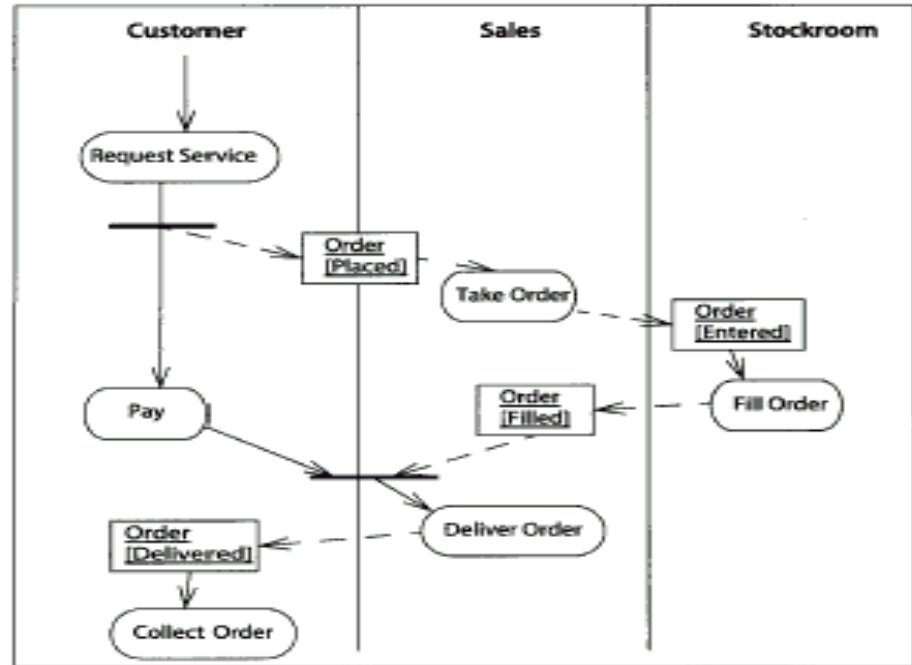
本例为一个按活动职责(带泳道)组织的处理订单用例的活动图(模型中的活动按职责组织)。活动被按职责分配到用线分开的不同区域(泳道):

Customer

Sales

Stockroom

- (1) 顾客要求服务, Sales负责接收订单, 并提交到Stockroom
- (2) Stockroom处理订单, 与此同时, Customer付款, 并由Sales处Deliver order至Customer。



# 7. 活动图

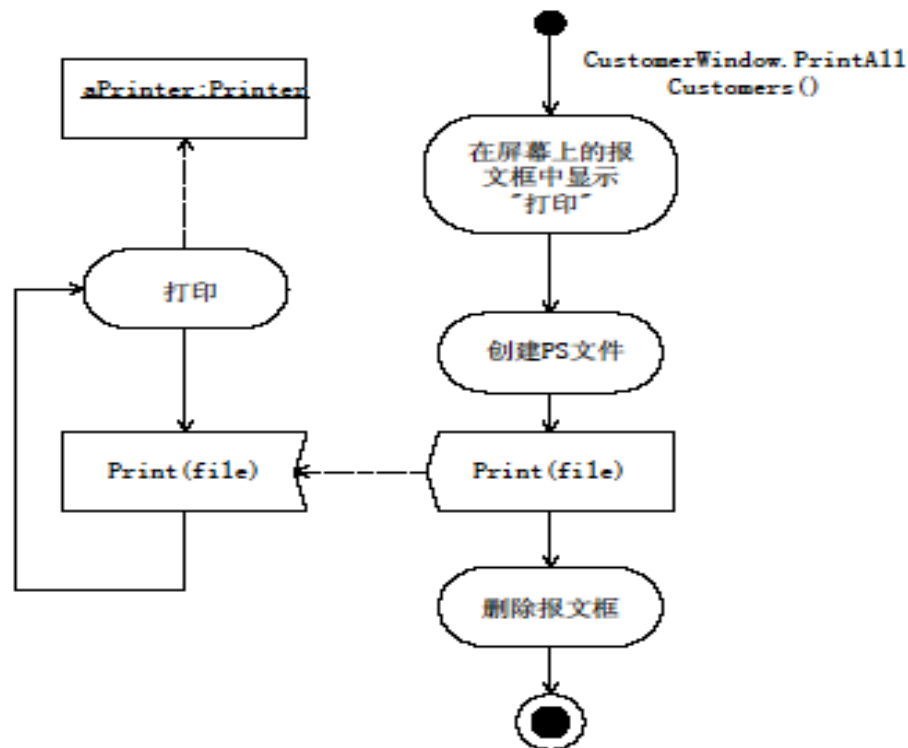
## 7.5 活动图练习

1. 请选择下面所列的活动图的事物中，表示信号的是( C )，表示对象流的是( B )。



2. 关于右面的活动图，下面的说法中正确的是 ( C )。

- A aPrinter:Printer是信号。
- B 操作开始从“删除报文框”活动开始。
- C 在“创建PS文件”和“删除报文框”活动中发送“print(file)”信号。
- D 信号发送的方向是从左到右。



### 习题答案

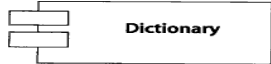

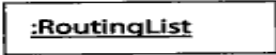
1 C, B 2 C

# 8. 构件图

## 8.1 构件图概要

构件图用于静态建模，是表示构件类型的组织以及各种构件之间依赖关系的图。  
构件图通过对构件间依赖关系的描述来估计对系统构件的修改给系统可能带来的影响。



## 8.2 构件图中的事物及解释

事物名称	含义	图例
构件	指系统中可替换的物理部分，构件名字(如图中的Dictionary)标在矩形中，提供了一组接口的实现。	
接口	外部可访问到的服务 (如图中的Spell-check)。	
构件实例	节点实例上的构件的一个实例，冒号后是该构件实例的名字(如图中的RoutingList)。	

可替换的物理部分包括软件代码、脚本或命令行文件，也可以表示运行时的对象，文档，数据库等。

节点(node)是运行时的物理对象，代表一个计算机资源。具体请参见教程“部署图(deployment diagram)”部分。

## 8.3 构件图中的关系及解释

关系名称	含义	图例
实现关系	构件向外提供的服务。	
依赖关系	构件依赖外部提供的服务(由构件到接口)。	

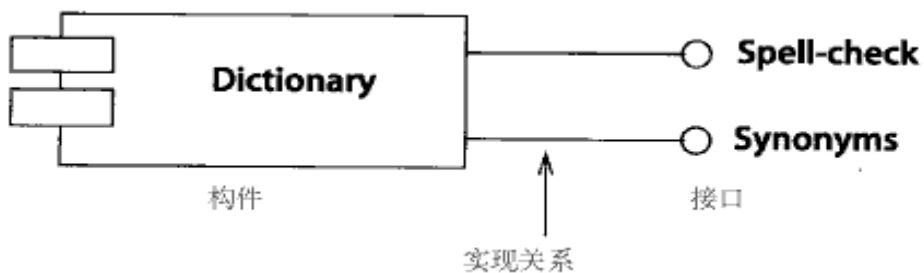
# 8. 构件图

## 8.4 构件图的例子

### 实例1.

图中的构件名称是Dictionary字典。

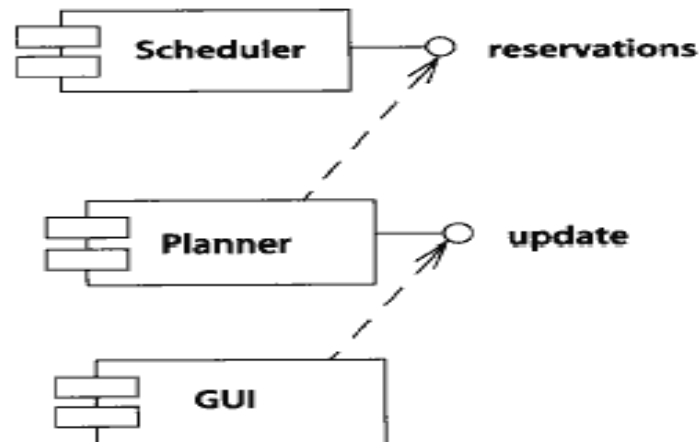
该构件向外提供两个接口，即两个服务Spell-check拼写检查、Synonyms同义词。



### 实例2.

图中“Planner计划者”构件向外提供一个“update更新”接口服务。

同时，该构件要求外部接口提供一个“Reservations预定”服务。



# 8. 构件图

## 实例3

图中**依赖关系**包括:

顾客需要信息亭接口提供服务

售票员需要职员接口提供服务

信用卡付款需要信用卡代理提供服务

职员接口需要预订销售、个人销售和团体销售提供服务

管理接口需要数据库状态提供服务

售票处需要付款和购买提供服务

等等.....

图中**实现关系**包括:

信用卡付款提供付款服务

票数据库提供购买和状态查询服务

售票处提供预订购买、个人购买和团体购买服务

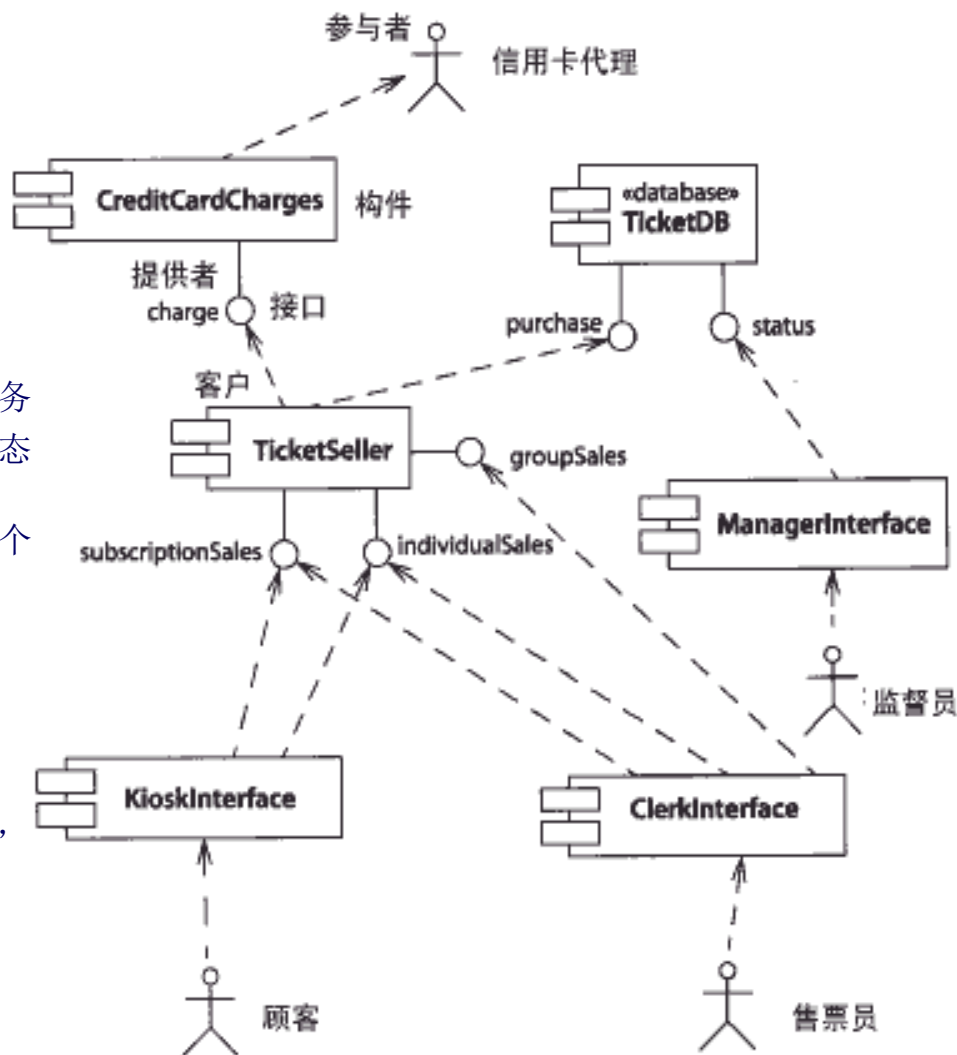
## 情景描述

情景一:

购买个人票可以通过公用信息亭订购也可直接向售票员购买,但购买团体票只能通过售票员。

情景二:

买票的人可以根据任意选择预订销售或个人销售或团体销售中的一种方式,售票处为了方便销售,需要信用卡付款服务的支持,同时也必然需要票数据库处在有票可卖的状况中。



# 8. 构件图

## 8.5 习题

1. 构件图用于那种建模阶段？ **B**  
A.动态建模                      B.静态建模
2. 一个构件只能对特定的另一个构件提供特定的一种服务。这种说法正确吗？ **B**  
A.正确                              B.错误
3. 构件图用于描述系统中各物理部件之间的服务的依赖提供关系。这种说法正确吗？ **A**  
A.正确                              B.错误
4. 构件图中实线箭头表示服务的依赖，虚线箭头表示服务的提供。这种说法正确吗？ **B**  
A.正确                              B.错误

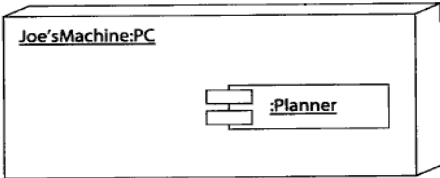



习题答案： 1.B 2.B 3.A 4.B

# 9. 部署图

## 9.1 部署图概要

部署图用于静态建模，是表示运行时过程节点结构、构件实例及其对象结构的图。  
如果含有依赖关系的构件实例放置在不同节点上，部署视图可以展示出执行过程中的瓶颈。  
部署图的两种表现形式：实例层部署图和描述层部署图(会在后面的实例中给出)。

## 9.2 部署图中的事物及解释

事物名称	解释	图例
节点	节点用一长方体表示，长方体中左上角的文字是节点的名字 (如图中的Joe’sMachine:PC) 。 节点代表一个至少有存储空间和执行能力的计算资源。 节点包括计算设备和(至少商业模型中的)人力资源或者机械处理资源，可以用描述符或实例代表。 节点定义了运行时对象和构件实例(如图中的Planner 构件实例)驻留的位置。	
构件	系统中可替换的物理部分。	
接口	外部可访问的服务。	
构件实例	构件的一个实例。	



# 9. 部署图

## 9.3 部署图中的关系及解释

关系名称	解释	图例
实现关系	构件向外提供服务。	(节点内) 
依赖关系	构件依赖外部提供的服务(由构件到接口)。	
关联关系	通信关联。	(节点间) 
其他关系	对象的移动(一个位置到另一个位置)。	

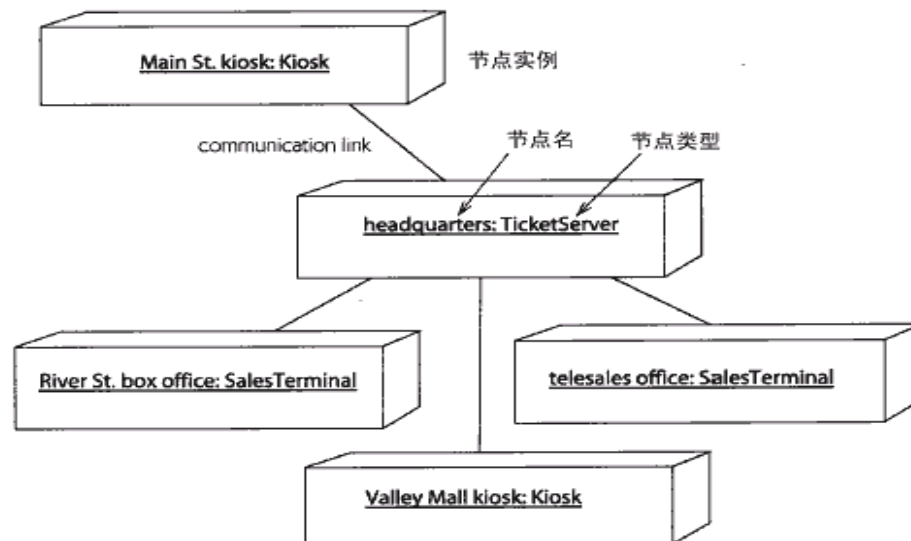
## 9.4 部署图的例子

### 实例1 实例层部署图

实例层部署图描述各节点和它们之间的连接。

本图中的信息与上张描述层部署图中的内容是相互对应的。

图中的关系是各个节点之间存在的通信关系。



# 9. 部署图

## 实例2 描述层部署图

描述层部署图表示了系统中的各节点和每个节点包含的构件。

图中包括的各种关系如下：

通信链关系(不带箭头的直线)

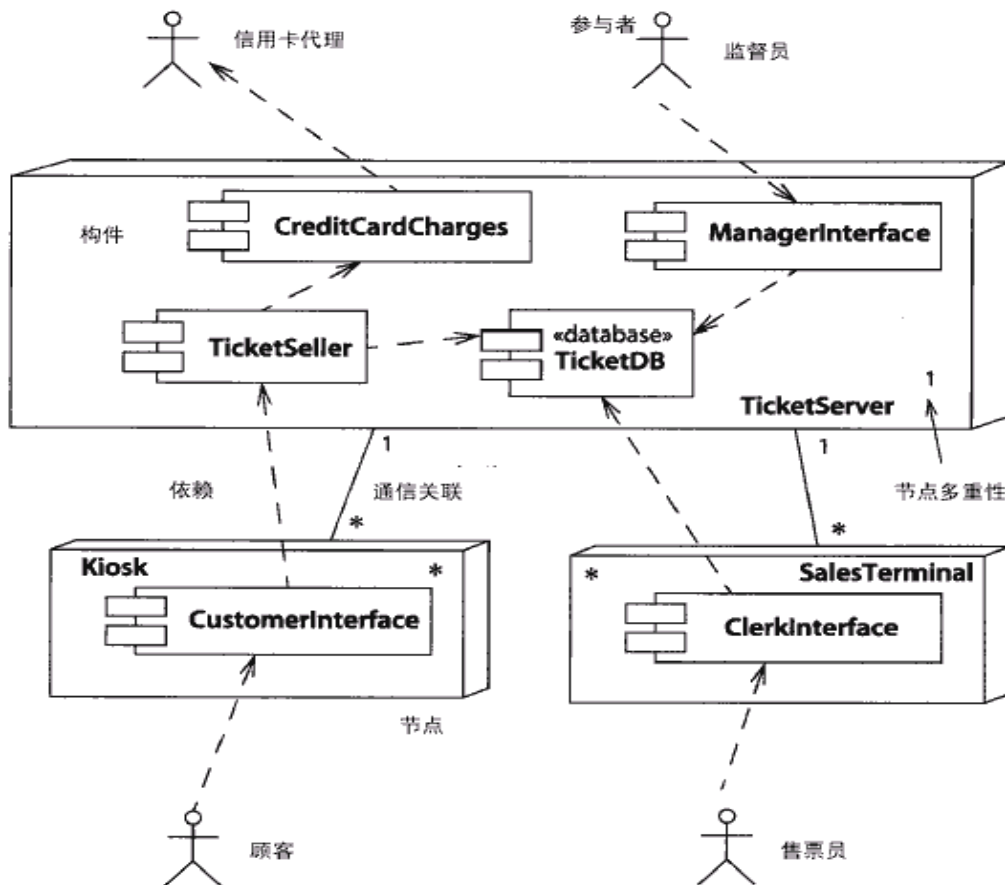
TicketServe票服务器与Kiosk信息厅之间存在一对多的通信关联；与SalesTerminal售票终端也存在一对多的通信关联；

依赖关系(带箭头的虚线)

TicketSeller售票构件依赖CreditCardCharges信用卡付款构件和TicketDB票数据库构件提供的服务。

图中**顾客购票**的情景如下：

顾客通过位于Kiosk节点的顾客接口控件进行购票的操作，该顾客接口控件的购票操作依赖于处于TicketServer节点上的售票构件提供的服务，售票构件要完成售票操作，又要依赖同一节点上信用卡付款构件提供的付款服务和票数据库构件



**节点TicketServer(售票服务)上的构件：**

CreditCardCharges/ManagerInterface/ TicketSeller/TicketDB

**节点Kiosk(信息亭)上的构件：**

CustomerInterface

**SalesTerminal(销售终端)上的构件：**

ClerkInterface

# 9. 部署图

## 9.5 关于部署图与构件图

部署图与构件图相同的构成元素：

构件、接口、构件实例、构件向外提供服务、构件要求外部提供的服务。

部署图与**构件图**的关系：

部署图表现构件实例；

构件图表现构件类型的定义。

部署图偏向于描述构件在节点中运行时的状态，描述了构件运行的环境；  
构件图偏向于描述构件之间相互依赖支持的基本关系。

# 9. 部署图

## 9.6 习题

- 1.部署图用于那种建模阶段? **B**  
A.动态建模                      B.静态建模
- 2.部署图表现构件实例，构件图表现构件类型定义。这种说法正确吗? **A**  
A.正确                              B.错误
- 3.部署图中一个节点实例的名称为HostMachine:Server，其中Server是\_\_**B**\_\_，HostMachine是\_\_**A**\_\_。  
A.节点名                              B.节点类型
- 4.各节点之间存在着虚线剪头表示的依赖关系，也存在着实线箭头表示的服务提供关系。这种说法正确吗? **B**  
A.正确                              B.错误
- 5.“接口”表示\_\_**A**\_\_对外提供的服务。  
A.构件                                  B.节点

习题答案：1.B 2.A 3.B， A 4.B 5.A

- 1 What to do? (Not how to do.)
- 2 Actor's point of view?
- 3 Value for the actor?
- 4 Entire flow of events?

- 1 用例描述了系统应该做什么，而不是如何去做。
- 2 用例必须依据参与者的视点。(即应该从参与者如何使用系统的角度出发定义用例,而不是从系统自身的角度)。
- 3 用例必须为参与者提供可辨识的价值。
- 4 用例及其参与者必须捕获系统使用过程中的一个完整的事件流。

# 附录 UML 学习参考书籍



# 附录 UML 学习参考书籍

1. 《用例驱动UML对象建模应用——范例分析》

**Doug Rosenberg、Kendall Scott 著**，人民邮电出版社，2005。

2. 《UML精粹——标准对象建模语言简明指南》(第3版)

**Martin Fowler 著**，徐家福 译，清华大学出版社，2005。

3. 《UML对象、组件和框架——Catalysis方法》

**Desmond Francis D'Souza、Alan Cameron Wills 著**，清华大学出版社，2004。

4. 《UML和模式应用》(第2版)

**Craig Larman 著**，机械工业出版社，2004。

5. 《有效用例模式》

**Steve Adolph、Paul Bramble 著**，车立红 译，清华大学出版社，2003。

6. 《用例建模》，**Kurt Bittner 著**，姜昊 译，清华大学出版社，2003

7. 《UML和统一过程实用面向对象的分析和设计》

**Jim Arlow、Ila Neustadt 著**，机械工业出版社，2003。

8. 《UML风格》**Scott W. Ambler 著**，王少峰 译，清华大学出版社，2004。

9. 《UML用户指南》

**Grady Booch、Ivar Jacobson 著**，邵维忠等译，机械工业出版社，2001年6月。

10. 《UML参考手册》

**Ivar Jacobson、James Rumbaugh 著**，姚淑兰，唐发根译。机械工业出版社，2001。



# 附录 UML 建模工具

1. [www.umlchina.com](http://www.umlchina.com)
2. [www.uml.org.com](http://www.uml.org.com)
3. [www.rational.com](http://www.rational.com)
4. [www.uml.net.cn](http://www.uml.net.cn)





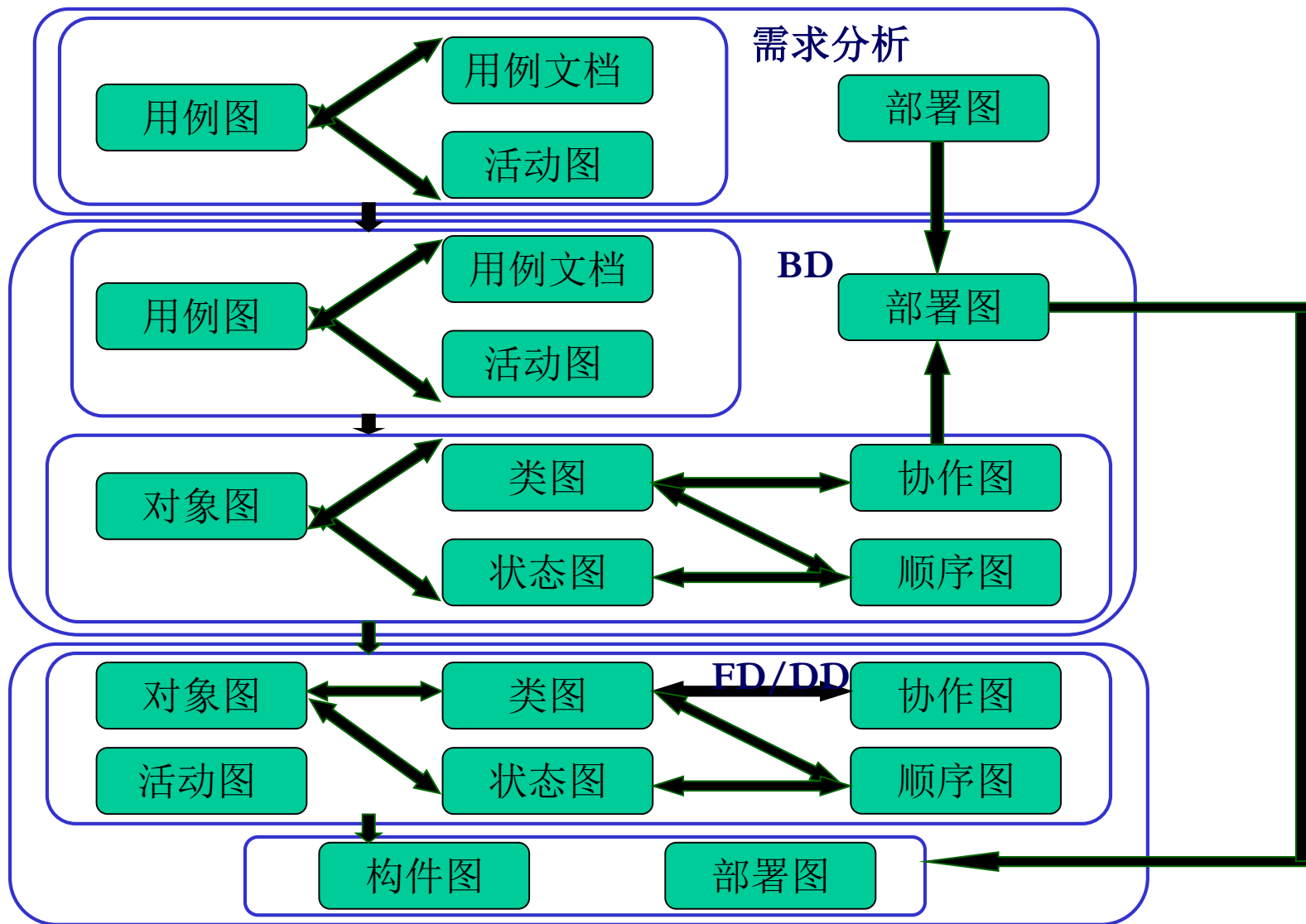
# 附录 各个阶段用到UML模型图

	需求分析	BD	FD	DD
用例图	◎	◎	—	—
类图	—	○	◎	◎
顺序图	—	○	◎	◎
活动图	○	○	○	○
对象图	—	△	△	△
协作图	—	△	△	△
状态图	—	△	○	○
构件图	—	—	○	○
部署图	○	○	△	△

◎：最适用  
○：适用  
△：可能适用  
—：不适用

# 附录 UML 全部图的关系

全部图之间的关系



# UML总结1

标准建模语言UML的重要内容可以由下列五类图（共9种图形）来定义：

第一类是用例图，包括用例图从用户角度描述系统功能，并指出各功能的操作者。

第二类是静态图 (Static diagram)，包括类图、对象图（和包图）。其中类图描述系统中类的静态结构。不仅定义系统中的类，表示类之间的联系如关联、依赖、聚合等，也包括类的内部结构（类的属性和操作）。类图描述的是一种静态关系，在系统的整个生命周期都是有效的。对象图是类图的实例，几乎使用与类图完全相同的标识。他们的不同点在于对象图显示类的多个对象实例，而不是实际的类。一个对象图是类图的一个实例。由于对象存在生命周期，因此对象图只能在系统某一段时间段存在。包由包或类组成，表示包与包之间的关系。包图用于描述系统的分层结构。

第三类是行为图 (Behavior diagram)，包括状态图、活动图。描述系统的动态模型和组成对象间的交互关系。其中状态图描述类的对象所有可能的状态以及事件发生时状态的转移条件。通常，状态图是对类图的补充。在实用上并不需要为所有的类画状态图，仅为那些有多个状态其行为受外界环境的影响并且发生改变的类画状态图。而活动图描述满足用例要求所要进行的活动以及活动间的约束关系，有利于识别并行活动。

第四类是交互图 (Interactive diagram)，包括顺序图、协作图。描述对象间的交互关系。其中顺序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互；协作图描述对象间的协作关系，合作图跟顺序图相似，显示对象间的动态合作关系。除显示信息交换外，协作图还显示对象以及它们之间的关系。如果强调时间和顺序，则使用顺序图；如果强调上下级关系，则选择合作图。这两种图合称为交互图。

第五类是实现图 (Implementation diagram)，包括构件图(组件图)、部署图(配置图)。其中构件图描述代码部件的物理结构及各部件之间的依赖关系。一个部件可能是一个资源代码部件、一个二进制部件或一个可执行部件。它包含逻辑类或实现类的有关信息。构件图(组件图)有助于分析和理解部件之间的相互影响程度。部署图(配置图)定义系统中软硬件的物理体系结构。它可以显示实际的计算机和设备（用节点表示）以及它们之间的连接关系，也可显示连接的类型及部件之间的依赖性。在节点内部，放置可执行部件和对象以显示节点跟可执行软件单元的对应关系。

从应用的角度看，当采用面向对象技术设计系统时：

- 1、首先是描述需求；
- 2、其次根据需求建立系统的静态模型，以构造系统的结构；
- 3、第三步是描述系统的行为。

其中在第一步与第二步中所建立的模型都是静态的，包括用例图、类图（包含包）、对象图、构建(组件)图和部署(配置)图等五个图形，是标准建模语言UML的静态建模机制。其中第三步中所建立的模型或者可以执行，或者表示执行时的时序状态或交互关系。它包括状态图、活动图、顺序图和合作(协作)图等四个图形，是标准建模语言UML的动态建模机制。因此，标准建模语言UML的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

## 1、UML包括5类图(9中图形)

1.1、用例图：用例图

1.2、静态图：类图，对象图

1.3、行为图：状态图(对类图的补充)，活动图

1.4、交互图：顺序图，协作图

1.5、实现图：构件图，部署图

## 2、UML分为静态建模机制和动态建模机制

2.1、静态建模机制：用例图，类图，对象图，构件图，部署图

2.2、动态建模机制：状态图，活动图，顺序图，协作图