CHAPTER 3

Software Requirement Analysis

UML 基础教程

UML图的画法

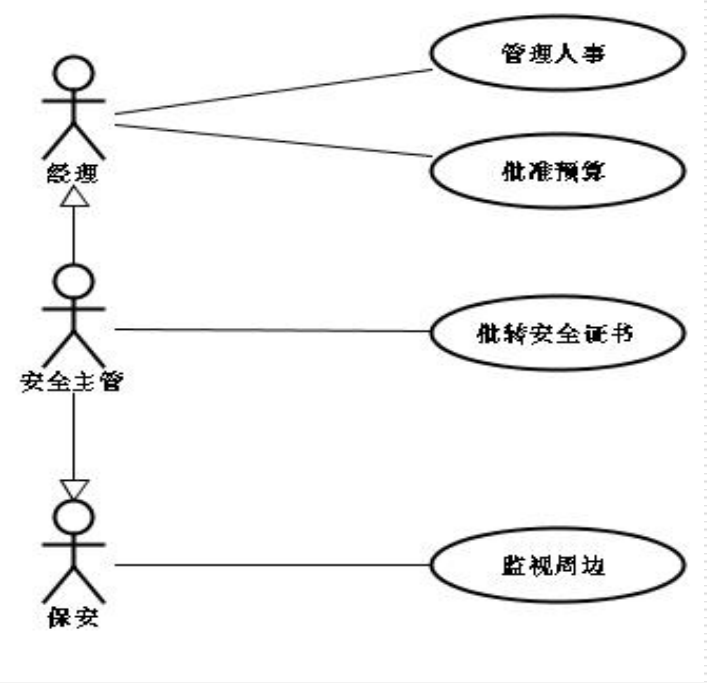
UML语法（图的画法，概要）

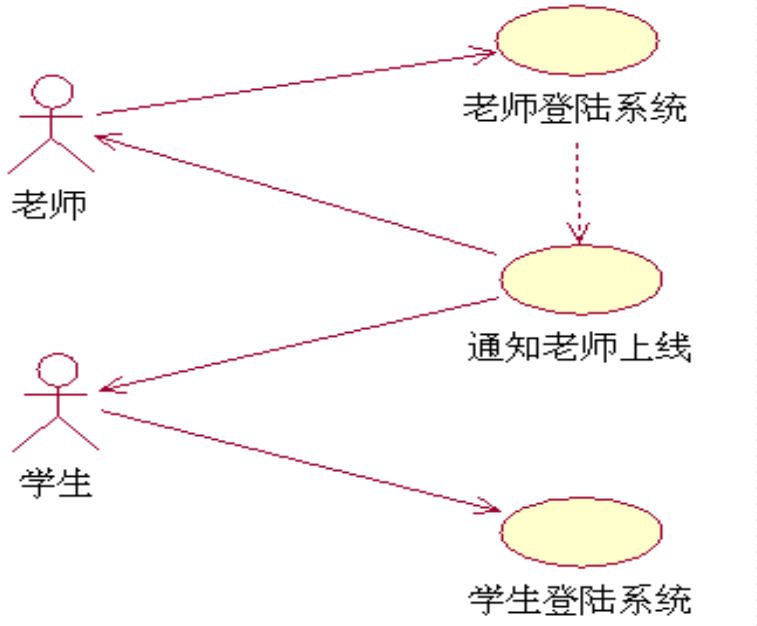
类	是对一组具有相同属性、相同操作、相同关系和相同语义的对象的描述	NewClass
对象		对象名：类名　：类名　类名
接口	是描述了一个类或构件的一个服务的操作集	Interface
协作	定义了一个交互，它是由一组共同工作以提供某种协作行为的角色和其他元素构成的一个群体	
用例	是对一组动作序列的描述	usecase
主动类	对象至少拥有一个进程或线程的类	class suspend() flush()
构件	是系统中物理的、可替代的部件	componet
参与者	在系统外部与系统直接交互的人或事物	actor

节点	是在运行时存在的物理元素	NewProcessor
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	state
包	把元素组织成组的机制	NewPackage
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	

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。
- 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)。





Use Case Diagram

- 用例图中的事物及解释（节点）

名称	解释	UML表示
参与者(Actor)	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)。 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。 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。 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。	
用例(Use Case)	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 。	

Use Case Diagram

用例图中的关系（边）及解释

关系（边）		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源头)	
用例之间的关系	包含	箭头指向的用例为被包含的用例，称为包含用例；箭头出发的用例为基用例。包含用例是必选的，如果缺少包含用例，基用例就不完整；包含用例必须被执行，不需要满足某种条件；其执行并不会改变基用例的行为。	
	扩展	箭头指向的用例为被扩展的用例，称为扩展用例；箭头出发的用例为基用例。扩展用例是可选的，如果缺少扩展用例，不会影响到基用例的完整性；扩展用例在一定条件下才会执行，并且其执行会改变基用例的行为。	
参与者之间的关系	泛化	发出箭头的事物“is a”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。	

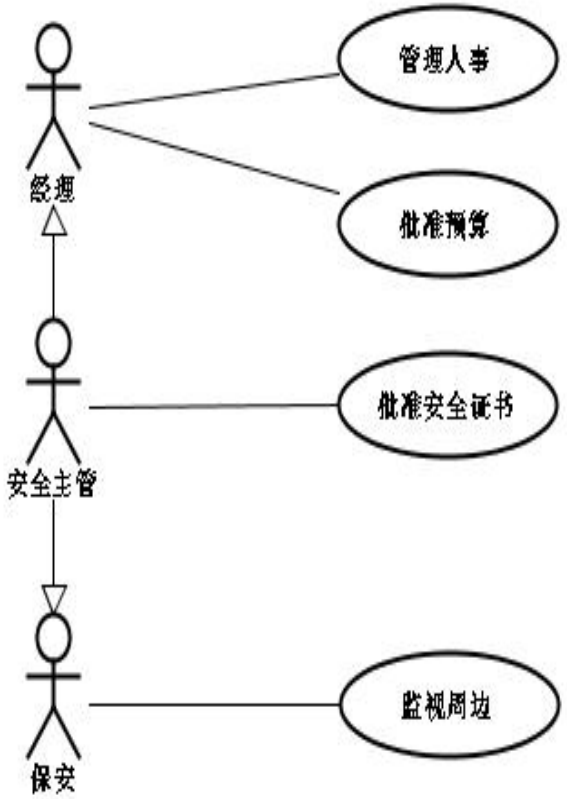
Use Case Diagram

- 实例1 参与者之间的泛化关系

  参与者：经理，安全主管，保安

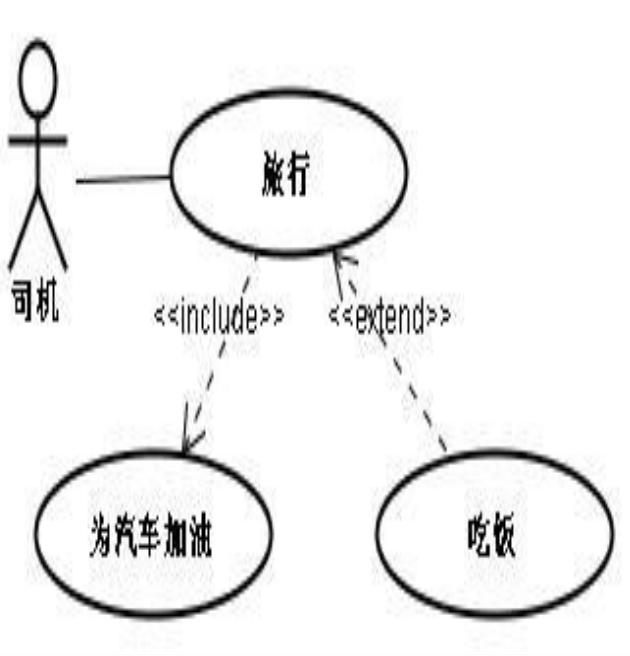
  用例：管理人事，批准预算，批准安全证书，监视周边

  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：经理参与者用例管理人事和批准预算；安全主管参与者用例批准安全证书；保安参与者用例监视周边。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。



Use Case Diagram

- 实例2 用例之间扩展和包含关系
  - ✓ 司机开车旅行时，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。



Use Case Diagram

□ 实例3. 航空售票的用例图

✧**actor**：**clerk，监督员，信用卡服务商，信息亭**

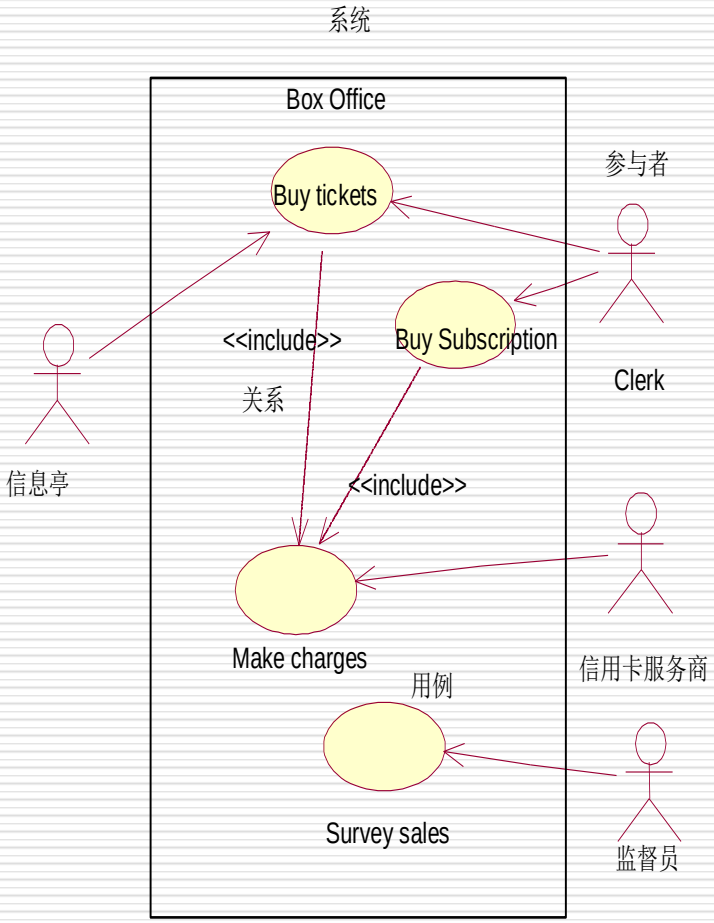
✧**use case**： **Buy tickets， Buy Subscription， Make charges， Survey sales**

✧**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。**

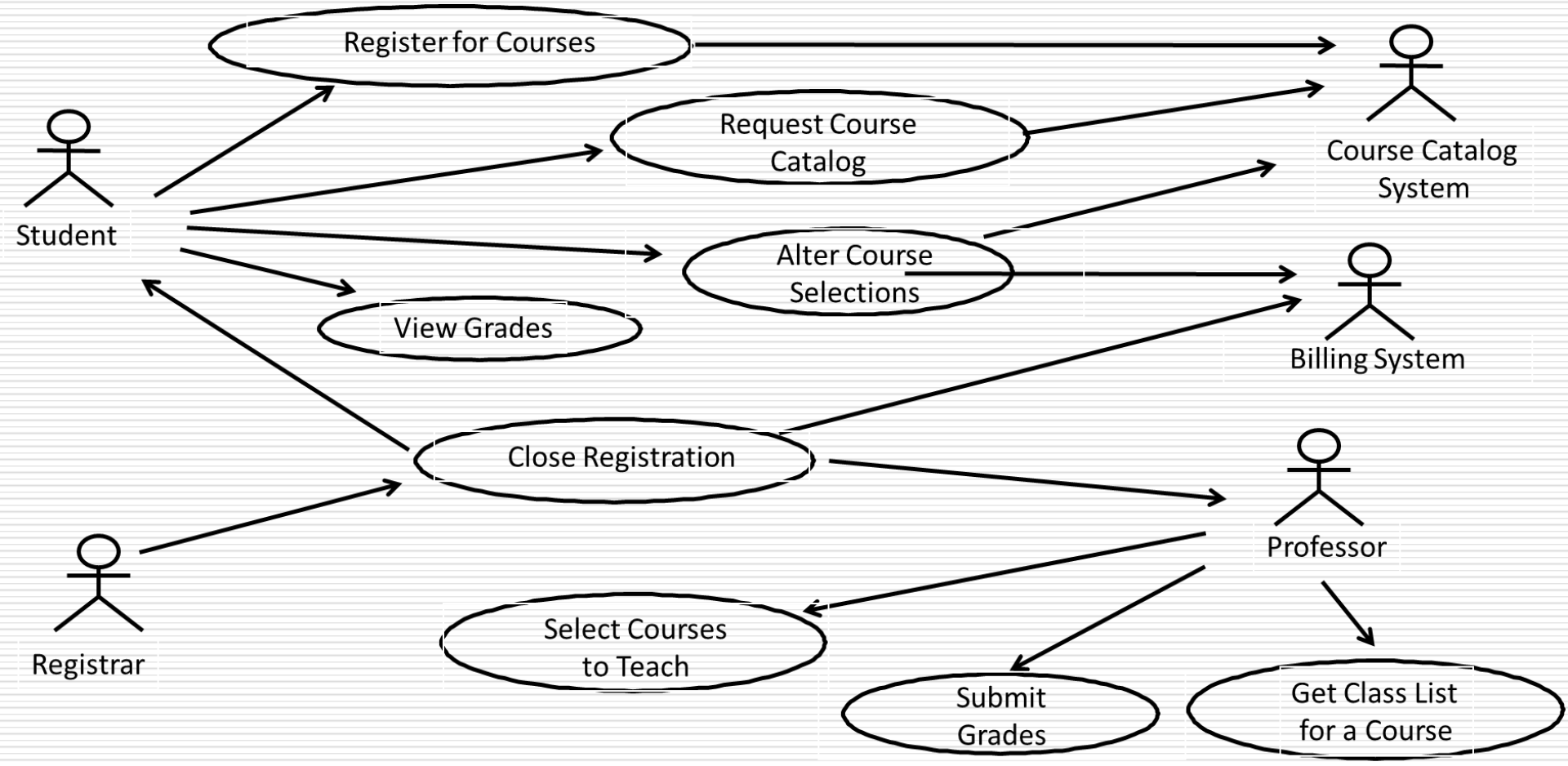
✧**该系统主要包含：Buy tickets， Buy Subscription，Make charges， Survey sales这几个功能。**

✧**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。**



Use Case Diagram

- **Course Registration System**

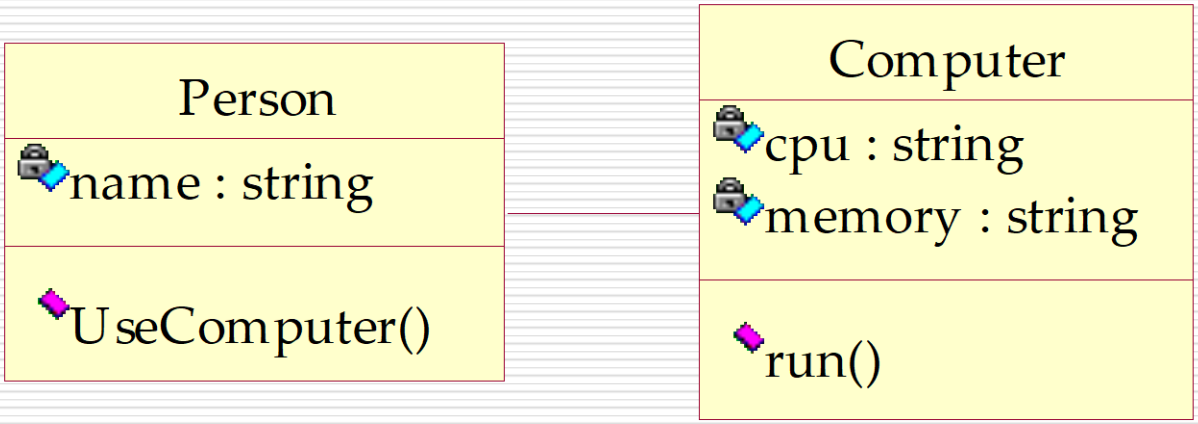


类图(Class Diagram)

1、类图概念

- 类图以反映类的组成(属性、操作)，以及类之间的关系为主要目的，描述了软件系统的静态结构，是一种静态建模方法
- 类图不仅定义系统中的类，还要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。
- 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（对象）的抽象。

Class Diagram

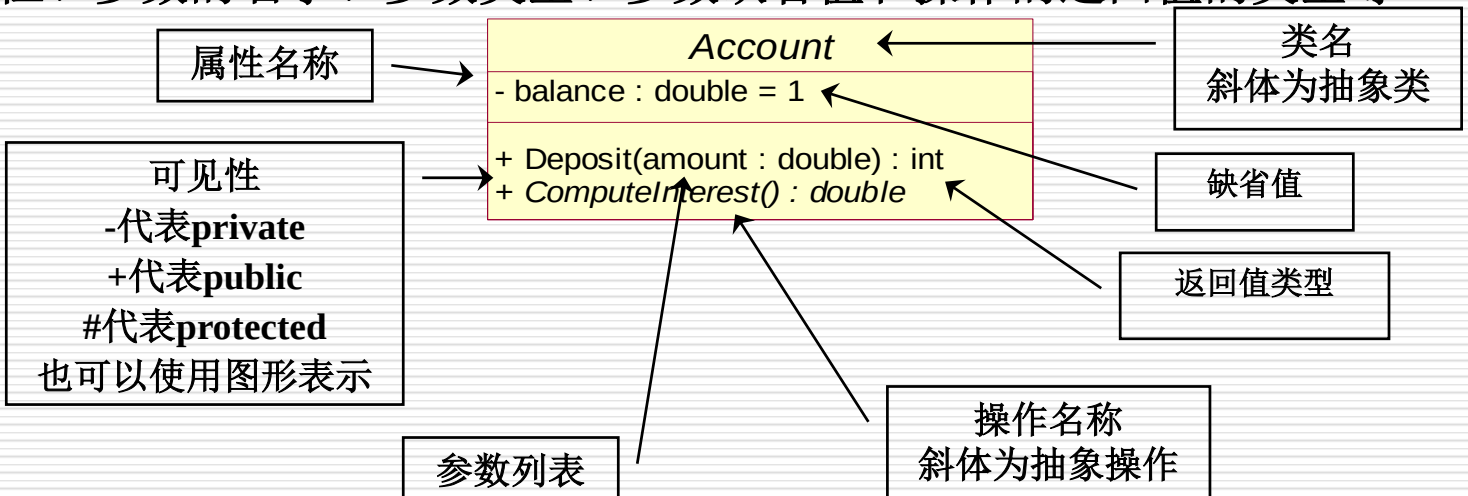


Class Diagram

2、类图中的术语及解释

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



Class Diagram

2、类图中的术语及解释

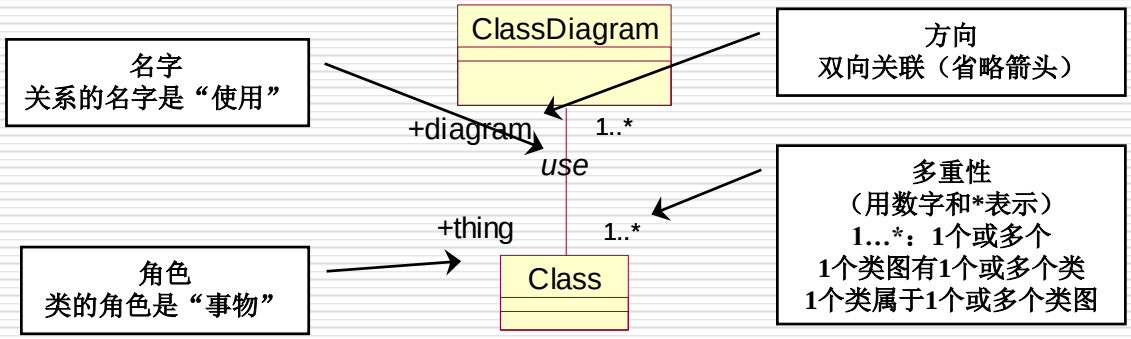
- 接口
  - ※一组操作的集合，只有操作的声明而没有实现
- 抽象类
  - ※不能被实例化的类，一般至少包含一个抽象操作
- 模版类
  - ※一种参数化的类，在编译时把模版参数绑定到不同的数据类型，从而产生不同的类

Class Diagram

3、类图中的关系及解释

（1）关联关系：描述了类的结构之间的关系。具有方向、名字、角色和多重性等信息。

- 一般关联：



实例

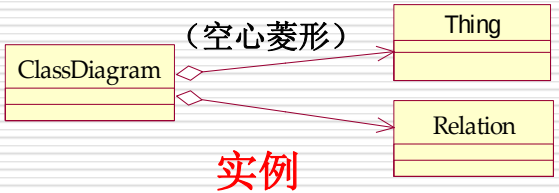
UML表示法

- 聚合关系 Aggregation ：
  - 特殊关联关系，指明一个聚集（整体）和组成部分之间的关系

UML表示法

- 组合关系 Composition
  - 语义更强的聚合，部分和整体具有相同的生命周期

UML表示法



实例

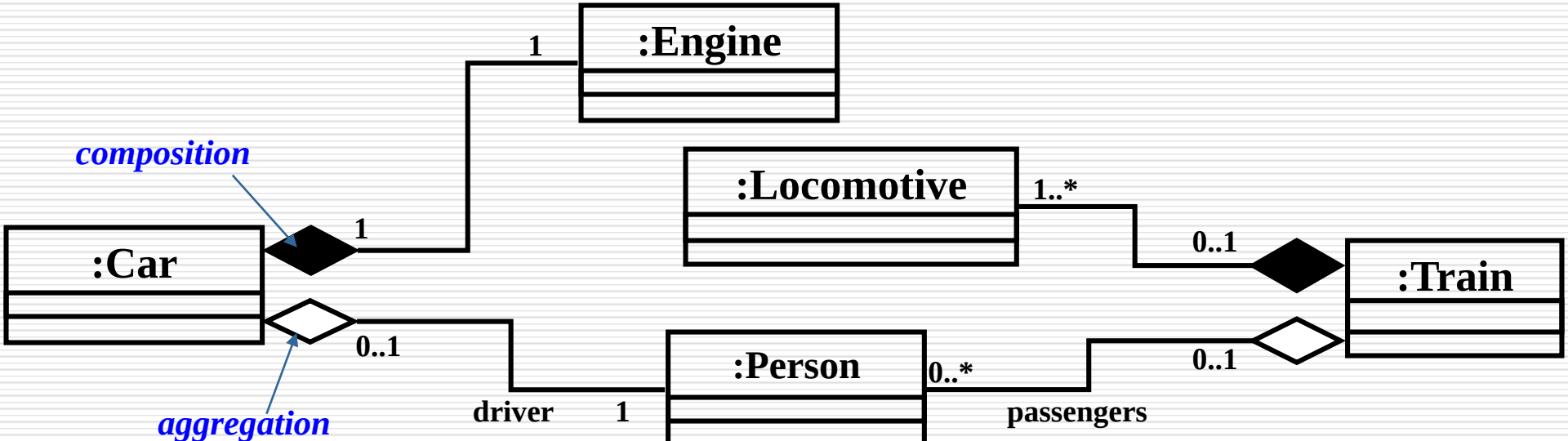
类图包含有事物和关系，类图不存在了，事物和关系还可用于其它的类图

实例

类与关联关系之间有组合关系，类不存在了，则相应的关联关系也不存在

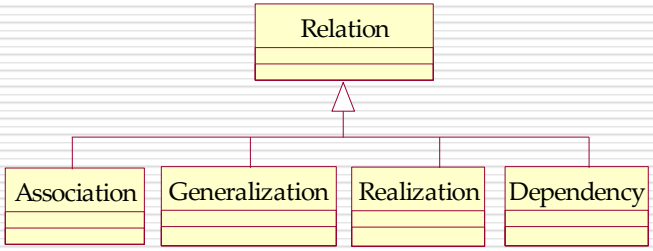
Class Diagram

- Aggregation
  - ✓ This is the "Has-a" or "Whole/part" relationship
- Composition
  - ✓ Strong form of aggregation that implies ownership:
    - ✓ if the whole is removed from the model, so is the part.
    - ✓ the whole is responsible for the disposition of its parts



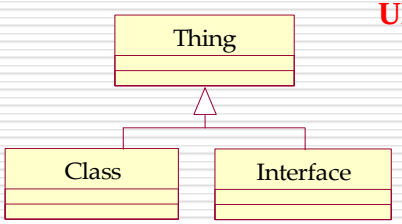
（2） 泛化关系

- ✓ 在面向对象中一般称为继承关系，存在于父类与子类、父接口与子接口之间



关联、泛化、实现、依赖都是一种关系

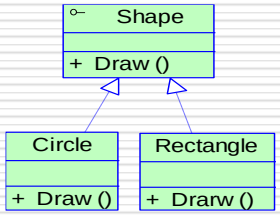
UML表示法



类、接口都是一种事物

（3）实现关系

- ✓ 对应于类和接口之间的关系



类Circle、Rectangle实现了接口Shape的操作

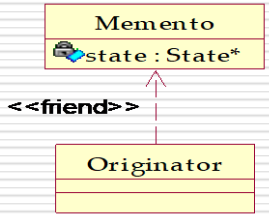
UML表示法

（4） 依赖关系

- ✓ 描述了一个类的变化对依赖于它的类产生影响的情况。
- ✓ 例如绑定(bind)、友元(friend)等

UML表示法

模板类Stack<T>定义了栈相关的操作；IntStack将参数T与实际类型int绑定，使得所有操作都针对int类型的数据

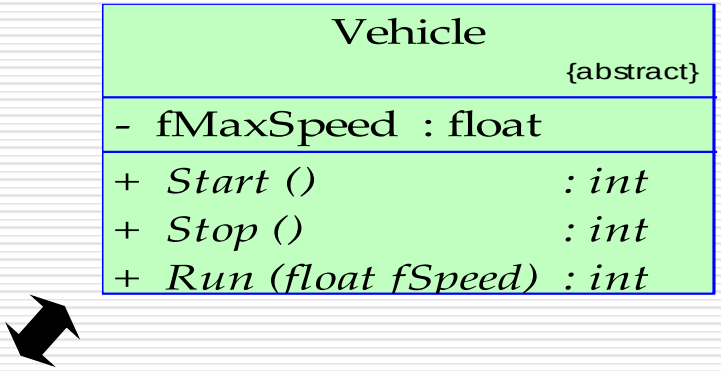


类Memento和类Originator建立了友元依赖关系，以便Originator使用Memento的私有变量state

Class Diagram

4、 类图与代码的映射

（1） 类的映射



C++代码

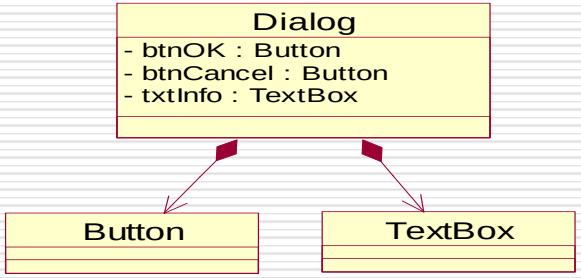
```
class Vehicle
{
   public:
      virtual int Start() = 0;
      virtual int Stop() = 0;
      virtual int Run(float fSpeed) = 0;
   private:
     float fMaxSpeed;
};
```

Java代码

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

Class Diagram

（2） 关联关系的映射

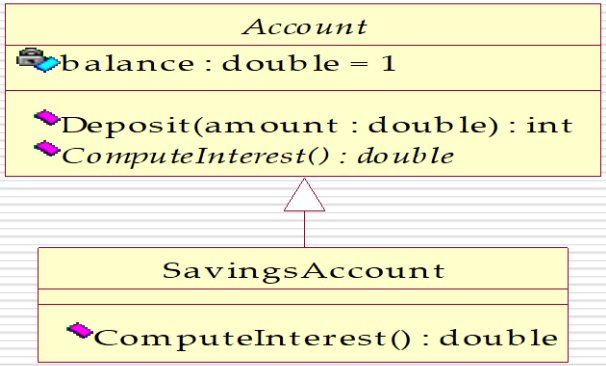


组合关系，代码表现为**Dialog**的属性有**Button**和**TextBox**的对象

C++代码

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

（3） 泛化关系的映射



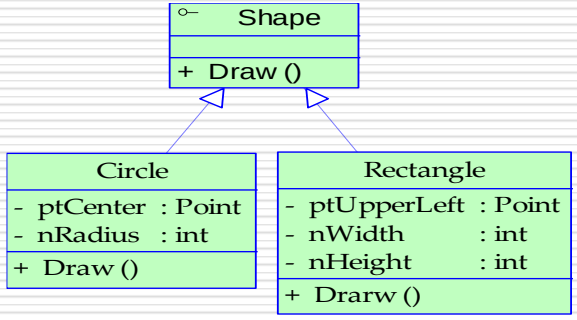
C++代码

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

Java代码

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

（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;
};
```

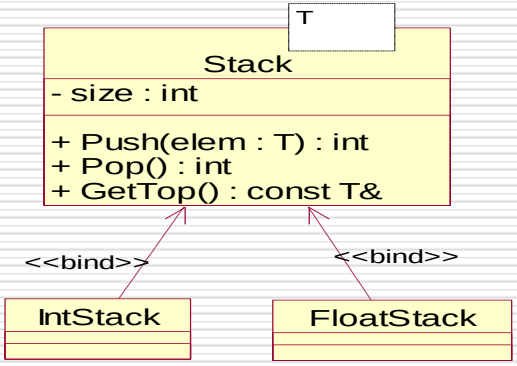
Java代码

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

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

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

（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)

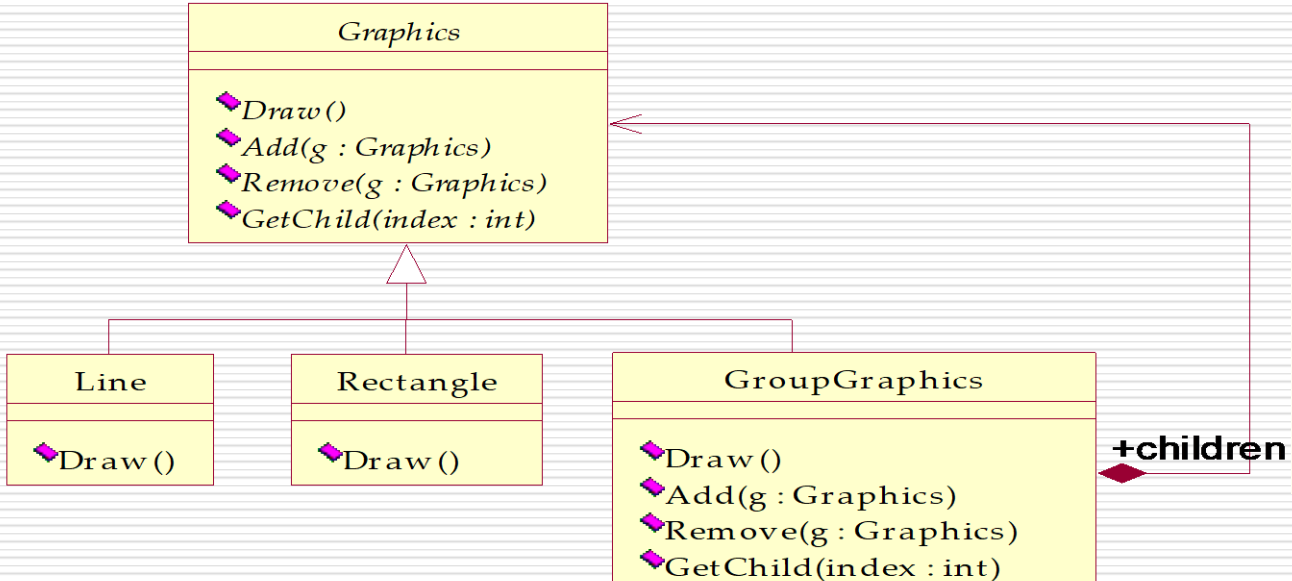
```
class FloatStack
{
private:
  int size;
public:
  int Push(float elem);
  int Pop();
  const float& GetTop();
};
```

Class Diagram

5、类图例子

（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。



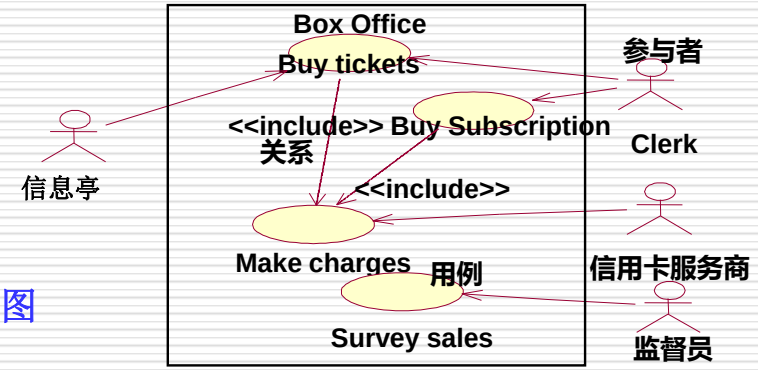
Graphics: 基本图形和组合图形的父类，声明了所有图形共同的操作，如**Draw**；也声明了专用于组合图形管理子图形的操作，如**Add**、**Remove**

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)

（2） 演出售票系统

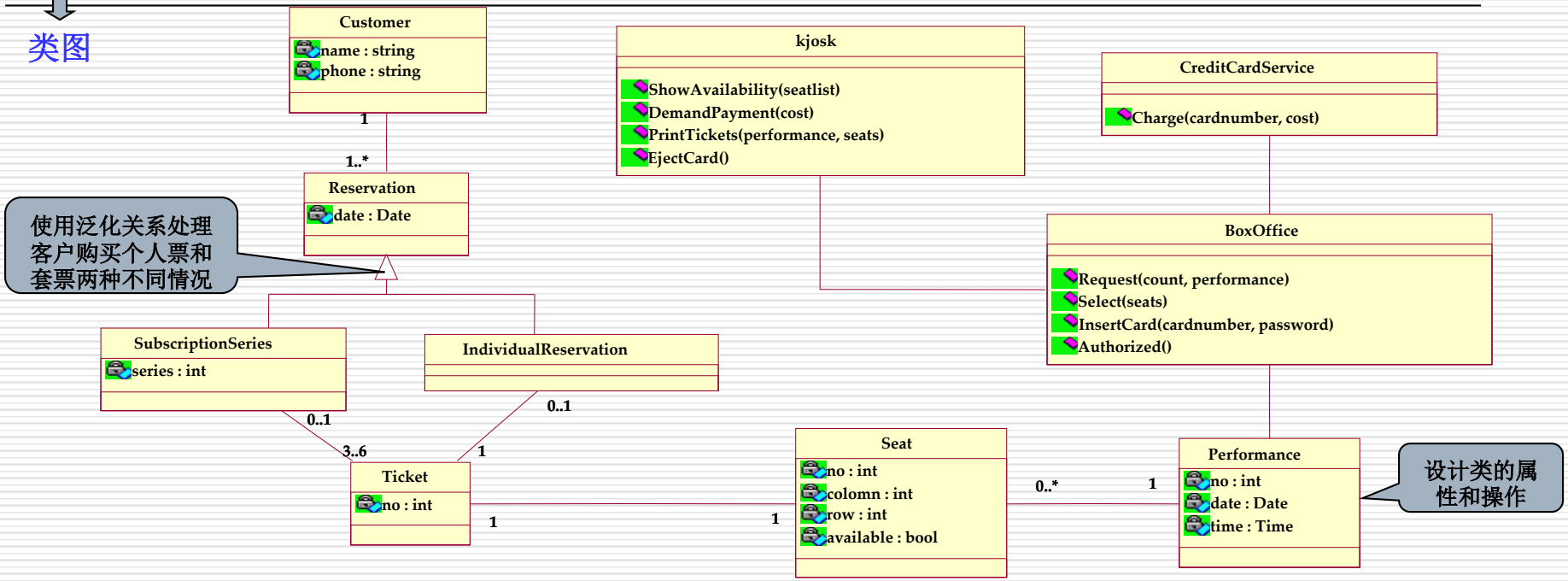
在用例驱动的开发过程中，通过分析各个用例及参与者得到类图。分析用例图的过程中需要根据面向对象的原则设计类和关系，根据用例的细节设计类的属性和操作



在这里只考虑以下三个用例：
Buy tickets: 买个人票
Buy Subscription: 买套票
Make charges: 信用卡付款

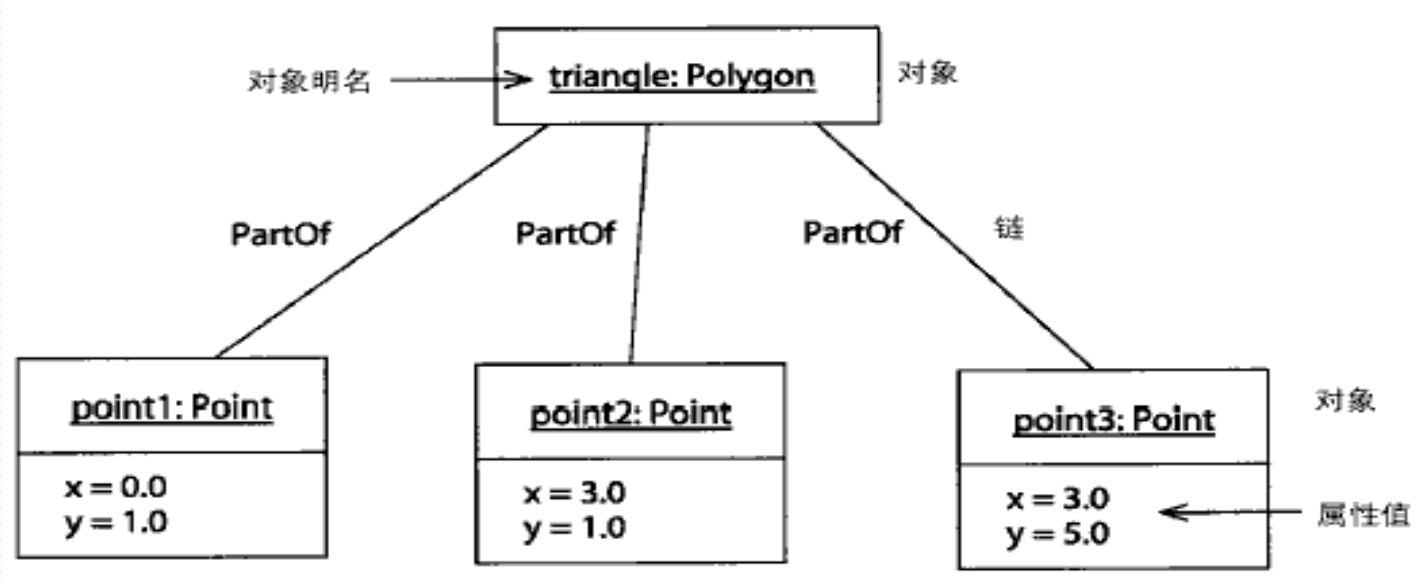
用例图

类图



对象图（ Object Diagram ）

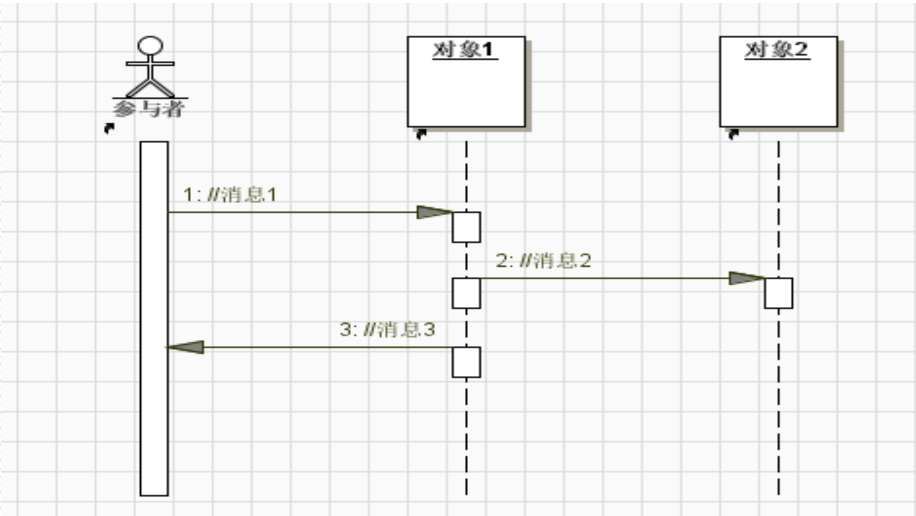
对象图是类图的实例，几乎使用与类图完全相同的标识。它们的不同点在于对象图显示类的多个对象实例，而不是实际的类。

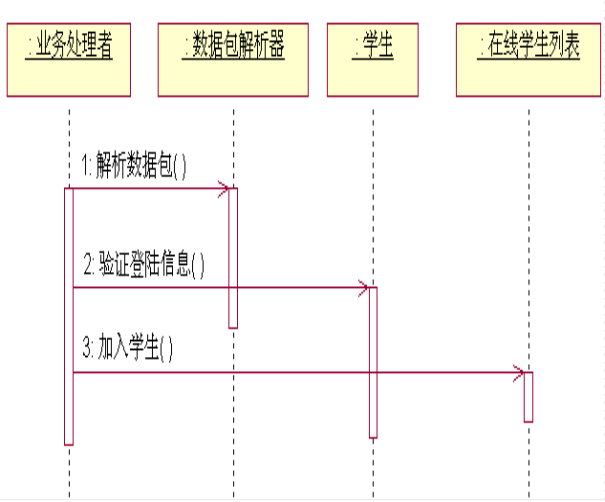


顺序图(Sequence Diagram)

1、概念

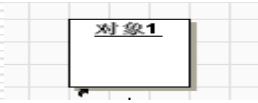
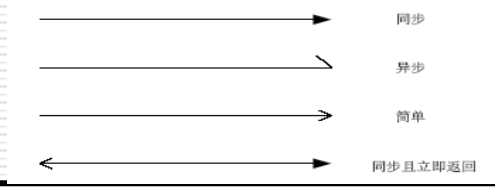
- ✓ 顺序图用来表示用例中的行为顺序。当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的事件。
- ✓ 顺序图展示对象之间的交互，这些交互是指在场景或用例的事件流中发生的。顺序图属于动态建模。
- ✓ 顺序图的重点在消息序列上，也就是说，描述消息是如何在对象间发送和接收的。表示了对象之间传送消息的时间顺序。
- ✓ 浏览顺序图的方法是：从上到下查看对象间交换的消息。





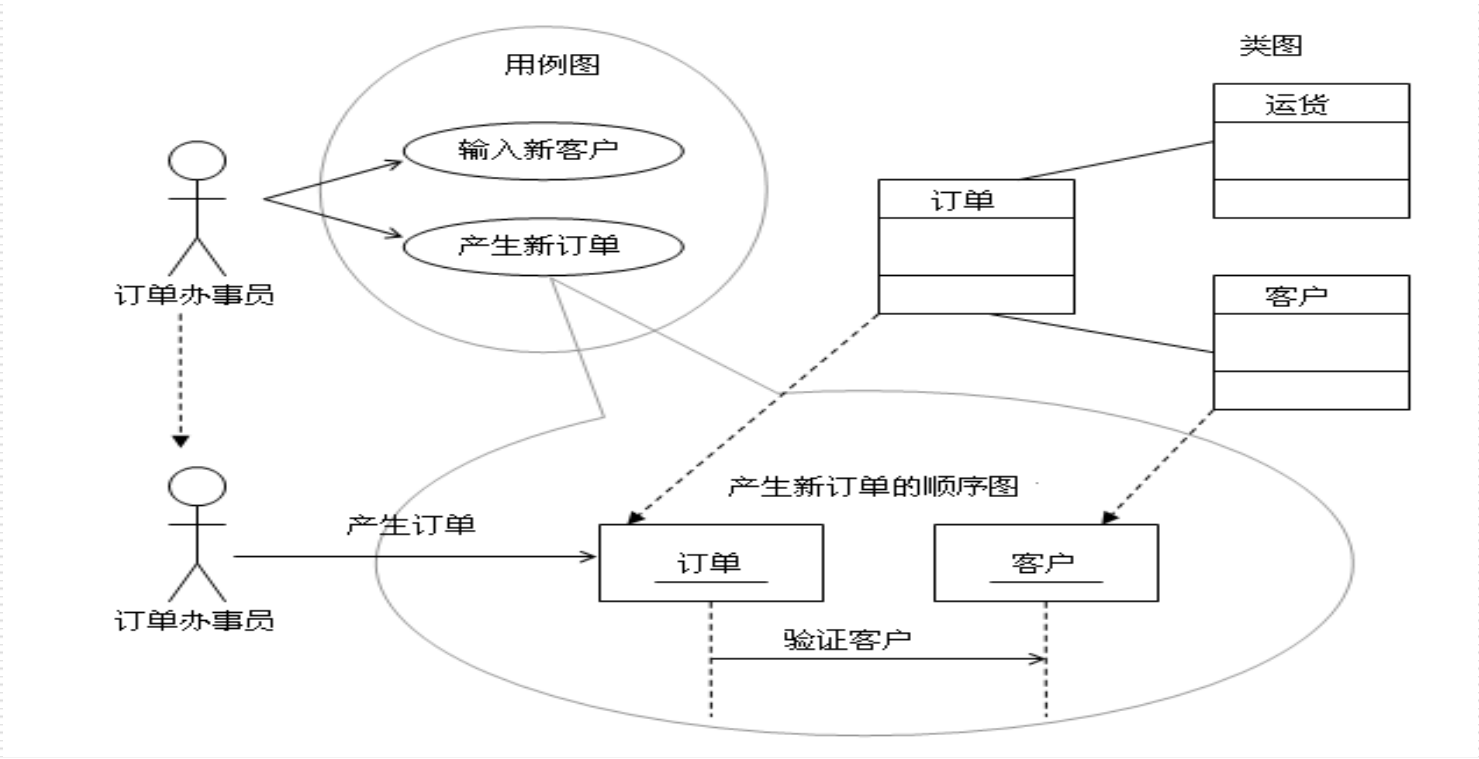
Sequence Diagram

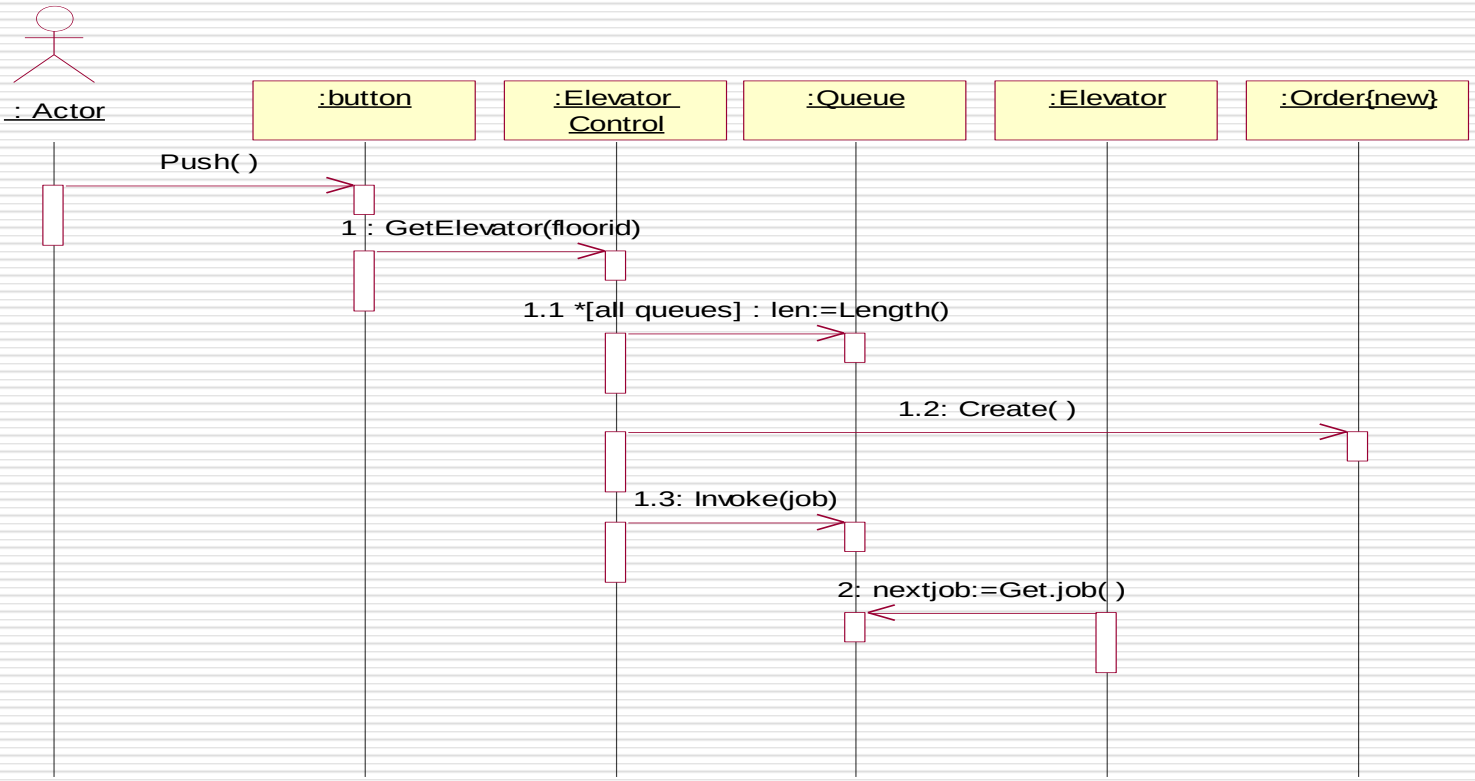
2、 顺序图中的术语及解释

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。	

Sequence Diagram

3、 顺序图与用例图和类图的关系





Rational Rose 工具生成

Sequence Diagram

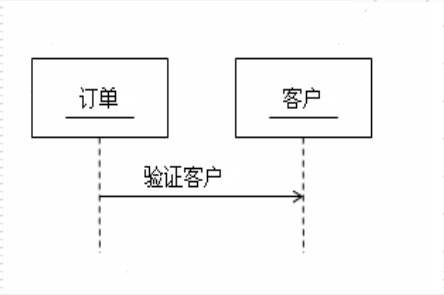
4、 顺序图例子

消息格式:

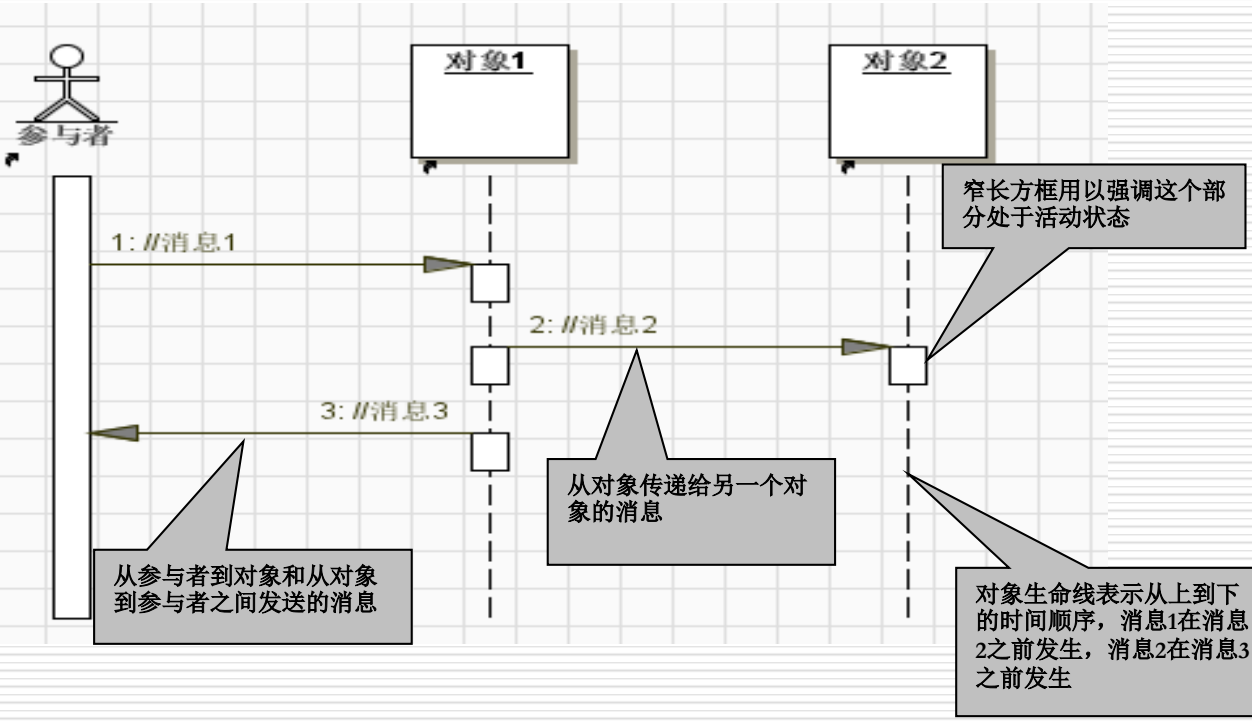
operation (parameter list)

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。

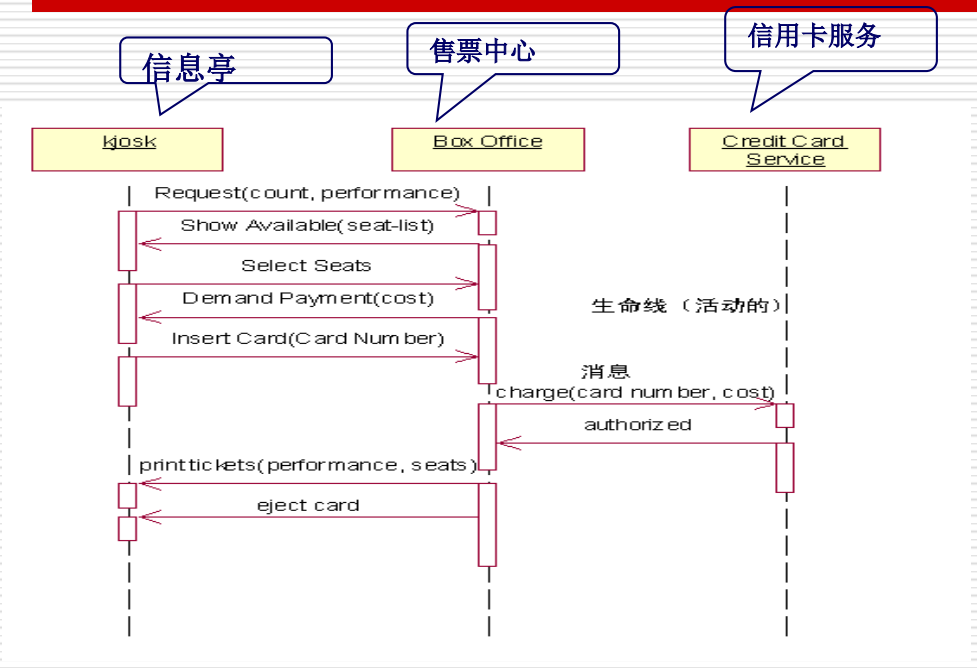
例：



订单类发消息给客户类调用客户类中的“验证客户”操作



Sequence Diagram



从这个例子中可以看出：

Kjosk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。

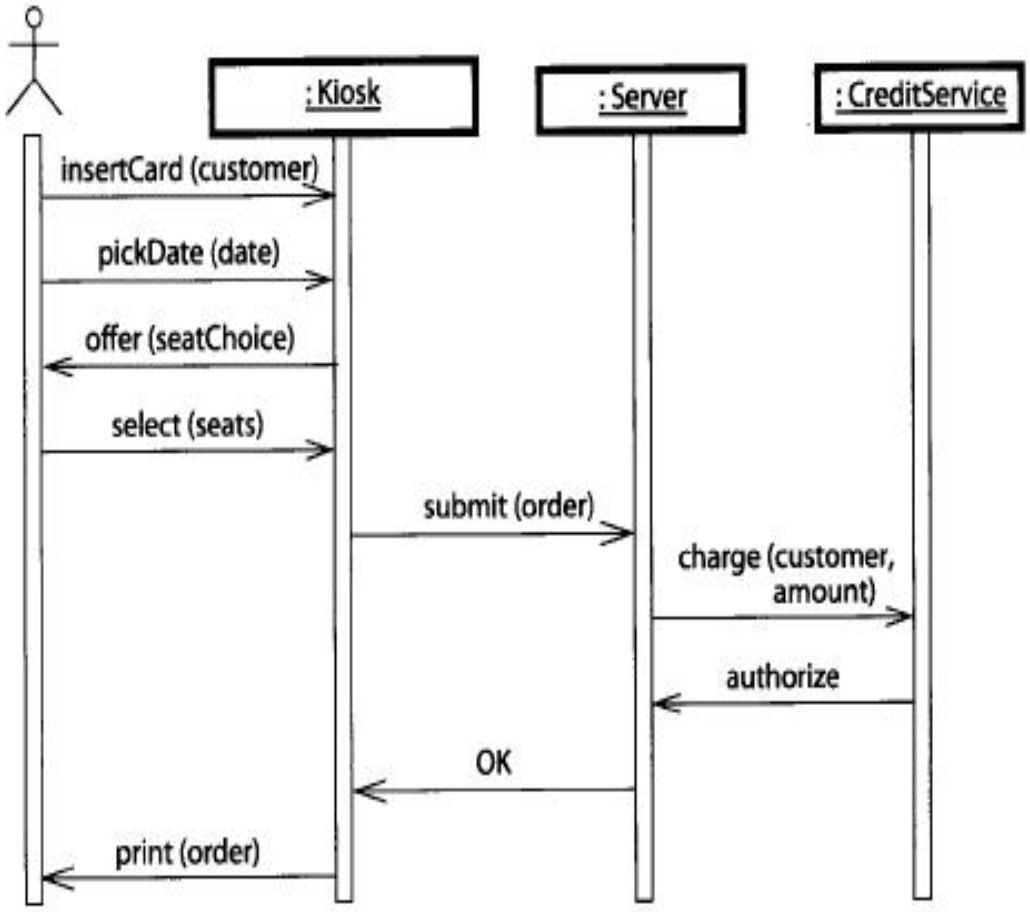
此图中存在的事物有：

对象(信息亭 **Kjosk** ，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。

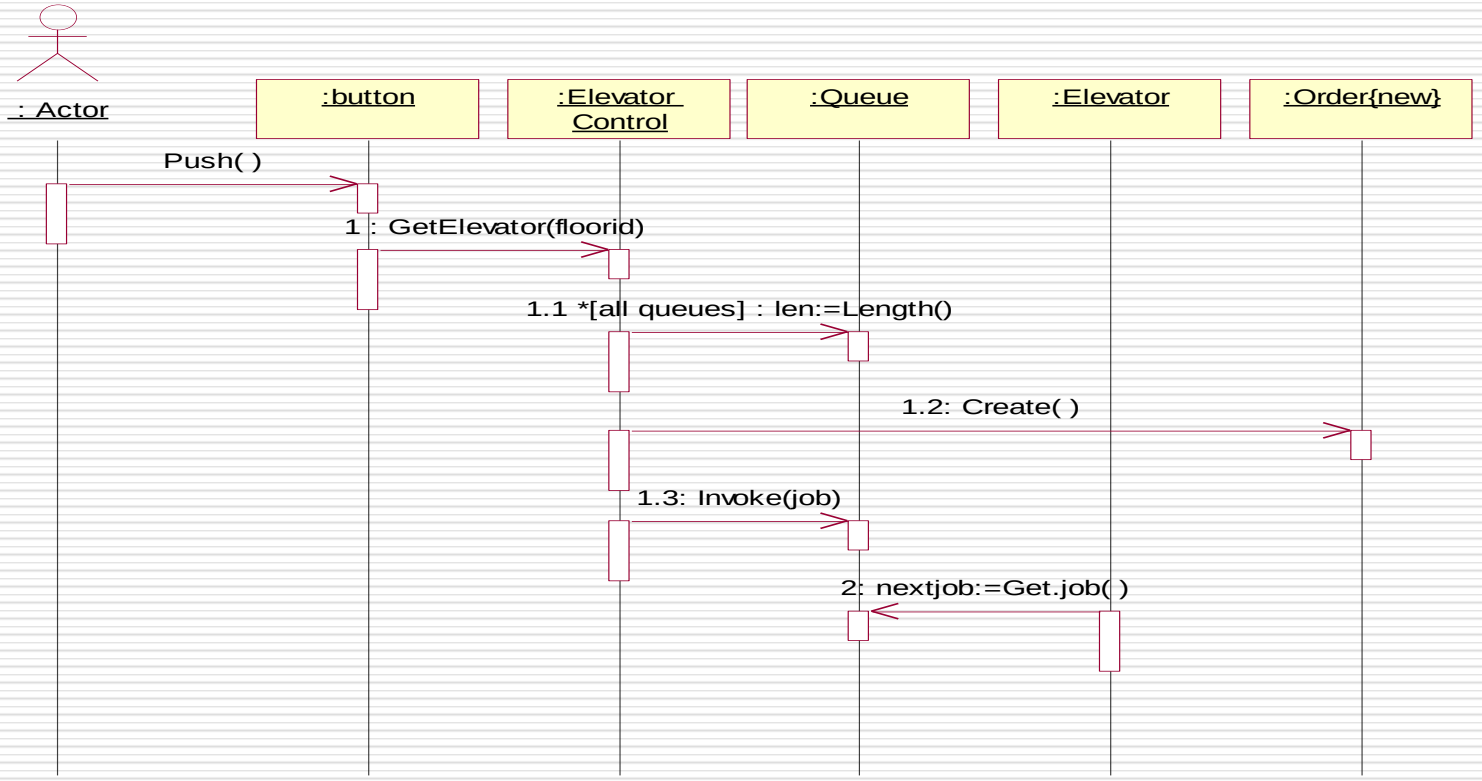
售票中心发**Show Available(seat-list)**消息给信息亭，表示调用信息亭类中的**Show Available(seat-list)**操作，给出可用的座位表。

Sequence Diagram



购票用例的顺序图

Sequence Diagram



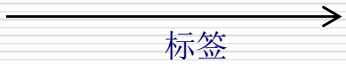
乘坐电梯

协作图（collaboration diagram）

1、概要

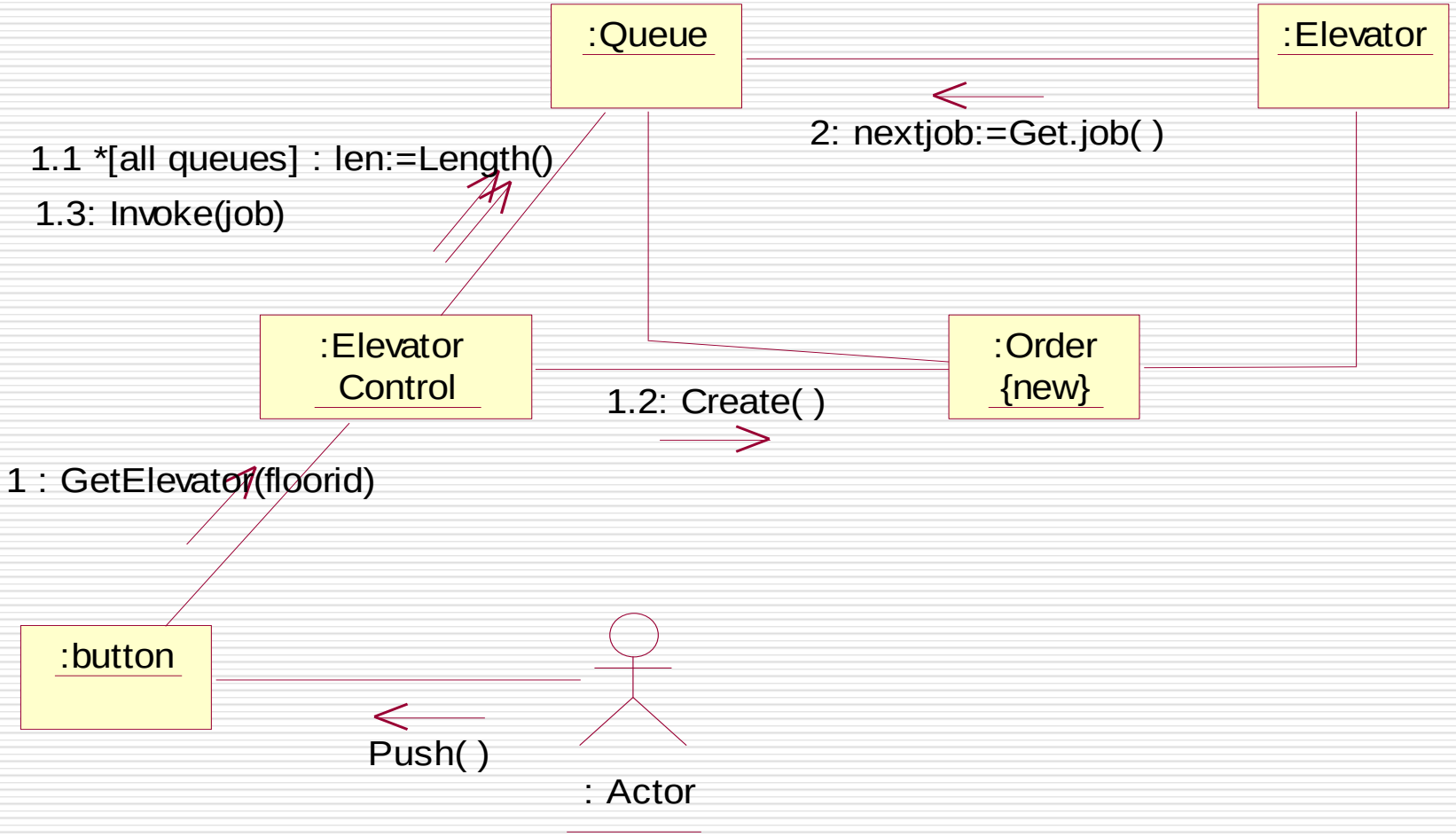
- 协作图是一种交互图，强调的是发送和接收消息的对象之间的组织结构，使用协作图来说明系统的动态情况。
- 协作图主要描述协作对象间的交互和链接，显示对象、对象间的链接以及对象间如何发送消息。
- 协作图可以表示类操作的实现。

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。	
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。	
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。	

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	


:Queue
:Elevator
2: nextjob:=Get.job( )
1.1 *[all queues] : len:=Length()
1.3: Invoke(job)
:Elevator Control
:Order {new}
1.2: Create( )
1 : GetElevator(floorid)
:button
Push( )
: Actor

乘坐电梯活动图

协作图（collaboration diagram）

4、 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。

状态图（Statechart Diagram）

1、状态图概念

- ✓ 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。
- ✓ 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。

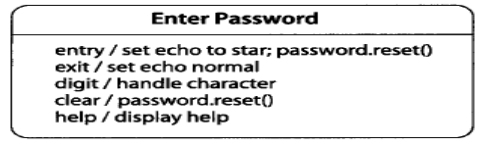
2、状态图的组成

- ✓ 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。
- ✓ 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。

Statechart Diagram

3、状态图中的术语及解释

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)	
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	
开始	初始状态(一个)	
结束	终态(可以多个)	

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	e(a:T)[exp]/action

Statechart Diagram

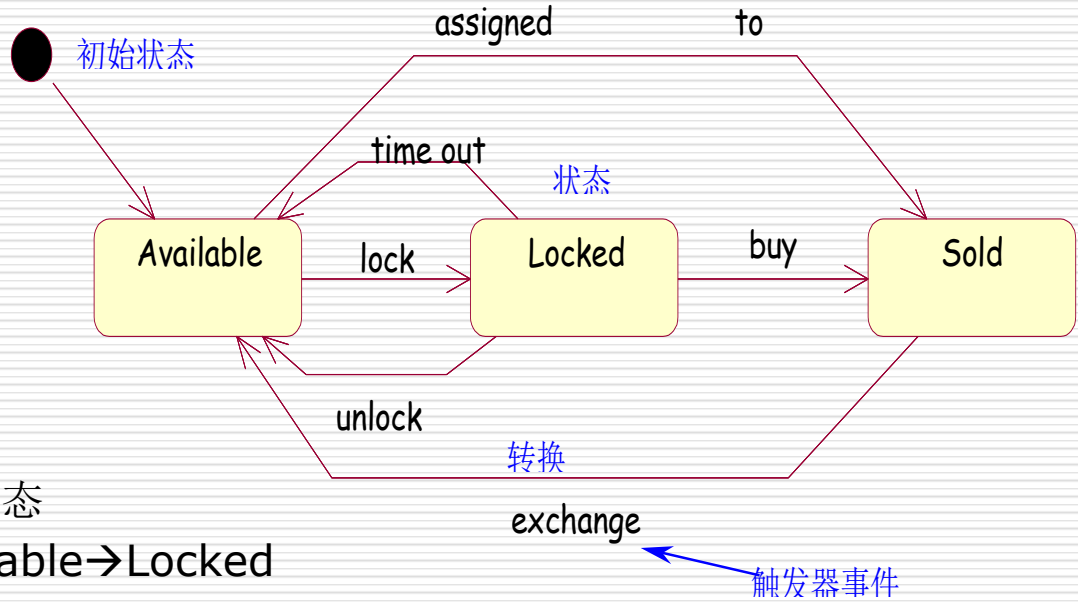
5、举例

图中包含以下状态

- ✓ 初始状态
- ✓ Available状态
- ✓ Locked状态
- ✓ Sold状态

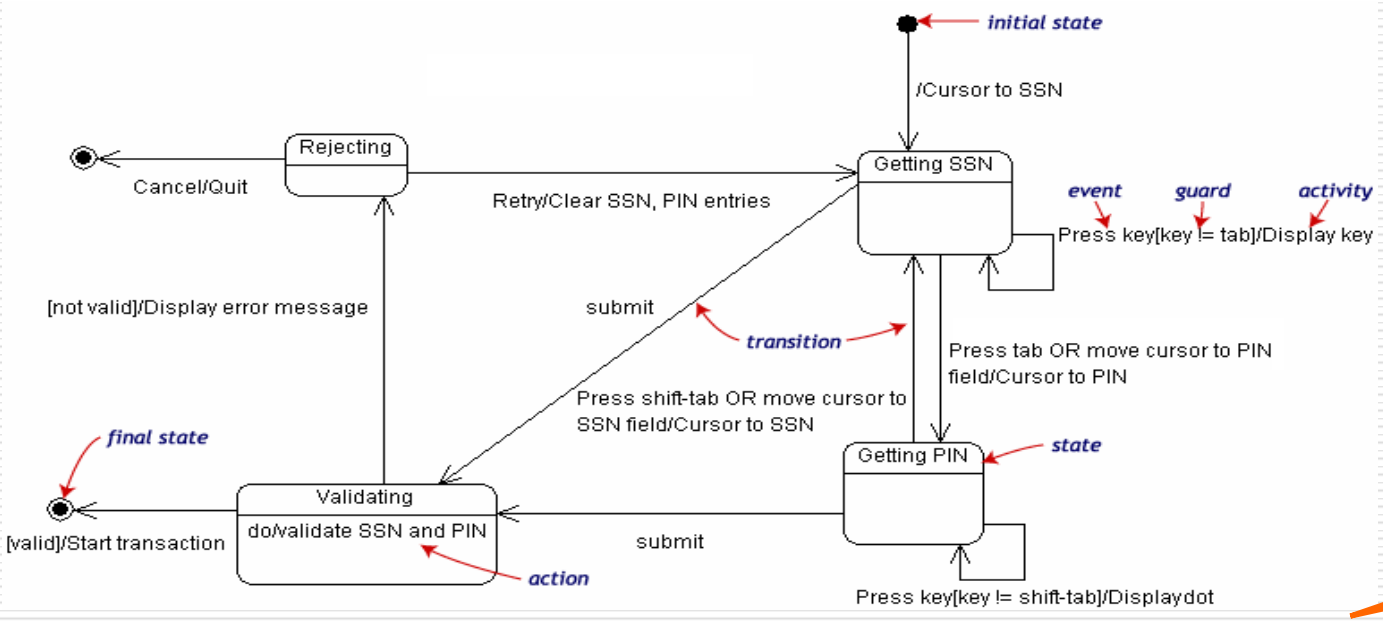
状态间的转移

- ✓ 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



（1）订票机的状态图

(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

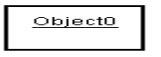
有两个不同的终态

活动图（Activity Diagram）

1、活动图概念

- ✓ 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。

2、活动图事物

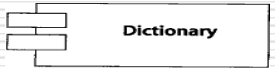
活动 (ActionState)	动作的执行	
起点 (InitialState)	活动图的开始	
终点(FinalState)	活动图的终点	
对象流(ObjectFlowState)	活动之间的交换的信息	
发送信号(signalSending)	活动过程中发送事件，触发另一活动流程	
接收信号(SignalReceipt)	活动过程中接收事件，接收到信号的活动流程开始执行	
泳道(SwimLane)	活动的负责者	

构件图（component diagram）

1、 构件图概念

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。

2、 构件图中的术语及解释

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)。	

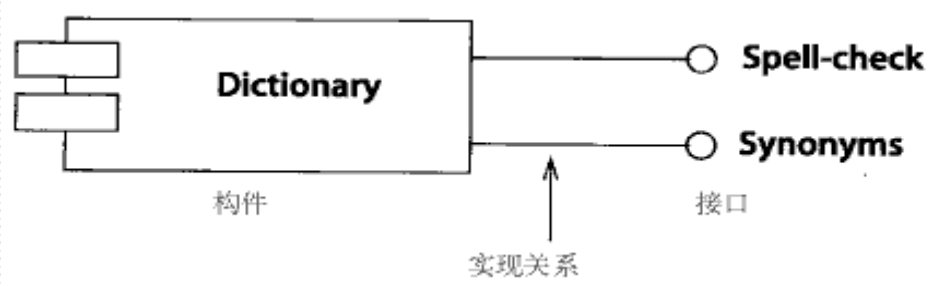
可替换的物理部分包括软件代码、脚本或命令行文件，也可以表示运行时的对象，文档，数据库等。
节点(node)是运行时的物理对象，代表一个计算机资源。具体请参见教程“部署图(deployment diagram)”部分。

构件图（component diagram）

实例1：

图中的构件名称是Dictionary字典。

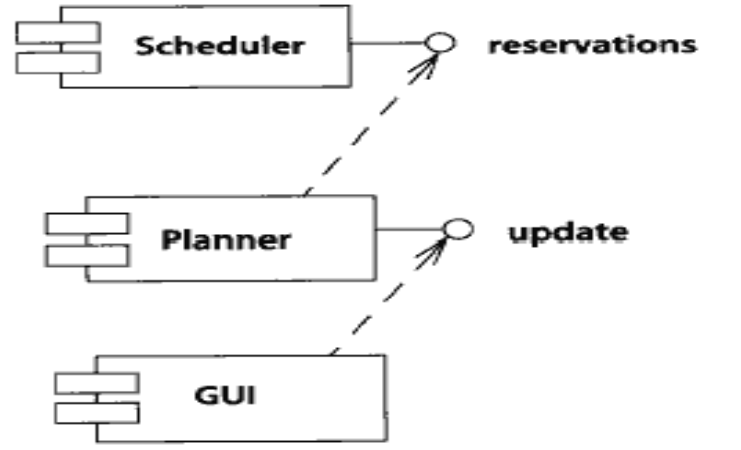
该构件向外提供两个接口，即两个服务Spell-check拼写检查、Synonyms同义词。



实例2.

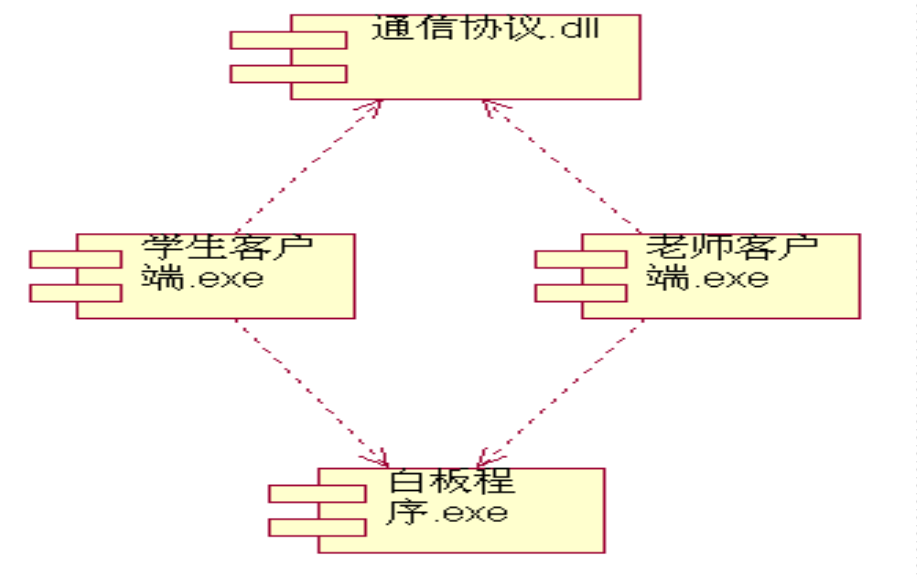
图中“Planner计划者”构件向外提供一个“update更新”接口服务。

同时，该构件要求外部接口提供一个“Reservations预定”服务。



构件图（component diagram）

组件图显示软件组件之间的依赖关系。一般来说，软件组件就是一个实际文件，可以是源代码文件、二进制代码文件和可执行文件等。可以用来显示编译、链接或执行时构件之间的依赖关系。



在线答疑系统组件图

部署图（deployment diagram）

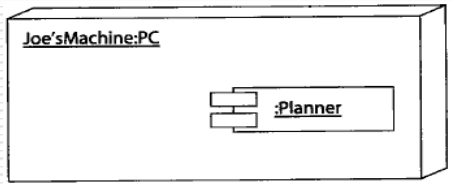
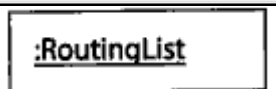
1、部署图概念

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。

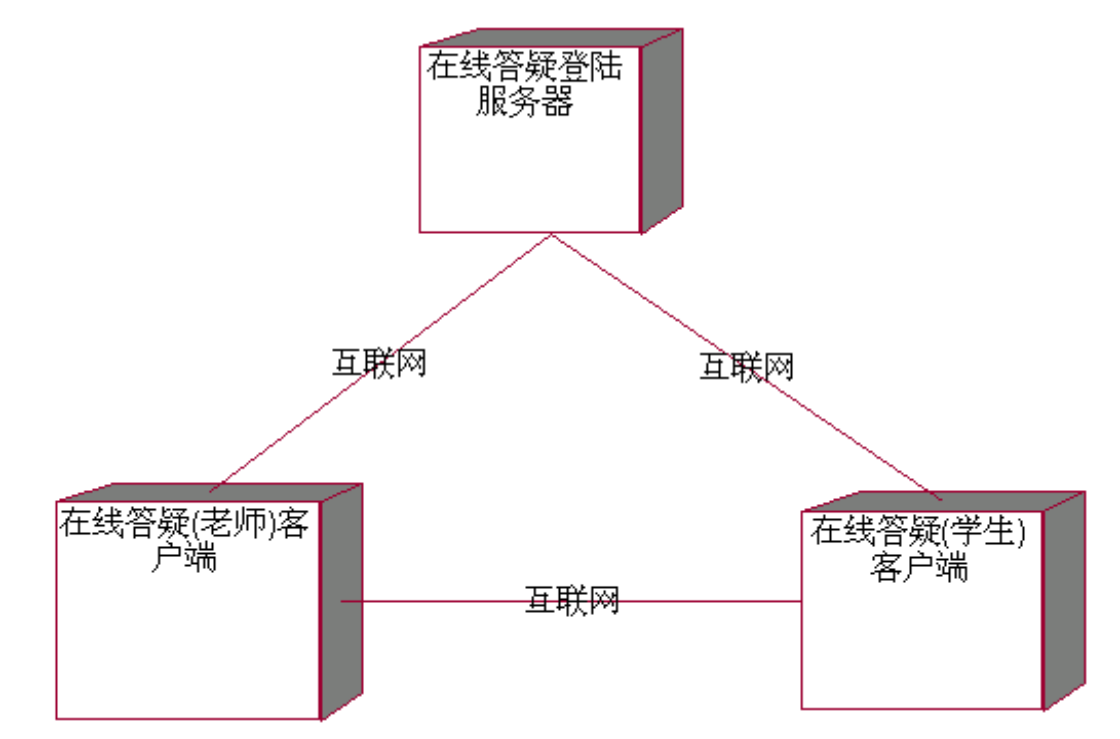
部署图的两种表现形式：实例层部署图和描述层部署图(会在后面的实例中给出)。

2、部署图中的术语及解释

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。	

部署图（deployment diagram）

配置图显示系统运行时刻的结构，显示系统不同的组件在何处物理地运行，以及它们将如何彼此通信。



在线答疑系统部署图

UML语法（图的画法，概要）

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	NewClass
对象		对象名：类名　：类名　类名
接口	是描述了一个类或构件的一个服务的操作集	Interface
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	usecase
主动类	对象至少拥有一个进程或线程的类	class suspend() flush()
构件	是系统中物理的、可替代的部件	componet
参与者	在系统外部与系统直接交互的人或事物	actor

节点	是在运行时存在的物理元素	NewProcessor
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	state
包	把元素组织成组的机制	NewPackage
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	