1.6UML的图

1.6.1用例图

用例图所包含的元素如下：

**1. 参与者(Actor)**

　　表示与您的应用程序或系统进行交互的用户、组织或外部系统。用一个小人表示。



**2. 用例(Use Case)**

 　　用例就是外部可见的系统功能，对系统提供的服务进行描述。用椭圆表示。



**3. 子系统(Subsystem)**

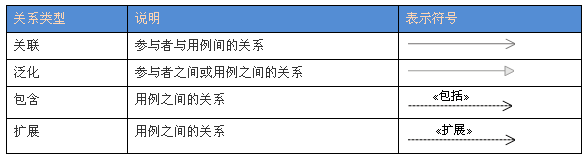
　　用来展示系统的一部分功能，这部分功能联系紧密。



**4. 关系**

　　用例图中涉及的关系有：关联、泛化、包含、扩展。

　　如下表所示：



**a. 关联(Association)**

　　表示参与者与用例之间的通信，任何一方都可发送或接受消息。

　　【箭头指向】：指向消息接收方



**b. 泛化(Inheritance)**

　　就是通常理解的继承关系，子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。

　　【箭头指向】：指向父用例



**c. 包含(Include)**

包含关系用来把一个较复杂用例所表示的功能分解成较小的步骤。

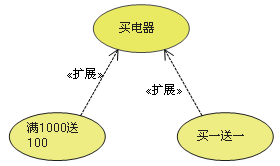
　　【箭头指向】：指向分解出来的功能用例



**d. 扩展(Extend)**

　　扩展关系是指用例功能的延伸，相当于为基础用例提供一个附加功能。

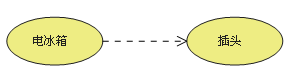
　　【箭头指向】：指向基础用例



**e. 依赖(Dependency)**

　　以上4种关系，是UML定义的标准关系。但VS2010的用例模型图中，添加了依赖关系，用带箭头的虚线表示，表示源用例依赖于目标用例。

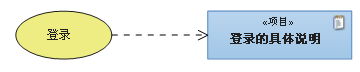
　　【箭头指向】：指向被依赖项

****

**5. 项目(Artifact)**

　　用例图虽然是用来帮助人们形象地理解功能需求，但却没多少人能够通看懂它。很多时候跟用户交流甚至用Excel都比用例图强，VS2010中引入了“项目”这样一个元素，以便让开发人员能够在用例图中链接一个普通文档。

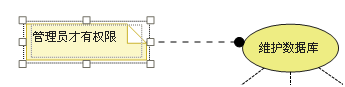
　　用依赖关系把某个用例依赖到项目上：



　　然后把项目-》属性 的Hyperlink设置到你的文档上；

　　这样当你在用例图上双击项目时，就会打开相关联的文档。

**6. 注释(Comment)**



**包含(include)、扩展(extend)、泛化(Inheritance) 的区别：**

　　条件性：泛化中的子用例和include中的被包含的用例会无条件发生，而extend中的延伸用例的发生是有条件的；

　　直接性：泛化中的子用例和extend中的延伸用例为参与者提供直接服务，而include中被包含的用例为参与者提供间接服务。

　　对extend而言，延伸用例并不包含基础用例的内容，基础用例也不包含延伸用例的内容。

　　对Inheritance而言，子用例包含基础用例的所有内容及其和其他用例或参与者之间的关系；

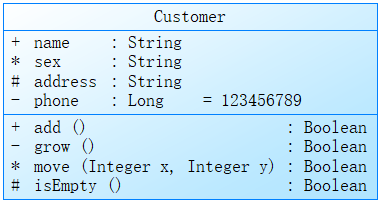
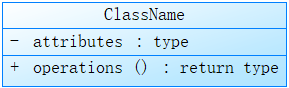
**一个用例图示例：**



1.6.2类图

## 一、类表示法

类图标由**三个部分**组成：第一个部分是**类名**，第二个部分是**属性**，第三个部分是**操作**。



**类名**在它的命名空间中唯一。类名以**大写字母开头**，省略多个单词之间的空格。

**属性和操作**在类的范围内必须无二义。属性和操作是**以小写字母开头**，**后续单词的首字母大写**，且同样省略空格。

**抽象类**和**抽象操作**用**斜体**表示。

* **属性**规格说明格式：

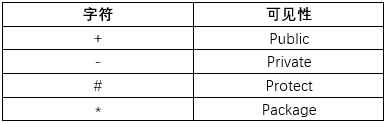
可见性 属性名称：类型 [多重性] = 默认值 {特性字符串}

* **操作**规格说明格式：

可见性 操作名称（参数名称：类型）：返回值 {特性字符串}

## 二、可见性

* **公有**可见性（+）：对能看到这个类的任何元素都可见。
* **保护**可见性（#）：对这个类及其子类的其他元素可见。
* **私有**可见性（-）：对这个类的其他元素可见。
* **包**可见性（~）：对同一个包中的其他元素可见。



## 三、类关系

类的基本联系包括**关联、泛化、聚合和组合**。

### 1. 关联

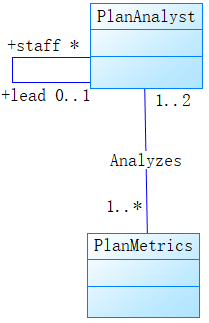
关联用**不带箭头的实线**表示。



关联连接了两个类，体现了一种**语义关系**。

关联通常用**名词词组**来标注，如下图中的Analyzes，以说明关系的实质。

类可能与它自己有关联（称为**自关联**），如PlanAnalyst类的实例之间的协作。注意，这里同时使用了**关联端名称**和**关联名称**，目的是提供清晰性。



关联可以进一步通过**多重性**来修饰（没有也可以）。多重性的语法如下：

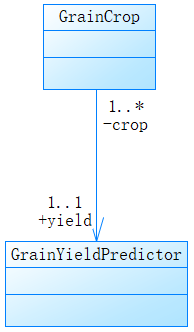
* 精确到1个。
* 数目不限（0个或多个）。
* 0..\*：0个或多个。
* 1..\*：1个多多个。
* 0..1：0个或1个。
* 3..7：指定范围（3~7个，包含3和7）

多重性应用于关联的目标端，说明**源类的每个实例与目标类实例的连接个数**。除非显式说明，否则关系的多重性就是未指定的。

### 高级概念：关联的方向性。

在分析时，我们认为关联是分析类之间的双向逻辑连接。在设计时，我们将关注的焦点转到关联的导航性上。

从GrainCrop类到GrainYieldPredictor类的单向关联通常意味着GrainCrop类的某些方法在实现时使用了GrainYieldPredictor类的服务。注意，GrainCrop类和GrainYieldPredictor类之间的关联端名称的可见性，GrainCrop对GrainYieldPredictor类似私有的。



### 2. 泛化

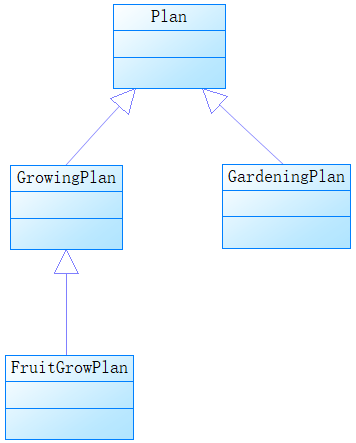
泛化描述的“**是一种**”的关系。

泛化用**带有封闭箭头的实线**表示。**箭头指向超类，关联的另一端是子类**。

子类继承超类的结构和行为。根据这些规则，一个类可以有一个（单继承）或多个（多继承）超类，超类间的名字冲突也可以根据所选语言的规则来处理。

泛化关系**不能有多重性指定**。

下图中的GrowingPlan类是超类，它的子类是FruitGrowingPlan。



### 3. 聚合

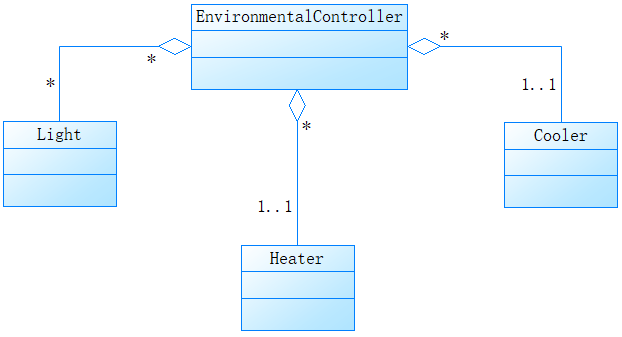
聚合表明一种**整体-部分**的层次结构。

聚合用**带有一个空心菱形的实线**表示。**菱形所在的一端是聚合体（整体），另一端的类代表它的实例构成了聚合对象的部分**。

自聚合和循环聚合关系是可能的。这种整体-部分的层次关系**并不意味着物理上的包容**：一个专业协会有一些成员，但不表示协会拥有它的成员。就如汽车和轮胎，当汽车销毁的时候，并不意味着轮胎也销毁了。即**两个对象的生命周期是相互独立的**。

聚合关系末端的\*（0个或多个）**多重性**进一步突出了这不是物理包容关系。

下图中，EnvironmentalController类有Light、Heater和Cooler类作为它的部分。

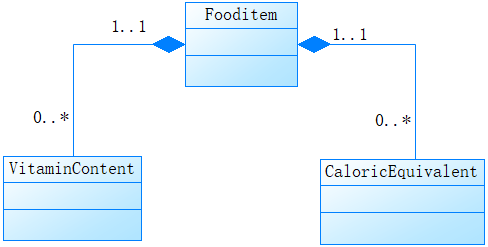


### 4. 组合

组合则是关联更强的聚合。**部分与整体共存亡**，是物理包容。

组合用**带有一个实心菱形的实线**表示。**菱形所在的一端是整体，另外一端是部分**。

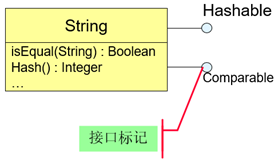
**整体**所在的一端的**多重性是1**，因为根据定义，**部分在整体之外就没有任何意义，整体拥有部分，部分的生命周期与整体式一样的**。



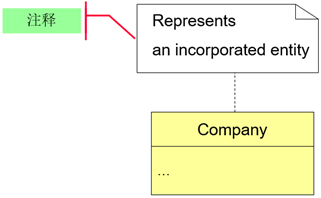
## 四、其他关系或符号

### 1. 接口

接口是未给出实现的对象行为的描述，接口包含操作，但没有属性，一个或多个类可以实现接口，每个类实现接口的操作。



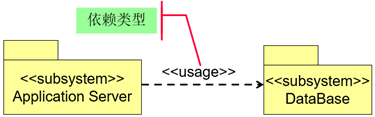
### 2. 注释



### 3. 依赖

依赖指明两个或两个以上模型元素之间的关系。

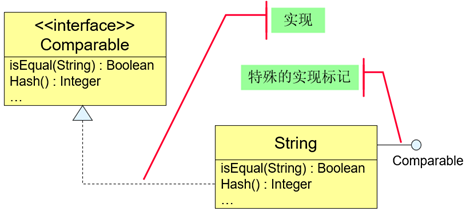
依赖有很多种类，比如：实现（realize）、使用（usage）、实例化（instantiate）、调用（call），派生（derive）、访问（access）、引入（import）、友元（friend）等等。



### 4. 实现

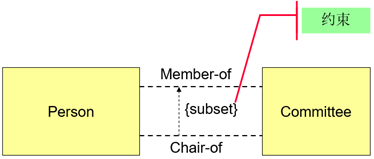
实现是依赖的一种，但由于它具有特殊意义，所以将它独立讲述。

实现是连接说明和实现之间的关系。



### 5. 约束

约束用来表示各种限制，如关联路径上的限制，和属性特征检测（存在、所有）。



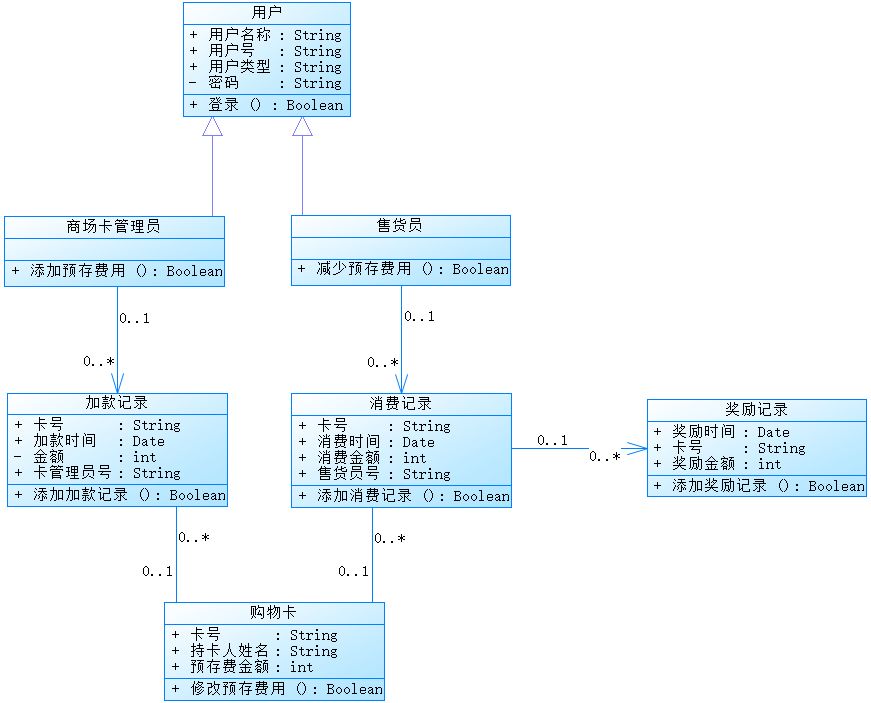
二：例子

**商场购物卡**

* 若某商场中发行一种**购物卡**，**用户**可以根据自己的需要提出**申请去办理购物卡**。
* 商场的**卡管理员**可以根据申请，**创建新的购物卡**。
* **用户**可以向购物卡**预存费用**，商场**卡管理员**为该用户**添加**相应的**预存费用**。
* **用户**可以用购物卡在商场进行**消费**，每进行一次消费，**售货员**就从购物卡中**扣除**相应的**费用**。
* 当购物消费每满一定数额时，**商场**自动在卡中**添加奖励费**。

**分析：**

* **购物卡**是一个类。
* 直接对购物卡类进行操作的有**商场卡管理员类**和**售货员类**，商场卡管理员和售货员都是用户，他们继承**用户类**的特性。
* 商场卡管理员可以添加预存费用，形成一个**加款记录类**。
* 售货员可以从卡中扣除费用，形成一个**消费记录类**。
* 当消费每满一定数额时，商场会在卡中添加奖励费，形成一个**奖励记录类**。



1.6.3对象图

1、什么是对象图

对象图（Object Diagram）描述的是参与交互的各个对象在交互过程中某一时刻的状态。和类图一样，对象图对系统的静态设计或静态进程视图建模，对象图更注重现实或原型实例，这种视图主要支持系统的功能需求，对象图描述了静态的数据结构。对象图可以被看作是类图在某一时刻的实例。

2、对象图的表示方法

在UML中，对象图使用的是与类图相同的符号和关系，因为对象就是类的实例。对于对象图来说无需提供单独的形式，类图中就包含了对象，所以只有对象而无类的类图就是一个“对象图”

3、对象图的目的

（1）正向和逆向工程

（2）一个系统的对象间的关系

（3）一个交互的静态视图

（4）了解对象的行为和他们的关系从实用的角度来看

4、对象图的适用范围

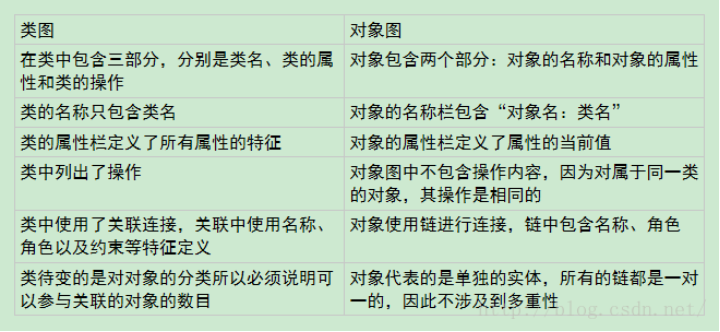
（1）一个系统的原型

（2）逆向工程

（3）造型复杂的数据结构

（4）从实用的角度了解系统

**5、类图与对象图的区别**



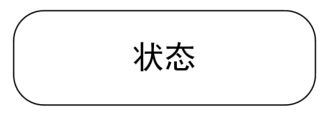
**二：例子**

1.6.4状态机图

一：定义

### 1. 状态

对象在事件发生之间某时刻所处的情形，用**圆角矩形**表示。



### 2. 转移

两个状态之间的关系，它表明当某事发生时，对象先从当前状态转换到后来的状态，用**带有标记事件的箭头**表示。



### 3. 事件

事件是引发变迁的消息，用**箭头上的标记**（事件表达式）表示。

### 4. 初始状态/终止状态

初态用**实心圆**表示，终态用一对**同心圆**表示；



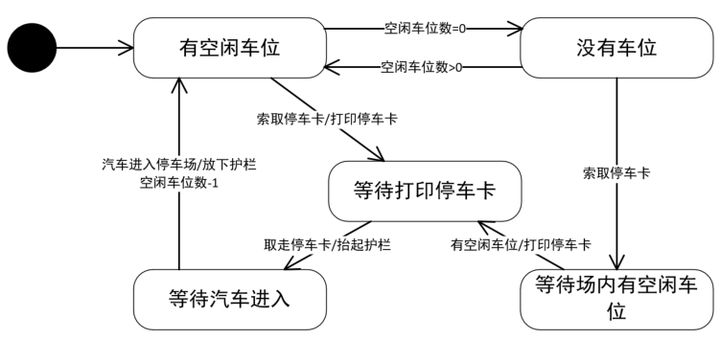
二：例子

**汽车停车场信息系统**

某汽车停车场欲建立一个信息系统，需求如下：

1. 在停车场的入口和出口分别安装一个自动栏杆、一台停车卡打印机、一台读卡器和一个车辆通过传感器。
2. 当汽车到达入口时，驾驶员按下停车卡打印机的按钮获取停车卡。当驾驶员拿走停车卡后，系统命令栏杆自动抬起；汽车通过入口后，入口处的传感器通知系统发出命令，栏杆自动放下。
3. 在停车场内分布着若干个付款机器。驾驶员将在入口处获取的停车卡插入付款机器，并缴纳停车费。付清停车费之后，将获得一张出场卡，用于离开停车场。
4. 当汽车到达出口时，驾驶员将出场卡插入出口处的读卡器。如果这张卡是有效的，系统命令栏杆自动抬起；汽车通过出口后，出口传感器通知系统发出命令，栏杆自动放下。若这张卡是无效的，系统不发出栏杆抬起命令而发出告警信号。
5. 系统自动记录停车场内空闲的停车位的数量。若停车场当前没有车位，系统将在入口处显示“车位已满”信息。这时，停车卡打印机将不再出卡，只允许场内汽车出场。

停车场入口护栏的**状态**有以下几个：有空闲车位，没有车位，等待打印停车卡，等待汽车进入，等待场内有空闲车位。



1.6.5活动图

活动图是UML用于对系统的动态行为建模的另一种常用工具，它描述活动的顺序，展现从一个活动到另一个活动的控制流。

活动图在本质上是一种流程图。活动图着重表现从一个活动到另一个活动的控制流，是内部处理驱动的流程。

**一、活动图的组成元素 Activity Diagram Element**

**1、活动状态图（Activity）——**活动状态用于表达状态机中的**非原子**的运行

　　活动状态图特点如下：

　　(1)、活动状态可以分解成其他子活动或者动作状态。

　　(2)、活动状态的内部活动可以用另一个活动图来表示。

　　(3)、和动作状态不同，活动状态可以有**入口动作**和**出口动作**，也可以有**内部转移**。

　　(4)、动作状态是活动状态的一个特例，**如果某个活动状态只包括一个动作，那么它就是一个动作状态**。

　　UML中活动状态和动作状态的图标相同，但是**活动状态可以在图标中给出入口动作和出口动作等信息**。

　　【图形】平滑的圆角矩形



**2、动作状态（Actions）——**动作状态是指**原子**的，不可中断的动作，并在此动作完成后通过完成转换转向另一个状态。

　　动作状态有如下特点：

　　(1)、动作状态是原子的，它是**构造活动图的最小单位**。

　　(2)、动作状态是**不可中断**的。

　　(3)、动作状态是**瞬时**的行为。

　　(4)、动作状态**可以有入转换**，入转换既可以是动作流，也可以是对象流。动作状态**至少有一条出转换**，这条转换**以内部的完成为起点，与外部事件无关**。

　　(5)、动作状态与状态图中的状态不同，**它不能有入口动作和出口动作，更不能有内部转移**。

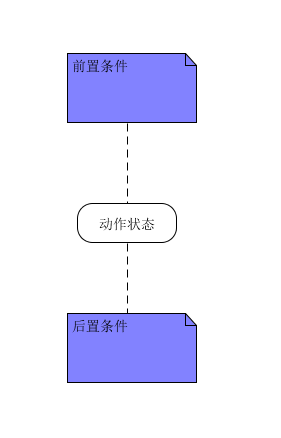
　　(6)、在一张活动图中，动作状态允许多处出现。

　　【图形】平滑的圆角矩形



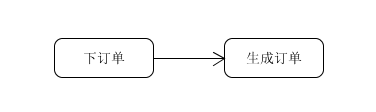
**3、动作状态约束（Action Constraints）——**动作状态约束：用来**约束动作状态**。

　　【图形】如下图展示了动作状态的前置条件和后置条件

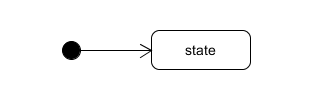


**4、动作流（Control Flow）——动作之间的转换**称之为动作流活动图的转换

　　【图形】用带箭头的直线表示，箭头的方向指向转入的方向。



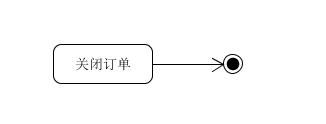
**5、开始节点（Initial Node）——**活动开始节点  
　　【图形】实心黑色圆点



**6、终止节点（Final Node）**——分为**活动终止节点（activity final nodes）**和**流程终止节点（flow final nodes）**

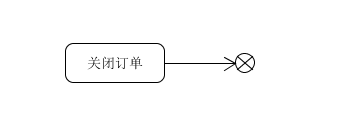
　　(1)、活动终止节点表示**整个活动**的结束

　　【图形】圆圈+内部实心黑色圆点



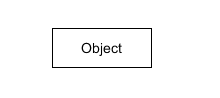
　　(2)、而流程终止节点表示是**子流程**的结束。

　　【图形】圆圈+内部十字叉



**7、对象（Objects）**

【图形】矩形方框　

****

**8、数据存储对象（DataStore）** ——使用关键字«datastore»

 　　【图形】矩形方框，内含关键字



**9、对象流（Object Flows）——**对象流是**动作状态**或者**活动状态**与**对象**之间的**依赖关系**，表示**动作使用对象或动作对对象的影响**。

　　用活动图描述某个对象时，可以把涉及到的对象放置在活动图中并用一个依赖将其连接到进行创建、修改和撤销的动作状态或者活动状态上，对象的这种使用方法就构成了对象流。

　　对象流中的对象有以下特点：

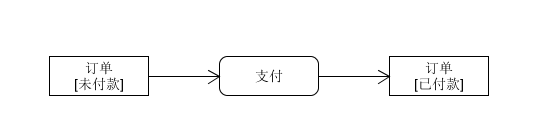
　　(1)、一个对象可以**由多个动作操作**。

　　(2)、一个动作输出的对象可以作为另一个动作输入的对象。

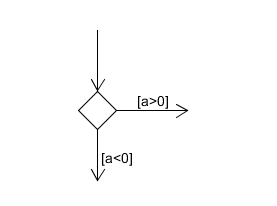
　　(3)、在活动图中，同一个对象可以多次出现，它的每一次出现表面该对象正处于对象生存期的不同时间点。

　　【图形】用带有箭头的虚线表示。如果箭头是从动作状态出发指向对象，则表示动作对对象施加了一定的影响。施加的影响包括创建、修改和撤销等。如果箭头从对象指向动作状态，则表示该动作使用对象流所指向的对象。

　　状态图中的**对象用矩形表示**，**矩形内是该对象的名称**，**名称下的方括号表明对象此时的状态**。



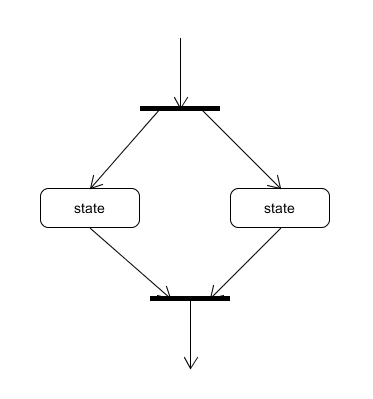
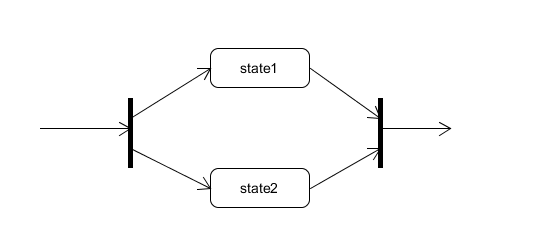
**10、分支与合并（Decision and Merge Nodes）——**选择分支  
　　【图形】分支与合并用菱形表示，它有一个进入转换（箭头从外指向分支符号），一个或多个离开转换（箭头从分支符号指向外）。而**每个离开转换上都会有一个监护条件**，用来表示满足什么条件的时候执行该转换。



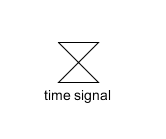
**11、分叉与汇合（Fork and Join Nodes）——**分叉用于将动作流分为两个或多个**并发运行的分支**，而**汇合则用于同步这些并发分支**，以达到共同完成一项事务的目的。

　　对象在运行时可能会存在两个或多个并发运行的控制流，为了对并发的控制流建模，UML中引入了分叉与汇合的概念。

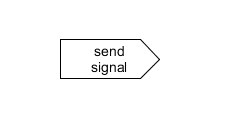
　　【图形】分为水平风向和垂直方向。



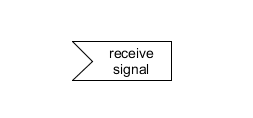
**12、时间信号**



**13、发送信号**

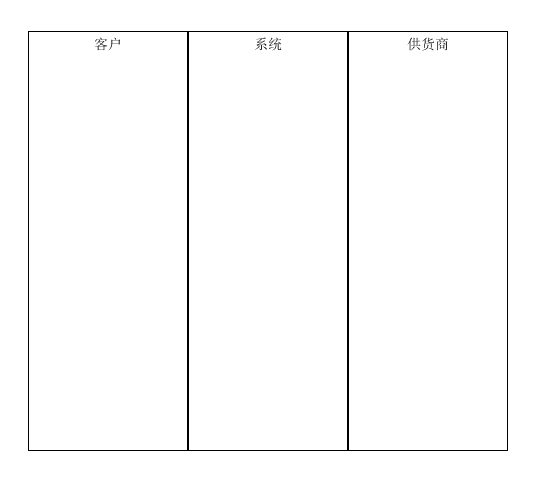
****

**14、接收信号**

****

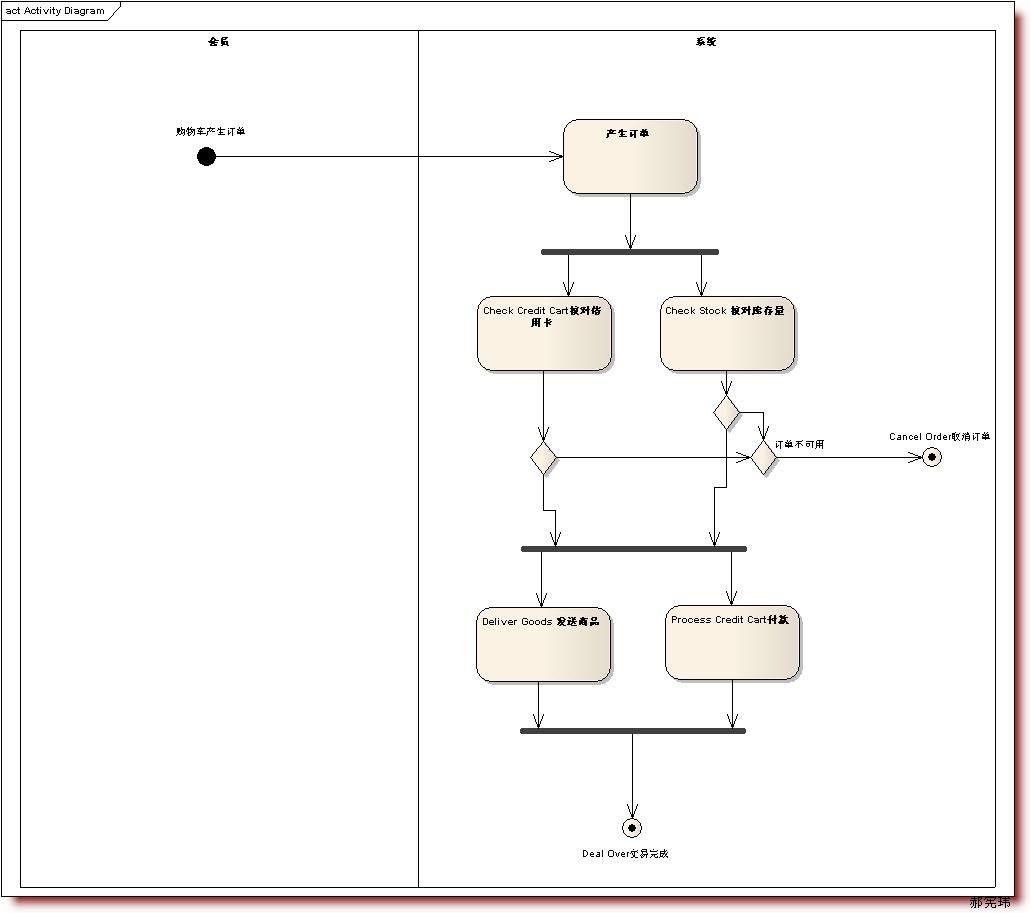
**14、泳道（Partition）——**泳道将活动图中的活动划分为若干组，并把每一组指定给负责这组活动的业务组织，即对象。  
在活动图中，泳道区分了负责活动的对象，它明确地表示了哪些活动是由哪些对象进行的。在包含泳道的活动图中，每个活动只能明确地属于一个泳道。

泳道是用垂直实线绘出，垂直线分隔的区域就是泳道。在泳道的上方可以给出泳道的名字或对象的名字，该对象负责泳道内的全部活动。泳道没有顺序，不同泳道中的活动既可以顺序进行也可以并发进行，**动作流和对象流允许穿越分隔线**。



**二、活动图案例分析**

**例1.购物用例图**



1、  泳道分为：会员泳道和系统泳道。会员选择商品并加入购物车，系统完成订单生成及其支付完毕。

2、  开始节点：会员添加商品到购物车，点击【订单确认】，开始交于系统处理订单流程

3、  结束节点：商品发送完毕和付款成功，订单处理流程结束

4、  活动状态：产生订单、Check Credit Cart核对信用卡、Check Stock 核对库存量、Deliver Goods 发送商品、Process Credit Cart付款

5、  分叉与汇合：【产生订单】份叉为检查库存量和会员支付金额是否足够，如果不足，取消订单，如过库存量和支付金额足够，发送商品和付款，最后汇合为订单完成。

1.6.6顺序图

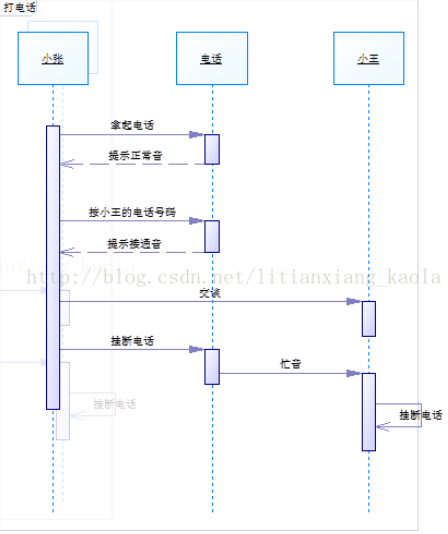
一：定义

1，UML顺序图一般用于确认和丰富一个使用情境的逻辑。

2，一个使用情境的逻辑或是一个用例的一部分；或是一条扩展路径；或是一个贯穿单个用例的完整路径，例如动作基本过程的逻辑描述；或是动作的基本过程的一部分再加上一个或多个的备用情境的逻辑描述；或是包含在几个用例中的路径。

3，顺序图将交互关系表现为一个二维图，纵向是时间轴，时间沿竖线向下延伸。横向轴代表了在协作中各独立对象的类元角色，类元角色的活动用生命线表示。

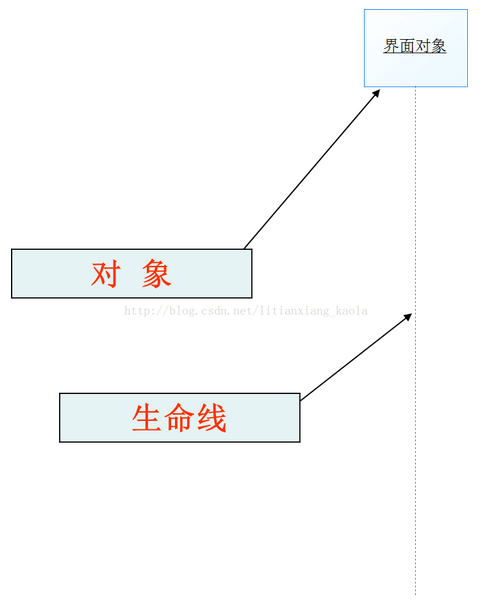
二：顺序图示例



三：顺序图的组成元素

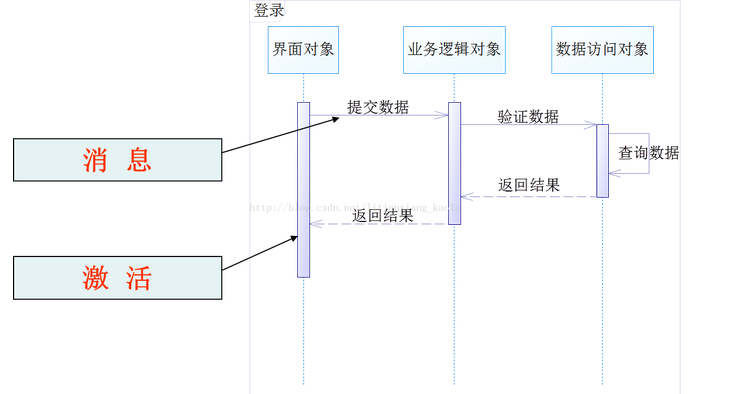
1，生命线用一条纵向虚线表示。

2，在UML中，对象表示为一个矩形，其中对象名称标有下划线。

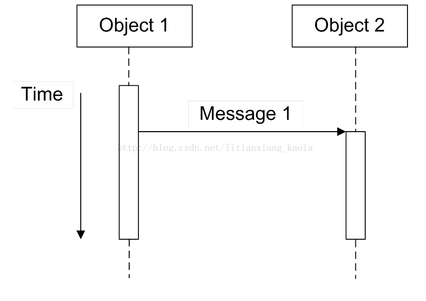


3，激活是过程的执行，包括等待过程执行的时间。在顺序图中激活部分替换生命线，使用长条的矩形表示。

4，消息是对象之间的通信，是两个对象之间的单路通信，是从发送者到接收者之间的控制信息流。消息在顺序图中由有标记的箭头表示，箭头从一个对象的生命线指向另一个对象的生命线，消息按时间顺序在图中从上到下排列。



5，在顺序图中，对象安排在X轴。启动交互的对象放在最左边，随后放入消息的对象放在启动交互对象的右边。交互中对象发送和接收的消息沿着Y轴以时间增加的次序放置。在顺序图中，有的消息对应于激活，表示它将会激活一个对象，这种消息称为调用消息(Call Message)；如果消息没有对应激活框，表示它不是一个调用消息，不会引发其他对象的活动，这种消息称为发送消息(Send Message)。

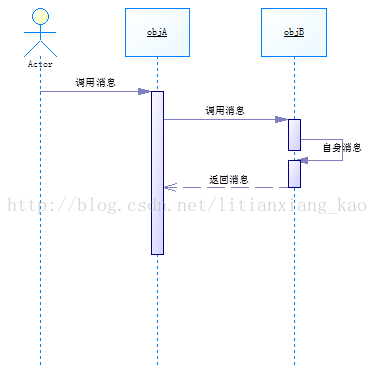


6，消息

>调用消息

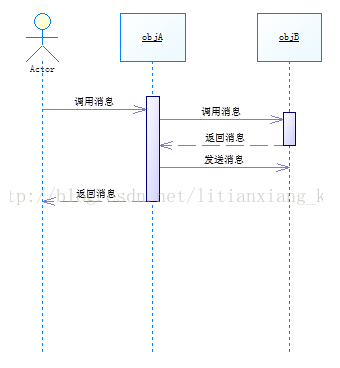
>返回消息

>自身消息



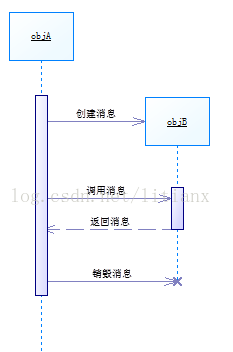
>调用消息

>发送消息



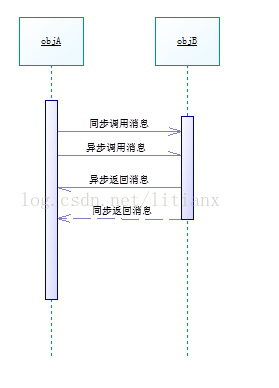
>创建消息

>销毁消息



>同步消息：发送消息的对象要等到接收消息的对象执行完所有操作后，发送消息的对象才能继续执行自己的操作。

>异步消息：发送消息的对象发送消息后，不用等待接收对象是否执行，继续执行自己的操作。



7，交互片段

一个复杂的顺序图可以划分为几个小块，每一个小块称为一个交互片段。每个交互片段由一个大方框包围，其名称显示在方框左上角的间隔区内，表示该顺序图的信息。常用操作符如下：

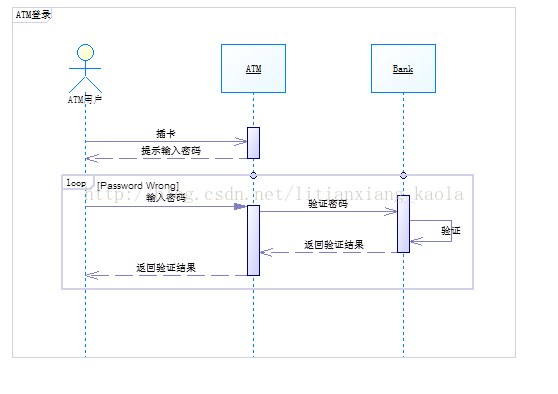
>alt:多条路径，条件为真时执行。

>opt:任选，仅当条件为真时执行。

>par:并行，每一片段都并发执行。

>loop:循环，片段可多次执行。

>critical:临界区，只能有一个线程对它立即执行。



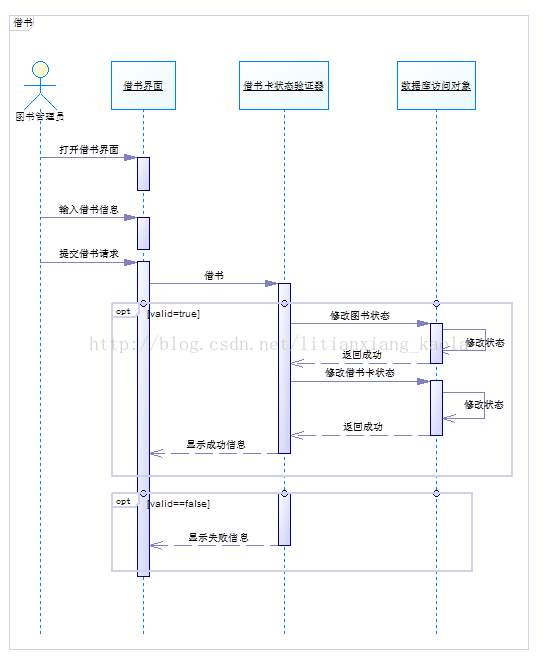
四：顺序图的作用

1，对于业务人员，顺序图可显示不同的业务对象如何交互，对于交流当前业务如何进行很有用。除记录组织的当前事件外，一个业务级的顺序图能被当作一个需求文件使用，为实现一个未来系统传递需求。

2，对于需求分析人员，顺序图能通过提供一个深层次的表达，把用例带入下一层次。通常用例被细化为一个或者更多的顺序图。顺序图的主要用途之一，是把用例表达的需求，转化为进一步、更深层次的精细表达。

3，对于技术人员，顺序图在记录一个未来系统的行为应该如何表现时非常有用。在设计阶段，架构师和开发者能使用顺序图挖掘出系统对象间的交互，进一步完善整个系统的设计。

五：例子



>绘制图书管理系统“借书”用例的顺序图（业务模型）。  
图书管理员打开借书界面，输入借书信息并提交借书请求；系统验证借书卡状态，如果借书卡未借书则记录借书信息且修改图书状态和借书卡状态，并提示借书成功；否则提示借书失败。

1.6.7通信图

**一：定义**

1，在UML1.0中，通信图称为协作图。

2，通信图强调参与一个交互对象的组织。

3，它与顺序图是同构图，也就是它们包含了相同的信息，只是表达方式不同而已，通信图与顺序图可以相互转换。

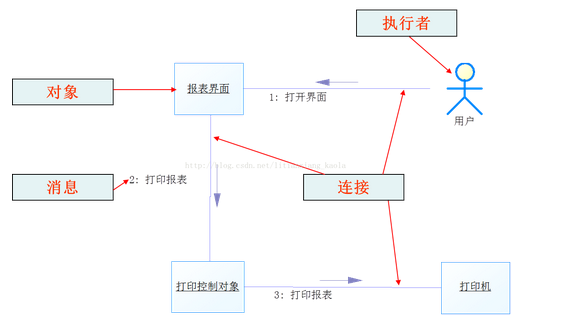
4，虽然通信图和顺序图均显示了交互，但它们强调了不同的方面。顺序图清晰地显示了时间次序，但没有显式指明对象间关系。通信图清晰地显示了对象间关系，但时间次序必须从顺序号来获得。

5，顺序图常用于用例场景描述，通信图更适合显示过程设计细节。采用哪种图，一般可以根据这种原则：当对象及其连接有利于理解交互时，选择通信图；当只需了解交互的次序时，选择顺序图。

二：组成元素

1，通信图由以下基本元素组成：执行者(Actor)、对象(Object)、连接(Link，也称为链)、消息(Message)和守护条件(Condition)。

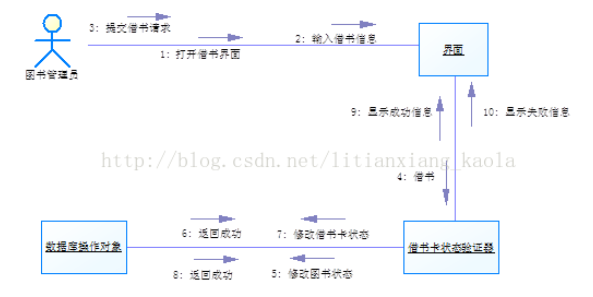
2，在UML中，使用实线表示两个对象之间的连接；通信图中的消息，由在连接上方的带有标记的箭头表示，同时可以用数字注明消息的次序。



3，守护条件：以“[条件表达式]”格式表示



三：实例



1.6.8构件图

描述软件的各种组件和他们之间的关系

主要思想：

能容易地在你的设计中重用及/或替换一个不同的组件实现，因为一个组件封装了行为，实现了特定接口。

元素：构件+接口+依赖关系

构件：

定义了良好接口的物理实现单元。构件是系统中可替换的物理部件，它包装了实现而且遵从并统一提供一组接口的实现。构件一般是实际存在的，物理的部件。

构件和类的相同点：都可以实现一组接口，都可以参与依赖关系，都可被嵌套，都可以有实例，都可以参与交互。

构件和类的区别：类描述了软件设计的逻辑组织。而组件图描述软件设计的物理实现，每个组件体现了系统设计中特定类的实现。

接口：

组件可以通过其他组件的接口，使用其他组件中定义的一些操作。

关系：

依赖（使用）

客户端组件依赖于提供者组件

实现

组件可以实现接口

### 1. 构件

构件是定义了良好接口的物理实现单元，是系统中可替换的物理部件。

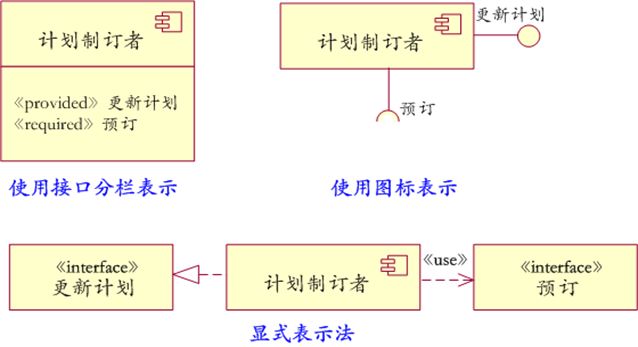
构件可以是源代码构件、二进制构件或一个可执行的构件。

在UML中，构件**用一个左侧带有突出两个小矩形的矩形来表示**。



对于一个构件而言，它有**两类接口**：**提供（provided）接口**和**所需（required）的接口**。

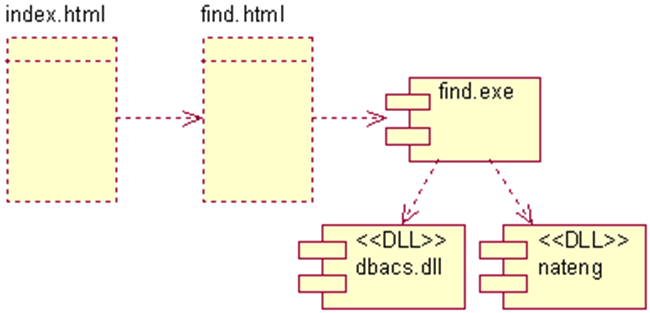
标识接口的构件表示方法有3种。



### 2. 依赖关系

构件间的关系以依赖的形式表达。把提供服务的构件称为提供者，把使用服务的构件称为客户。

在UML中，构件图中依赖关系的表示方法与类图中依赖关系相同，都是**一个由客户指向提供者的虚线箭头**。



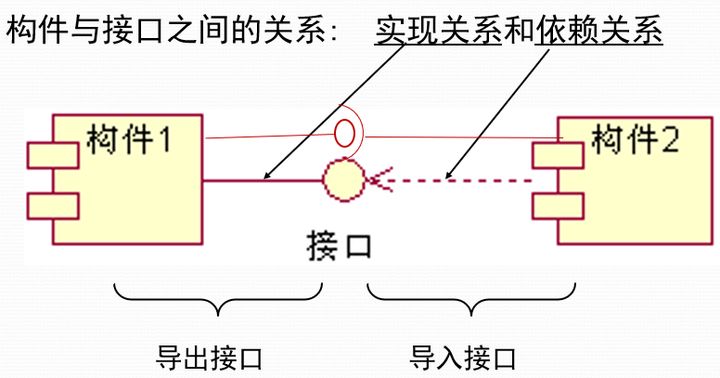
### 3. 接口

接口用于描述构件所提供的服务的一组操作集合，指定了构件的外部可见操作。可以通过接口访问一个构件。

接口和构件之间的关系分为两种：

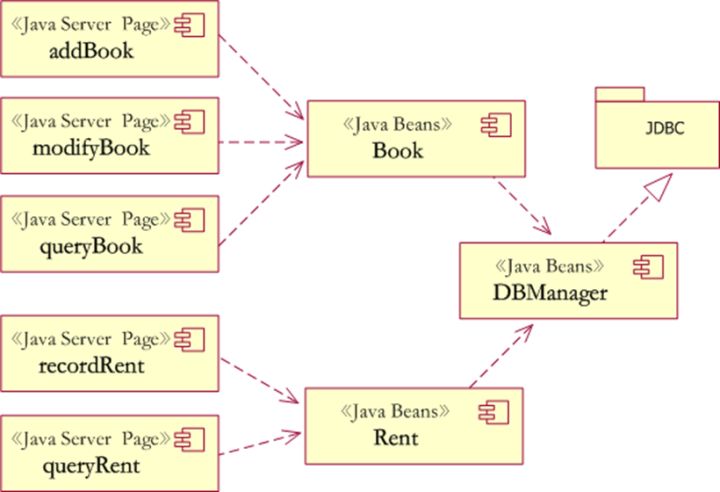
* 实现关系（Realization）
* 依赖关系（Dependency）

在UML图中，**接口和构件**之间用**实线**连接表示**实现关系**；而**接口和构件**之间用**虚线箭头**连接则表示**依赖关系**。



* 导出接口由提供操作的构件提供。
* 导入接口供访问操作的构件使用。

## 例题：对系统实现的体系结构进行建模



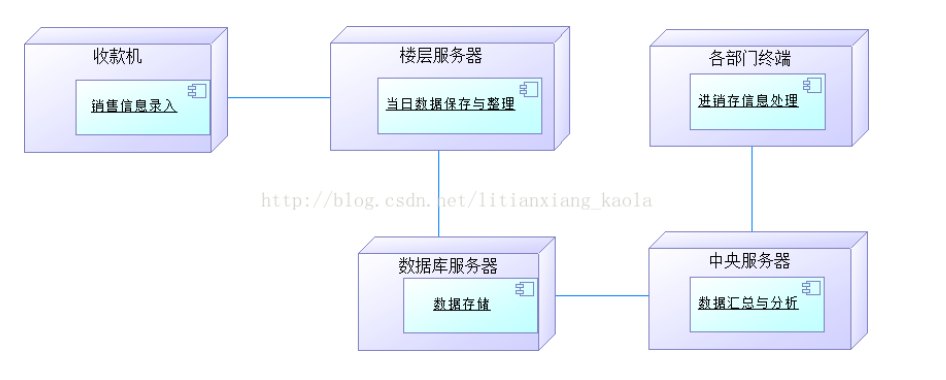
1.6.9部署图

一：定义

1，部署图(Deployment Diagram)，也称为实施图，它和组件图一样，是面向对象系统的物理方面建模的两种图之一。组件图是说明组件之间的逻辑关系的，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及在此结构上执行的软件。部署图可以显示计算节点的拓扑结构和通信路径、节点上运行的软件组件。

2，在UML中，部署图显示了系统的硬件和安装在硬件上的软件，以及用于连接异构计算机之间的中间件。部署图通常被认为是一个网络图或者物理架构图。

二：实例



三：使用场景

部署图可用于以下用途：

1. 为系统的网络和硬件拓扑建模。
2. 为分布式网络和系统建模。
3. 实施转发和逆向工程流程。
4. 为客户端/服务器系统的硬件详细信息建模。
5. 用于对嵌入式系统进行建模。