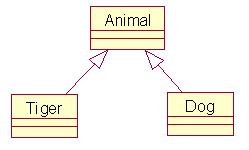
# 1.UML关系

UML类图中的关系分为四种：泛化关系、依赖关系、关联关系、实现关系；关联关系又可以细化为聚合和组合。

## 1.1 泛化（Generalization)

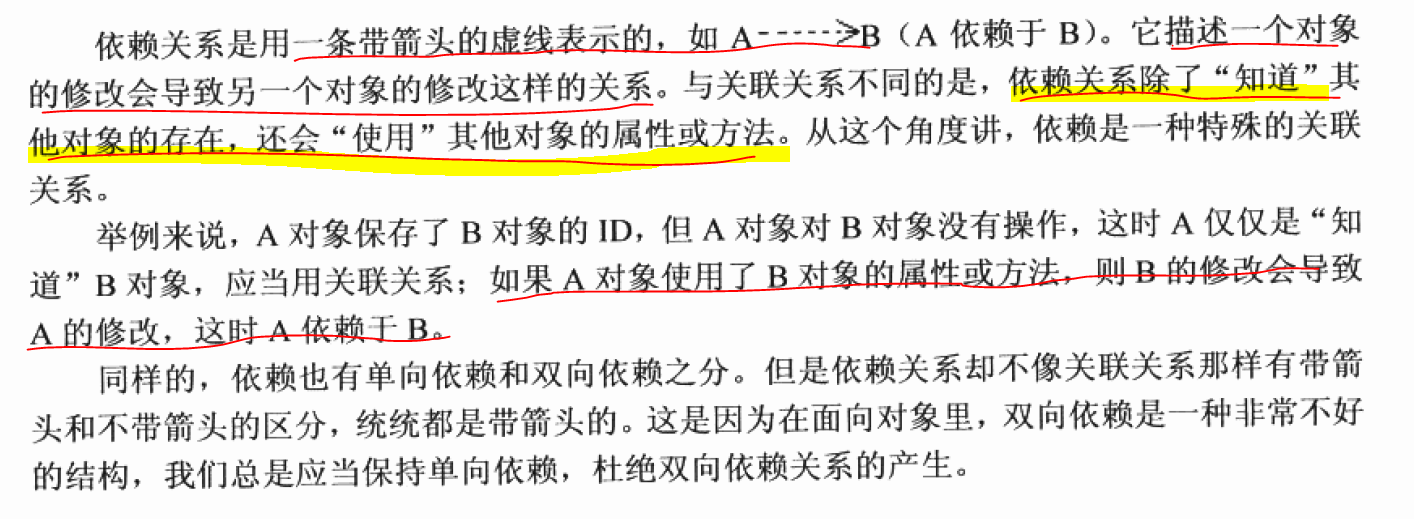
泛化是**父类和子类之间的关系，子类继承父类的所有结构和行为**。在子类中可以增加新的结构和行为，也可以覆写父类的行为。

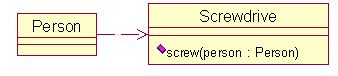
**[](http://photo.blog.sina.com.cn/showpic.html#blogid=4c4d6e740100aixn&url=http://s8.sinaimg.cn/orignal/4c4d6e74t558fadcf4497)**

## 1.2. 依赖（Dependencies）

依赖关系是一种**使用关系**，特定事物的改变有可能会影响到使用该事物的事物，反之不成立。在你想显示一个事物使用另一个事物时使用，两个元素之间的一种关系，其中一个元素（服务者）的变化将影响另一个元素（客户），或向它（客户）提供所需信息。它是一种组成不同模型关系的简便方法。依赖表示两个或多个模型元素之间语义上的关系。它只将模型元素本身连接起来而不需要用一组实例来表达它的意思。它表示了这样一种情形，提供者的某些变化会要求或指示依赖关系中客户的变化。

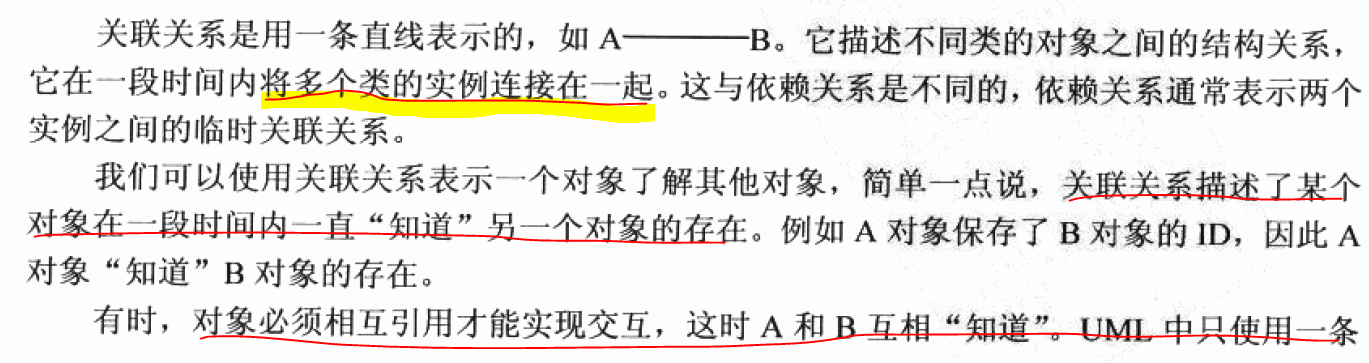
根据这个定义，**关联和泛化都是依赖关系**，但是它们有更特别的语义，故它们有自己的名字和详细的语义。我们通常用依赖这个词来指其他的关系。**依赖用一个从客户指向提供者的虚箭头表示**，用一个构造型的关键字来区分它的种类，通常情况下，**依赖关系体现在某个类的方法使用另一个类作为参数** 。

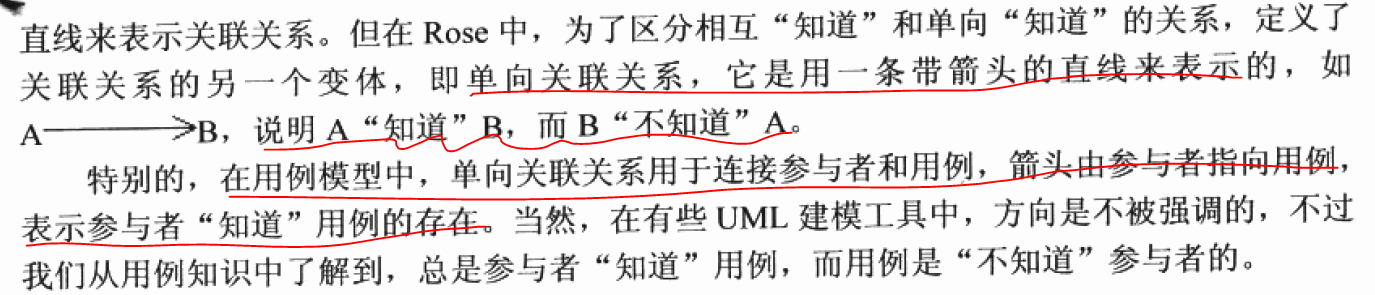


**[](http://photo.blog.sina.com.cn/showpic.html#blogid=4c4d6e740100aixn&url=http://s1.sinaimg.cn/orignal/4c4d6e74t558fca7db7d0)**

## **1.3. 关联（Association**）

关联是一种结构化的关系，指**一种对象和另一种对象有联系**。**给定有关联的两个类，可以从一个类的对象得到另一个类的对象**。



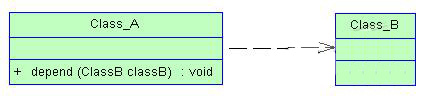


类与类之间由弱到强关系是:    **没关系 > 依赖 > 关联 > 聚合 > 组合。**

 类和类之间八竿子打不着那就是没关系，这个没啥歧义。

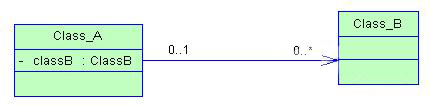
**依赖(dependency)**

　　可以简单的理解，就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、、临时性的、非常弱的，但是B类的变化会影响到A；比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖；表现在代码层面，为类B作为参数被类A在某个method方法中使用。用带虚线的箭头。



**关联(association)**

　　他体现的是两个类、或者类与接口之间语义级别的一种强依赖关系，比如我和我的朋友；这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，而且双方的关系一般是平等的、关联可以是单向、双向的；表现在代码层面，为被关联类B以类属性的形式出现在关联类A中，也可能是关联类A引用了一个类型为被关联类B的全局变量；



**依赖和关联区别：**我用锤子修了一下桌子，我和锤子之间就是一种依赖，我和我的同事就是一种关联。依赖是一种弱关联，只要一个类用到另一个类，但是和另一个类的关系不是太明显的时候（可以说是“uses”了那个类），就可以把这种关系看成是依赖，依赖也可说是一种偶然的关系，而不是必然的关系。关联是类之间的一种关系，例如老师教学生，老公和老婆这种关系是非常明显的。依赖是比较陌生，关联是我们已经认识熟悉了。

### 1.3.1 聚合（Aggregation）

聚合是**一种特殊的关联。它描述了“has a”关系，表示整体对象拥有部分对象**。

关联关系和聚合关系来语法上是没办法区分的，从语义 上才能更好的区分两者的区别。聚合是较强的关联关系，强调的是整体与部分 之间的关系。

与关联关系一样，聚合关系也是通过类的成员变量 来实现的。

图 12: 一个聚合关联的例子

### 1.3.2 组合（Composition）

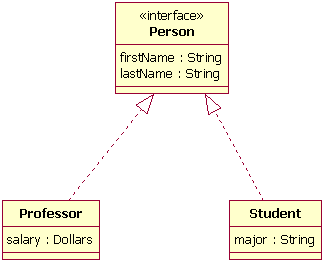
**组合是聚合的一种形式，它具有更强的拥有关系，强调整体与部分的生命周期 是一致的**。整体负责部分的生命周期的管理。**如果整体被销毁，部分也必须跟着一起被销毁**，如果所有者被复制，部分也必须一起被复制。

与关联关系一样，组合关系也是通过类的成员变量 来实现的。

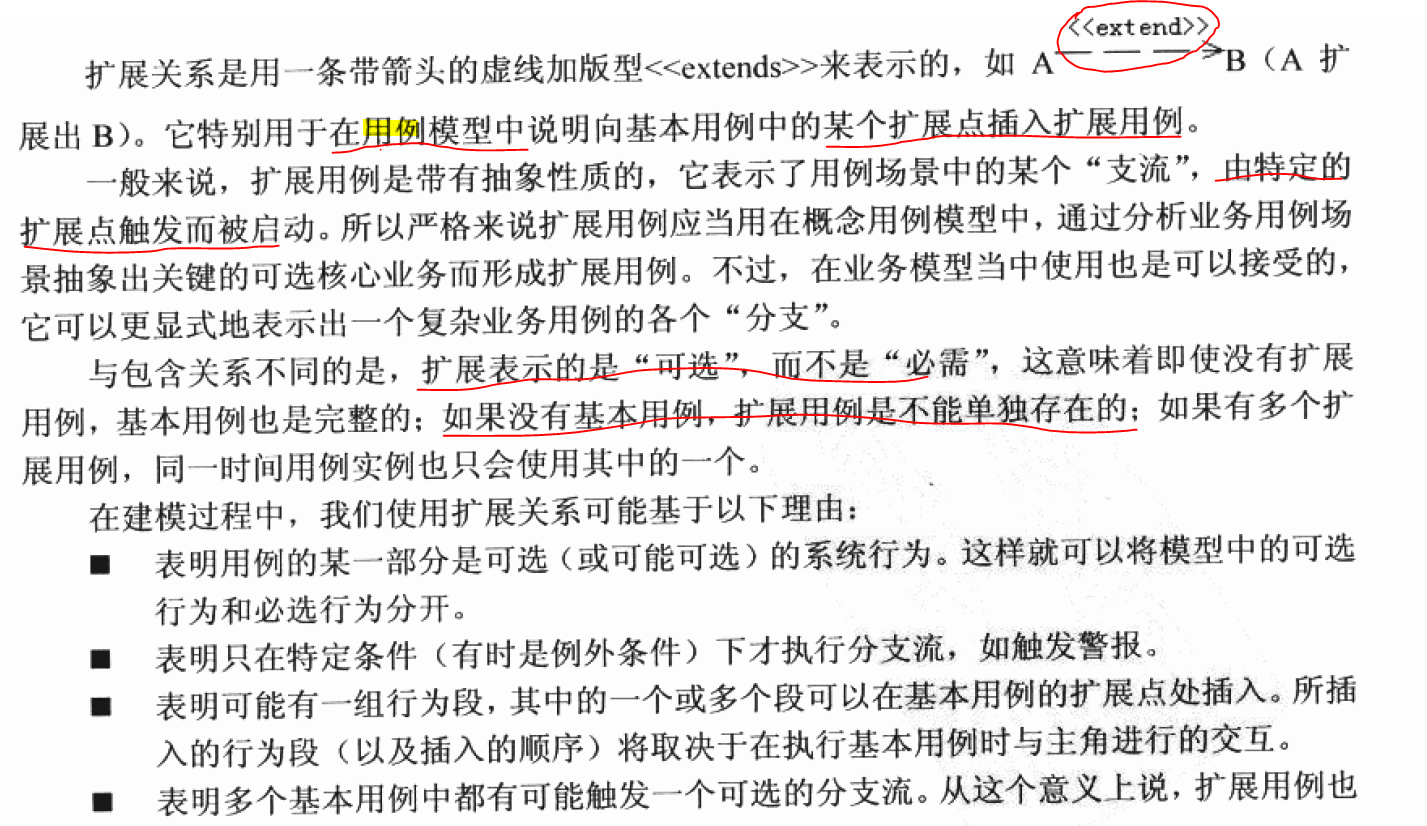
图 13: 一个组合关系的例子

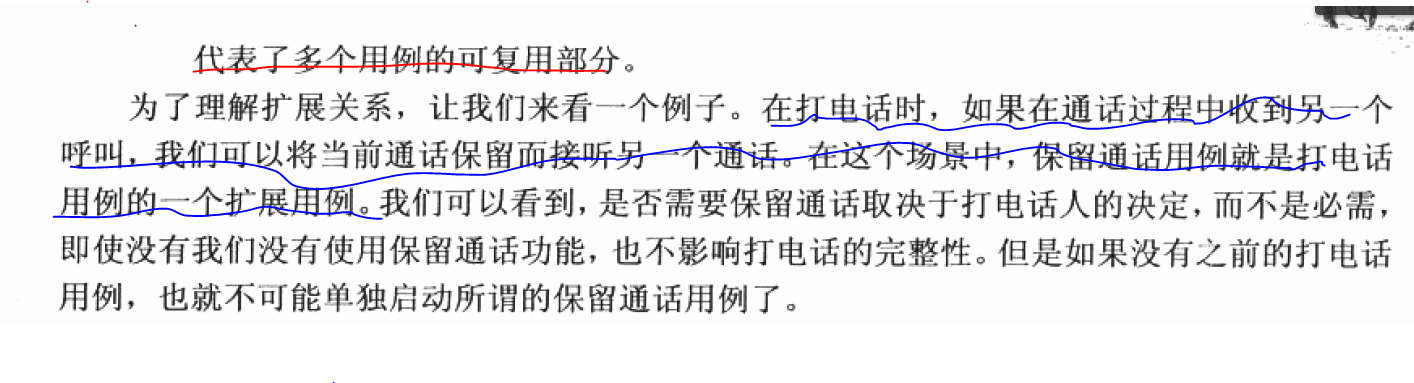
## 1.4. 实现（Realization）

**实现关系指定两个实体之间的一个合约。换言之，一个实体定义一个 合约 ，而另一个实体保证履行该 合约 。**

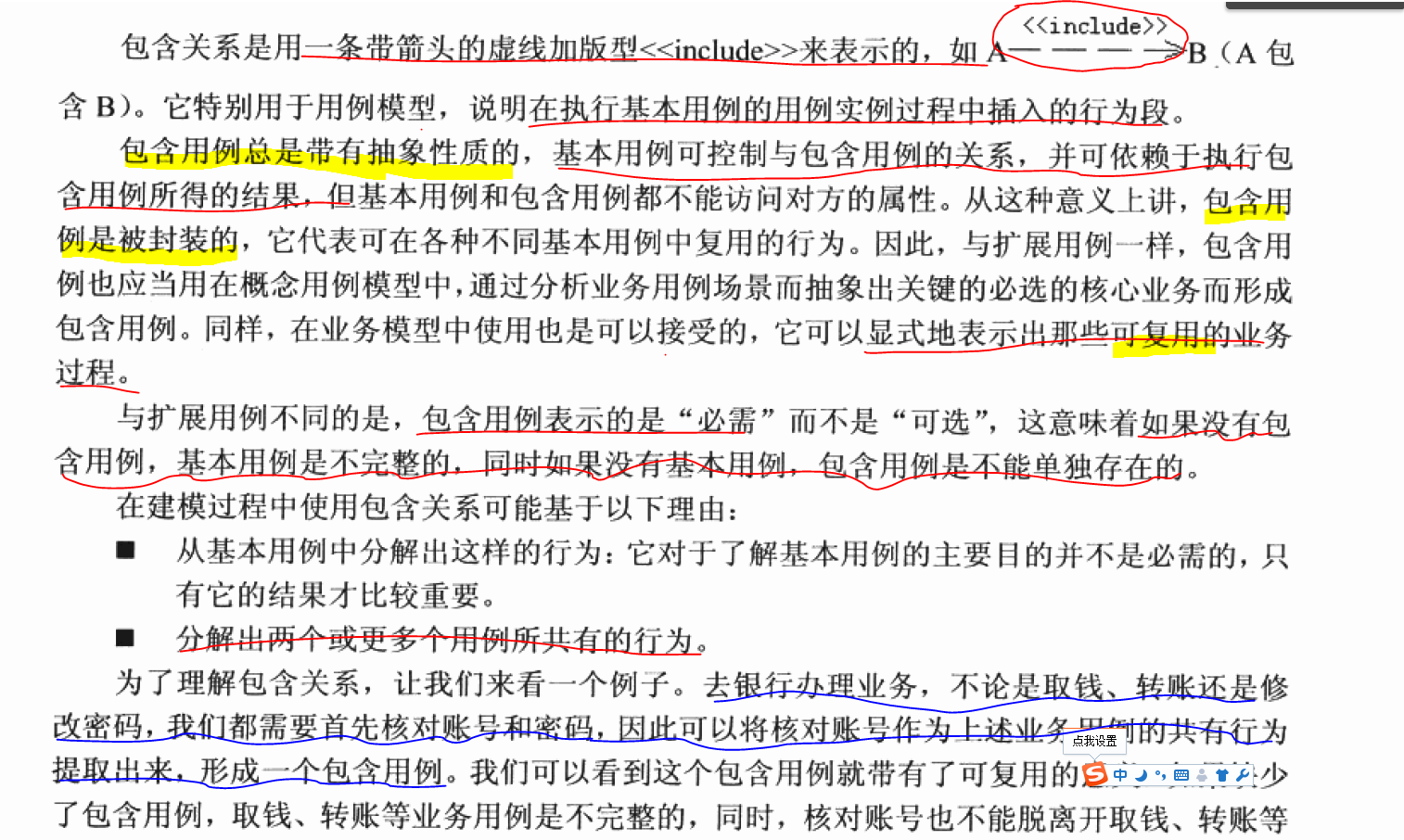


## 1.5 扩展关系(extends)

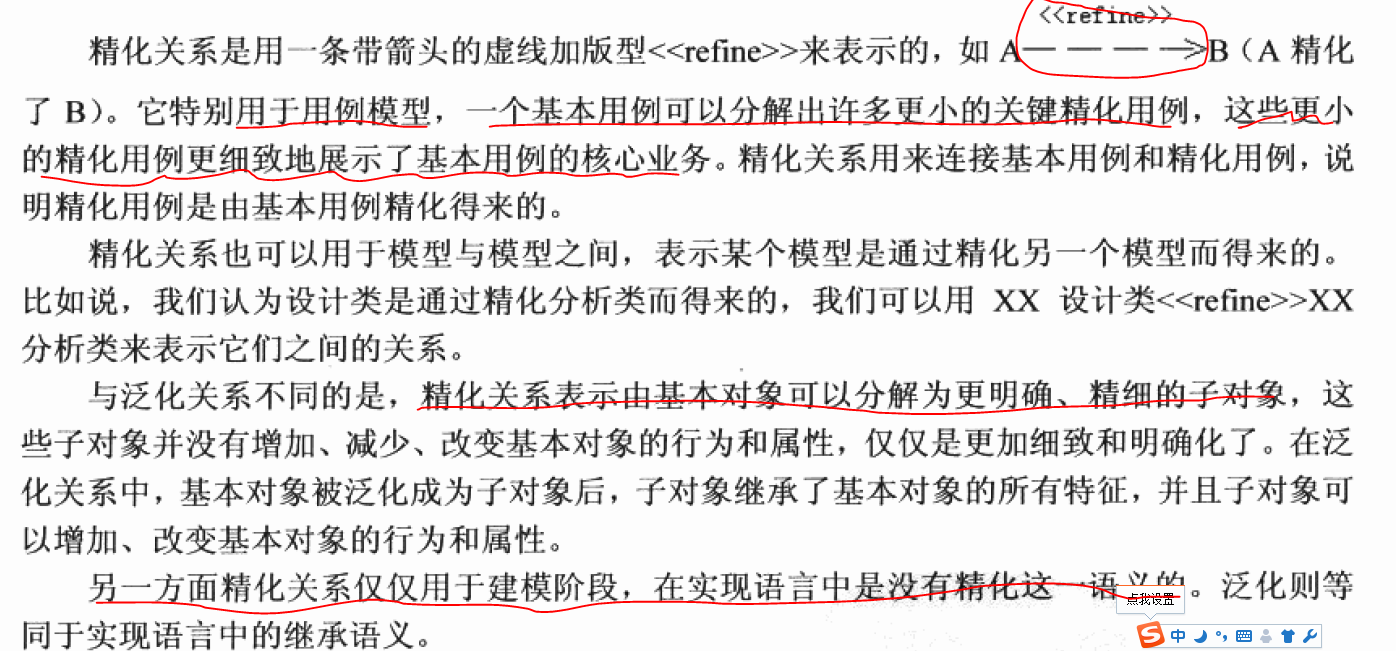


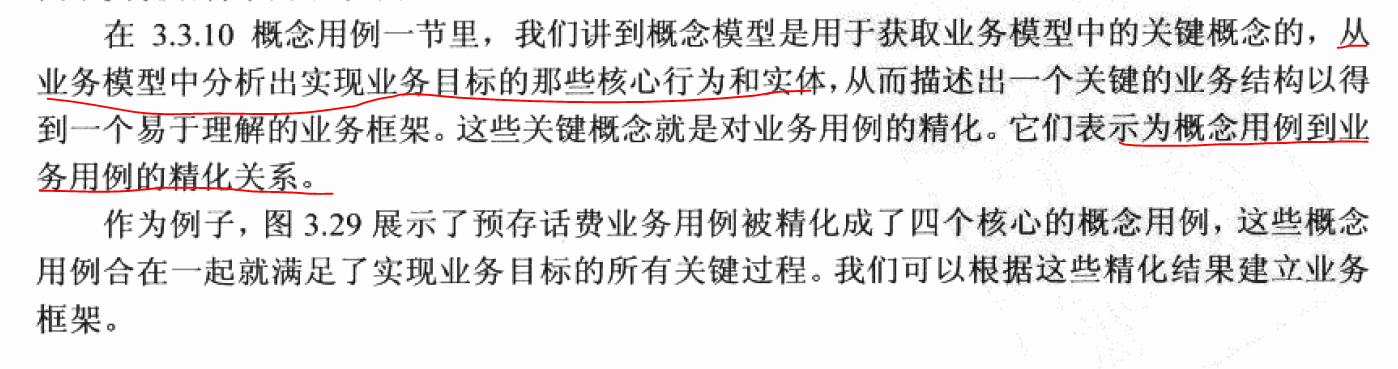


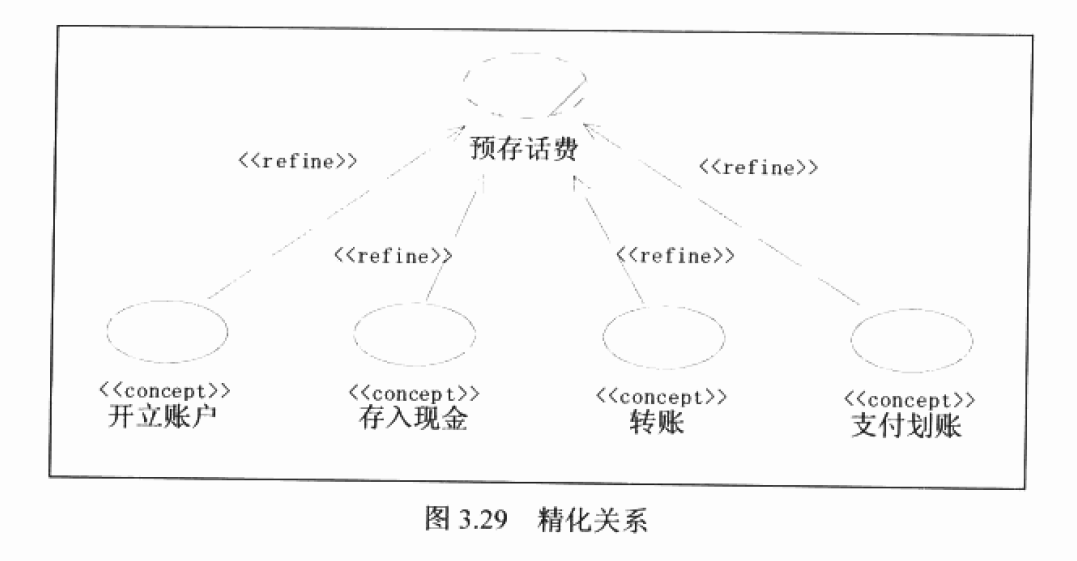
## 1.6 包含(include)



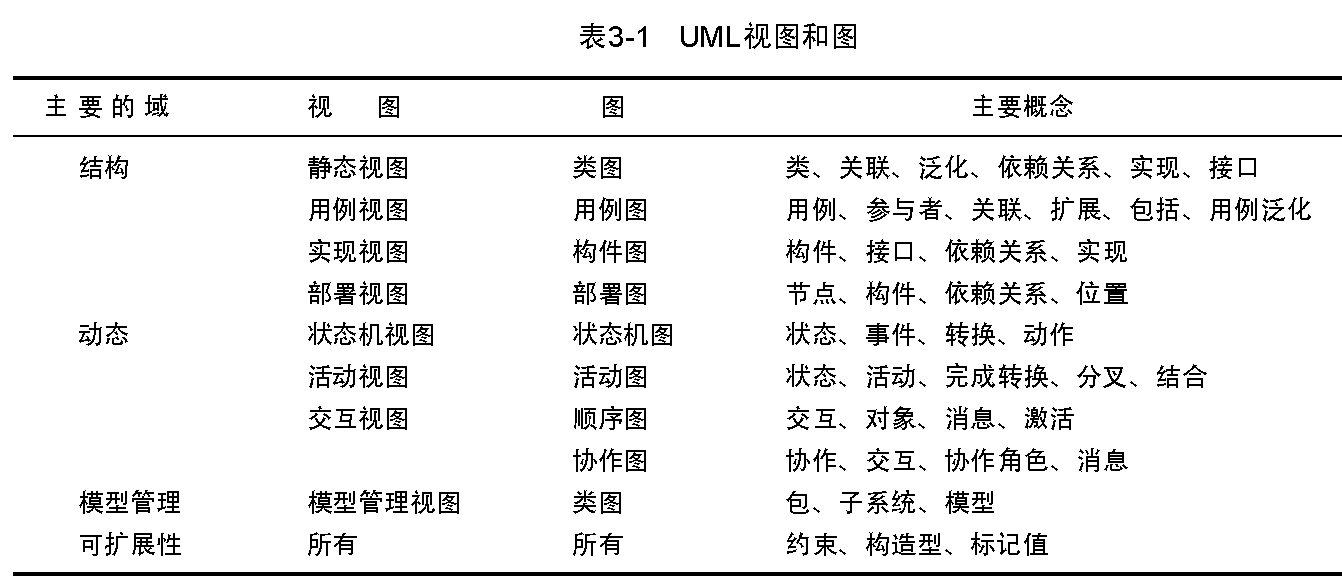
## 1.7 精化关系（refine）

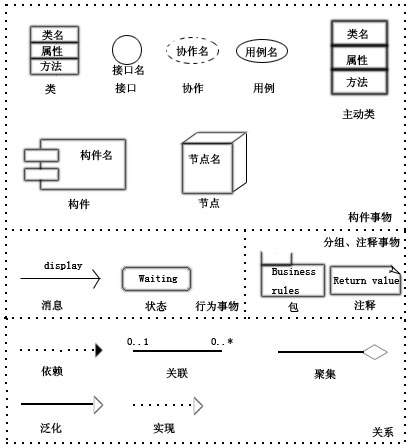






# 2 UML视图





说明：

构件事物是名词，是模型的静态部分。

行为事物是动态部分，表示行为。

分组事物是组织部分。

注释事物是解释部分。

依赖：一个事物变化会引起另一个事物变化。

聚集：特殊的关联，描述整体与部分的组合关系。

泛化：是一种特殊与一般的关系，如子元素（特殊）与父元素（一般），箭头指向父元素。

实现：类元之间的关系，其中一个类元指定了由另一个类元保证执行的契约。一般用在接口和实现他们的类之间或用例和实现它们的协作之间。

## 2.1 类图

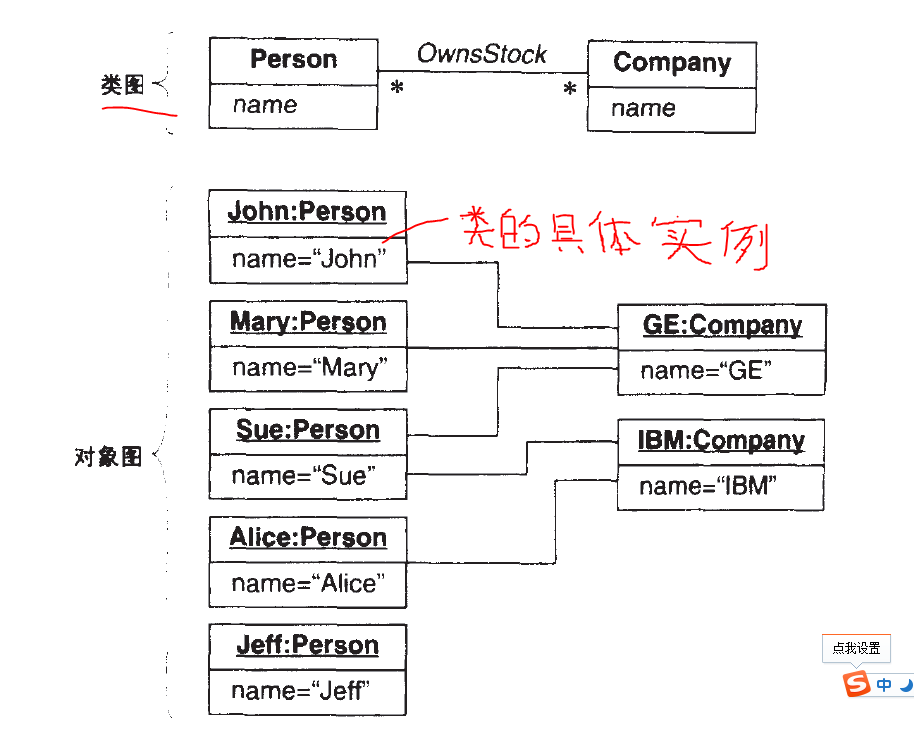
**用于展现系统中的类以及其之间的关系**

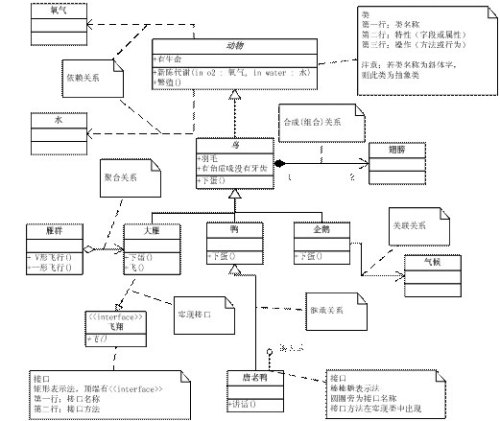
**对象图：显示了单独的对象及其关系。对象图有助于记录测试用例以及讨论用例。**

•静态图：包括类图和对象图。

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。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时间段存在**。



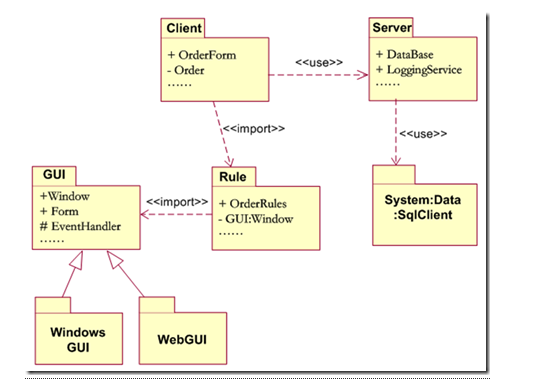


### 2.1.1 包

**包可直接理解为命名空间，文件夹，是用来组织图形的封装，包图可以用来表述功能组命名空间的组织层次**。

•在面向对象软件开发的视角中，类显然是构建整个系统的基本构造块。但是对于庞大的应用系统而言，其包含的类将是成百上千，再加上其间“阡陌交纵”的关联关系、多重性等，必然是大大超出了人们可以处理的复杂度。这也就是引入了“包”这种分组事物构造块。

•包的作用是：   
1）对语义上相关的元素进行分组；   
2）定义模型中的“语义边界”；   
3）提供配置管理单元；   
4）在设计时，提供并行工作的单元；   
5）提供封装的命名空间，其中所有名称必须惟一



上图解释

•首先根据《use》关系，可以发现Client包使用Server包，Server包使用System.Data.SqlClient包，结合其元素，不难得知Client负责Order（订单）的输入，并通过Server来管理用户的登录（LoggingService）和数据库存储（DataBase），而Server包还将通过.NET的SQL Server访问工具包来实现与数据库的实际交互。

•接着再看两个《import》，从包的命名和其所属的元素不难发现Rule负责处理一些规则，并引用一个具体的窗体（Window），而Client包则通过引用Rule来实现整个窗体和表单的显示、输入等。并且还将暂存Order（订单）信息。

•最后来看包的泛化关系，GUI有两个具体实现，一个是针对C/S的WindowsGUI，一个是实现B/S的WebGUI。

依赖关系

•**《use》使用关系：是一种默认的依赖关系，说明客户包（发出者）中的元素以某种方式使用提供者包（箭头指向的包）的公共元素**，也就是说**客户包依赖于提供者**包

**•《import》引用关系**：最普遍的包依赖类型，说明**提供者包(箭头指向的包)的命名空间（包本身代表命名空间）将被添加到客户包（发出者）的命名空间中，客户包中的元素也能够访问提供者包的所有公共元素**

**•《access》访问关系**：只想使用提供者包中的元素，而不想将其命名空间合并则应使用该关系

**•《trace》追溯关系**：想表示一个包到另一个包的历史发展，则需要使用《trace》关系来表示

**例子描述**

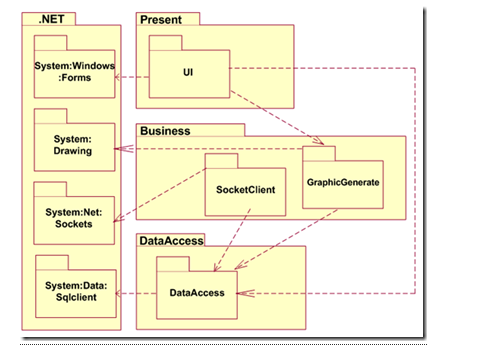
•分析系统工作流程：   
1）通过Internet连接到股票信息服务器，获取实时的股票信息，并存入数据库中。   
2）根据用户的输入和选择，从数据库中获取相应的信息，展现在屏幕中。   
3）在数据的展现过程中，将需要绘制大量的图表

•根据功能模块组织包：

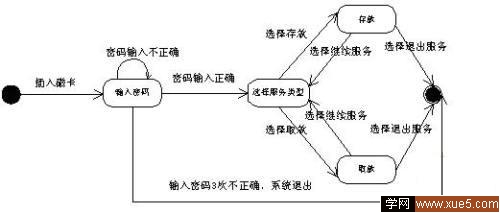


包之间的依赖关系

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/999a0590db07_F8CE/image_10.png)



## 2.2 状态图

展示了一个**状态机，由状态、转换、事件和活动组成。强调事件行为的顺序**。如下图(摘自网络)：  


### 2.2.1 事件

**事件是指某个时刻发生的事情。**

**信号是指从一个对象到另一个对象的明确的单向信号流动。**

**信号事件：是指发送或者接受信号的事件。**

**区别：信号是对象间的消息，而信号事件是指某个时刻发生的事情。**

**变更事件：是指满足布尔表达式而引起的事件。**

**时间事件：是指在绝对时间上或者在某个时间间隔内发生的事情而引起的事件。**

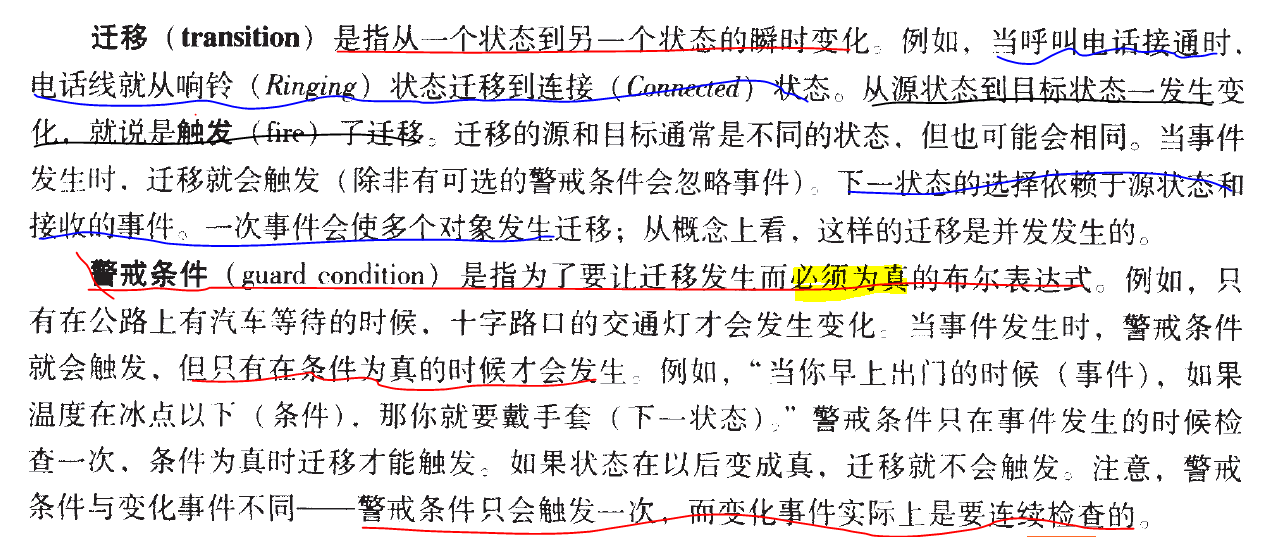
### 2.2.2 状态

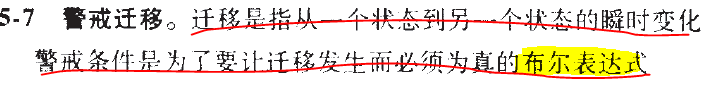
是对象取值和链接的抽象。根据**对象的总体行为，将取值和链接的集合组成一个状态**。

事件表示时间点，状态表示时间段。

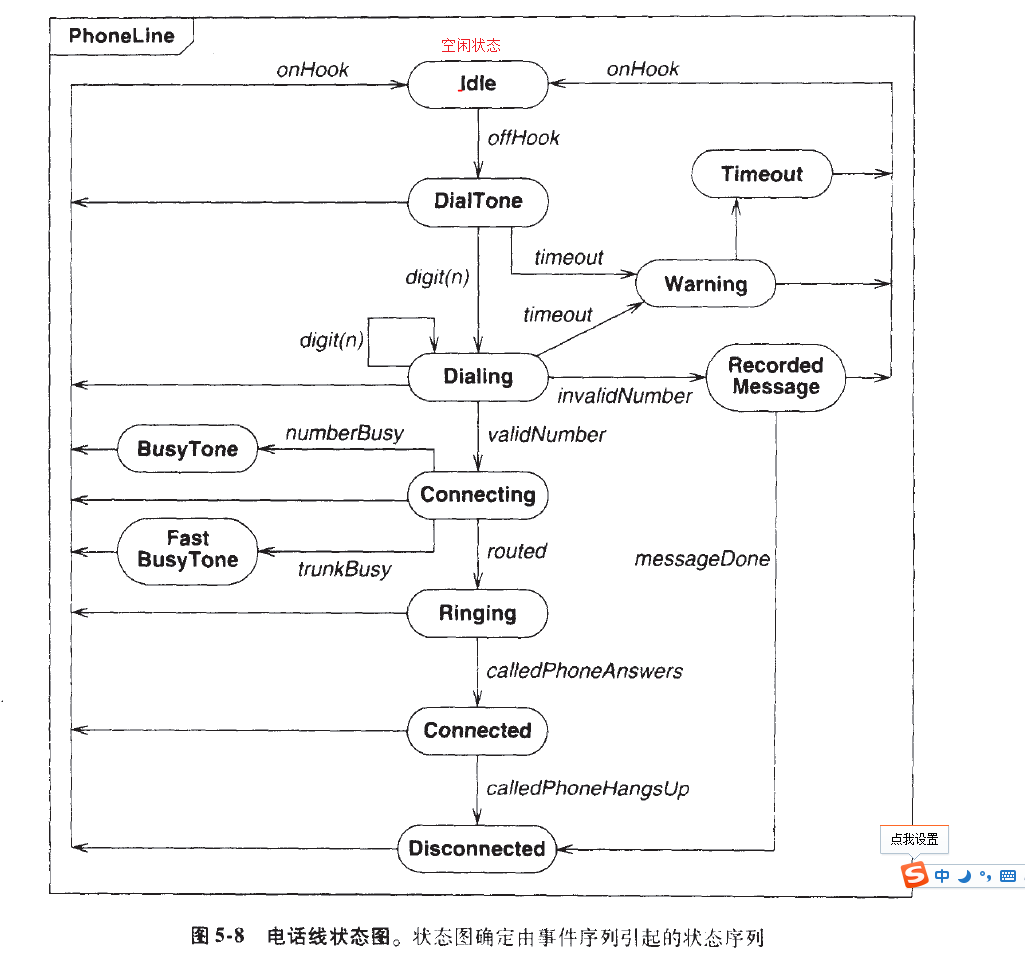
### 2.2.3 迁移

是指从一种状态到另一种状态的瞬时变化。

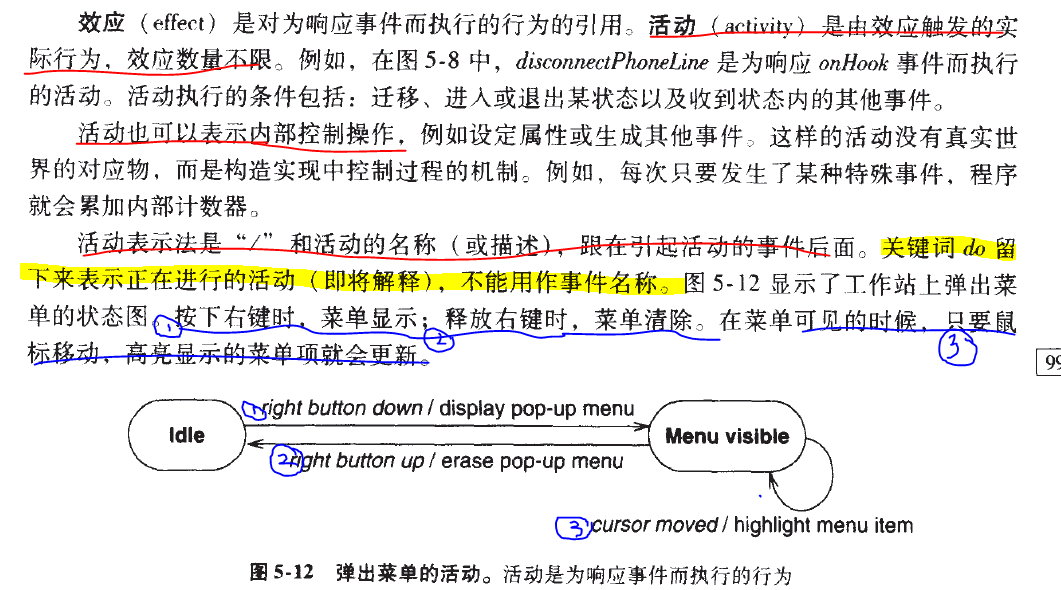




### 2.2.4 电话状态图

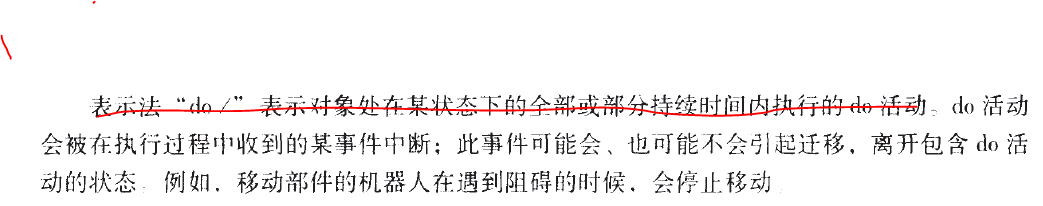


### 2.2.5 活动

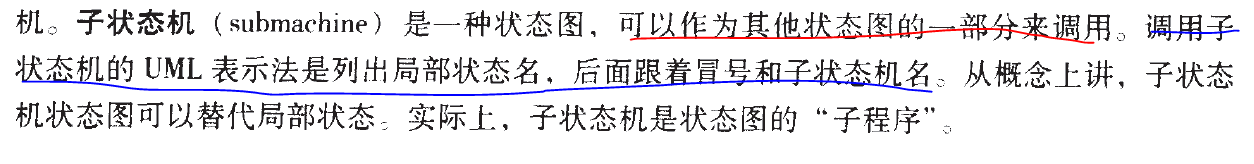


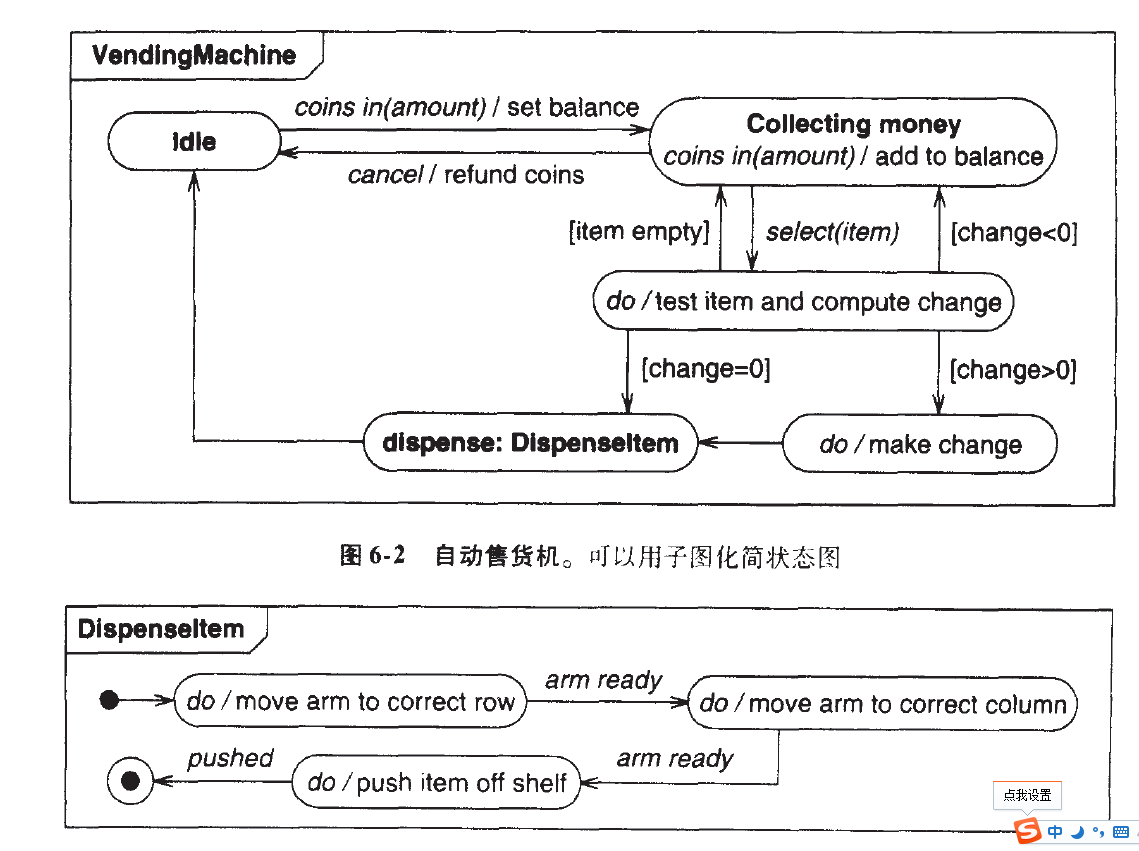
### 2.2.6 增加了活动的电话状态图





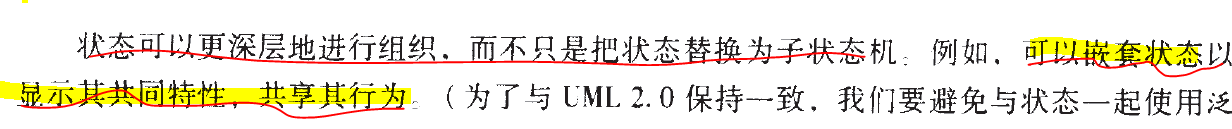
### 2.2.7 嵌套状态图

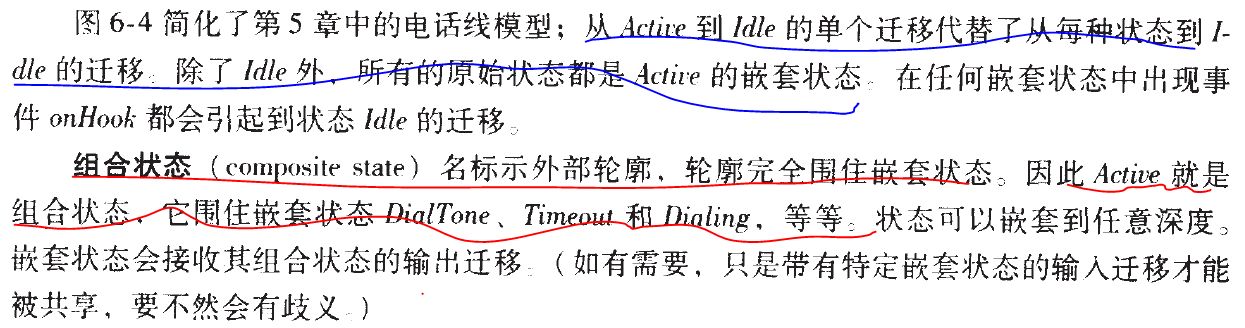


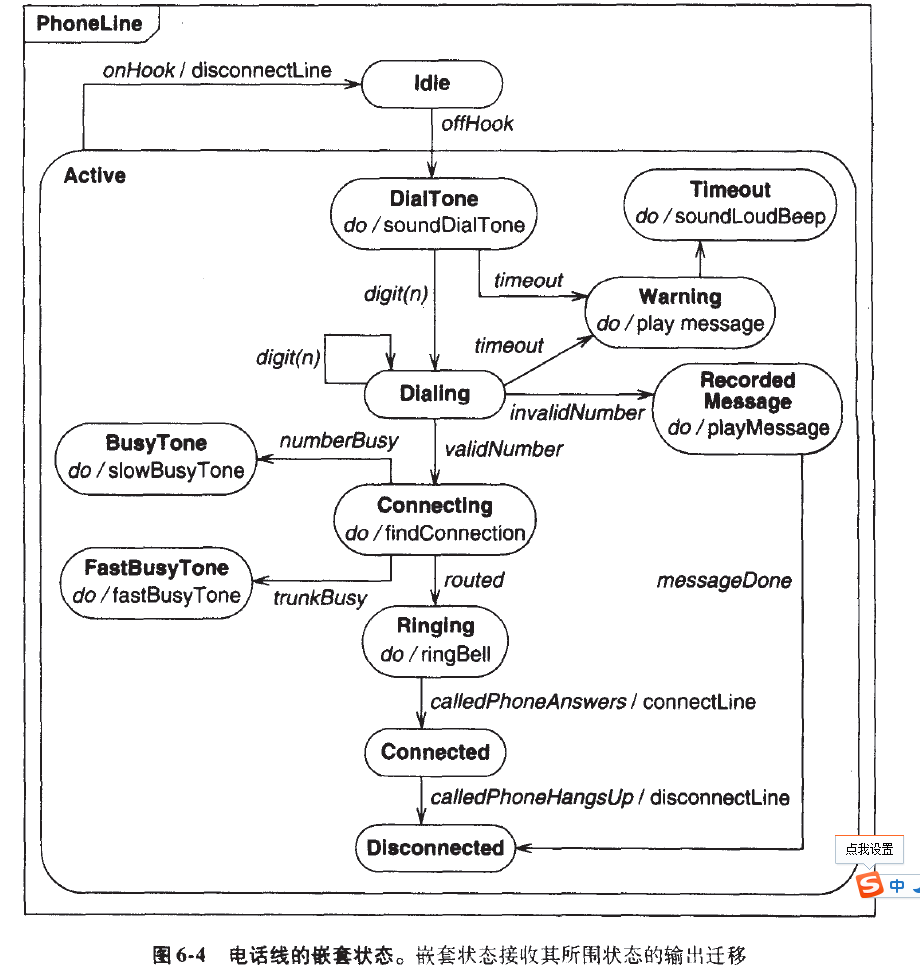




**嵌套状态**

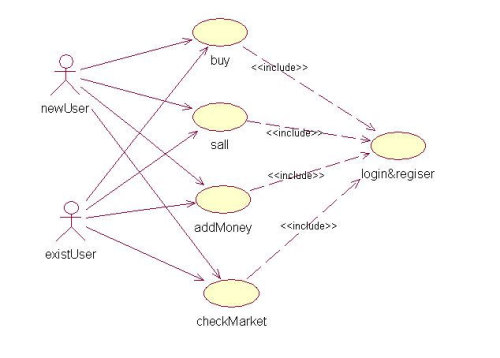






## 2.3 用例图

**描述一组用例、参与者以及它们之间的关系，其展示的是该系统在它的外面环境中所提供的外部可见服务**



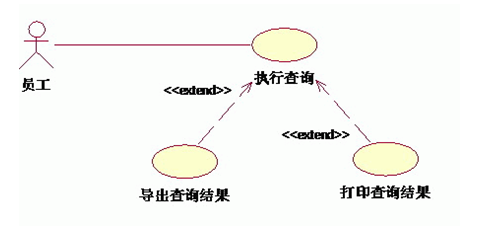
### 2.3.1 用例中的包含

包含关系：使用包含（Inclusion）用例来**封装一组跨越多个用例的相似动作（行为片断），以便多个基（Base）用例复用**。基用例控制与包含用例的 关系，以及被包含用例的事件流是否会插入到基用例的事件流中。基用例可以依赖包含用例执行的结果，但是双方都不能访问对方的属性。



### 2.3.2 扩展

**将基用例中一段相对独立并且可选的动作，用扩展（Extension）用例加以封装，再让它从基用例中声明的扩展点（Extension Point）上进行扩展，从而使基用例行为更简练和目标更集中**。扩展用例为基用例添加新的行为。扩展用例可以访问基用例的属性，因此它能根据基用例中扩展点的当前状态来判断是否执行自己。但是扩展用例对基用例不可见。

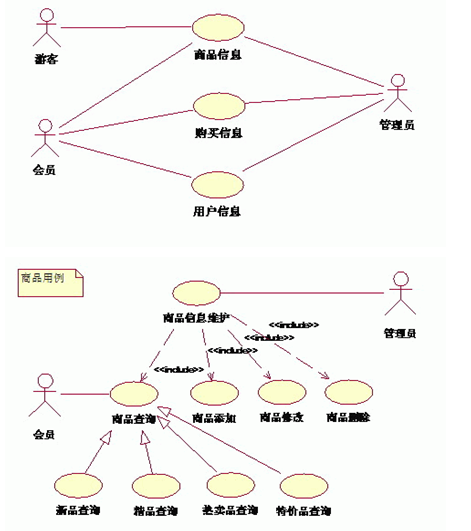


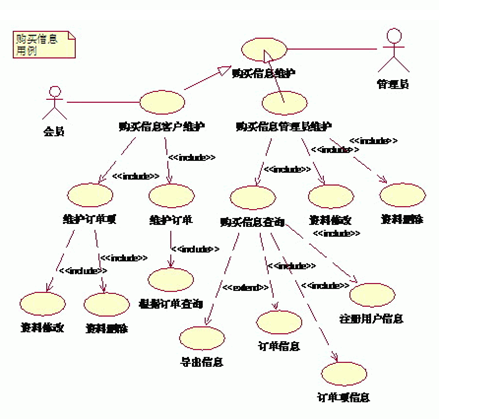
### 2.3.3 泛化

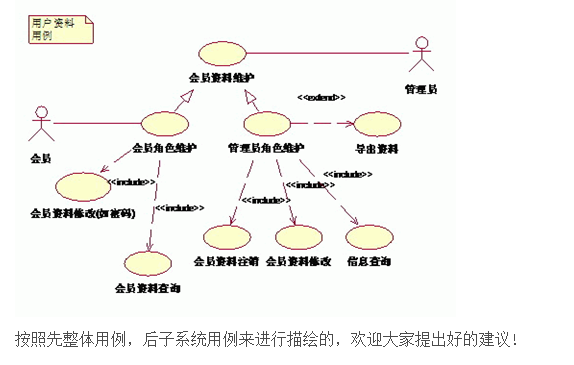
泛化关系：子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。在实际应用中很少使用泛化关系，子用例中的特殊行为都可以作为父用例中的备选流存在。



### 2.3.4 实例







## 2.4 交互图

**场景是指系统在某个特定的执行期内发生的一系列事件**。

包括序列图（顺序图）和协作图，两者对应，**顺序图是强调消息时间顺序，有对象生命线和控制焦点**。

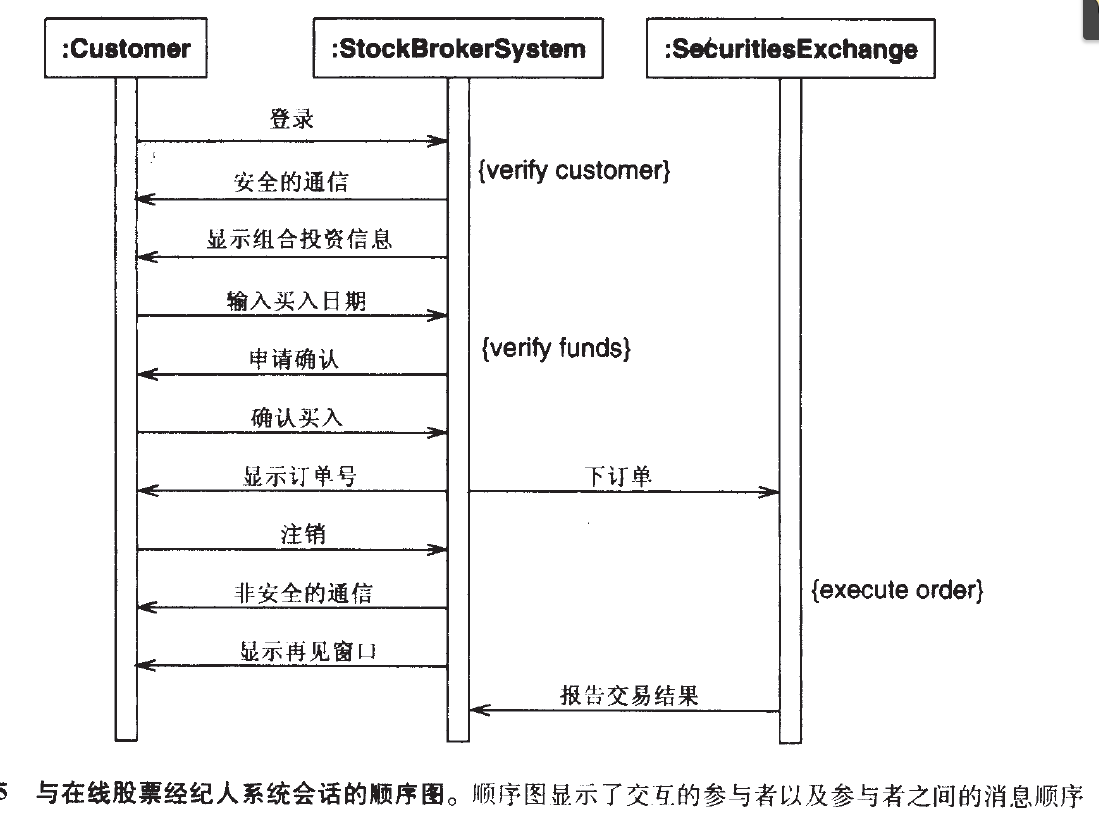
**协作图是强调接收和发送消息的对象的结构组织，有路径和顺序号**

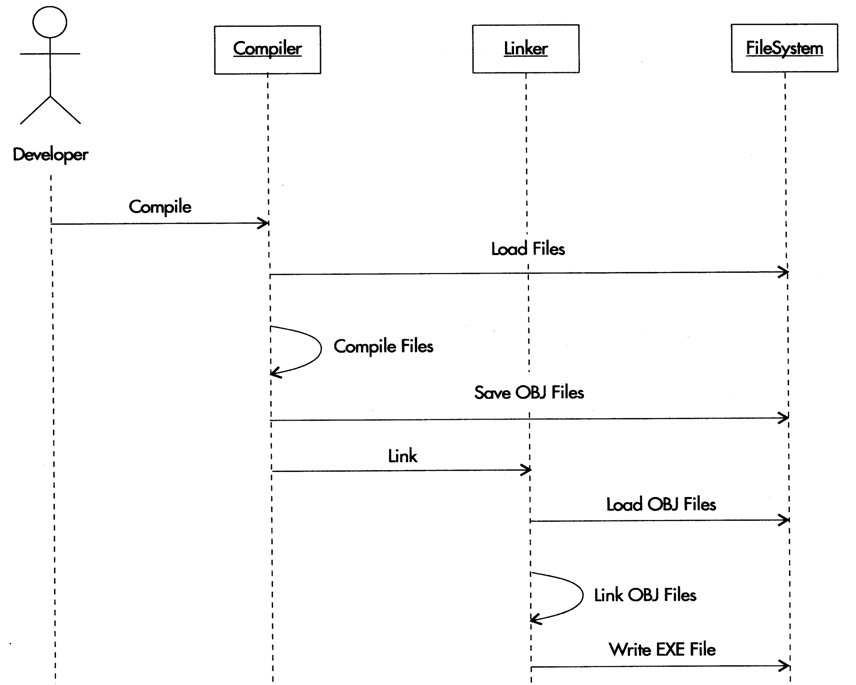
**交互图强调的是*对象到对象*的控制流，而活动图则强调的是从*活动到活动*的控制流**

•活动图是一种表述过程基理、业务过程以及工作流的技术。它可以用来对业务过程、工作流建模，也可以对用例实现甚至是程序实现来建模

### 2.4.1 顺序图

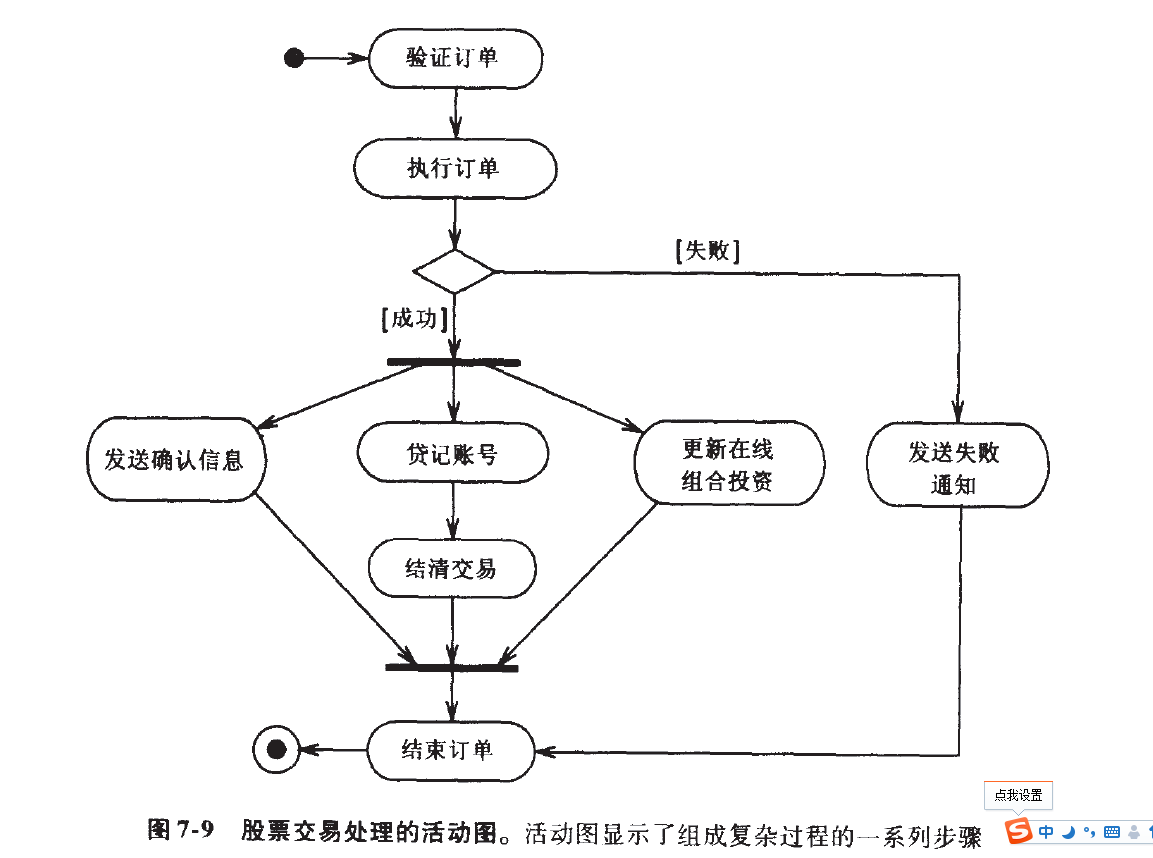
显示了交互的参与者以及参与者之间的消息顺序。也显示了系统为了执行全部或部分用例而与参与者的交互。





### 2.4.2 活动图

**显示了组成复杂过程的步骤序列。**



活动图的主要元素

•**初始节点和活动终点**：用一个**实心圆**表示初始节点，用一个**圆圈内加一个实心圆**来表示活动终点

•**活动节点**：是活动图中最主要的元素之一，它用来表示一个活动

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_6.png)

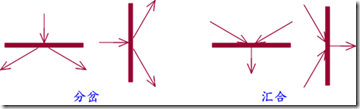
•转换：当一个活动结束时，控制流就会马上传递给下一个活动节点，在活动图中称之为“转换”，用一条**带箭头的直线**来表示

**活动图的主要元素**

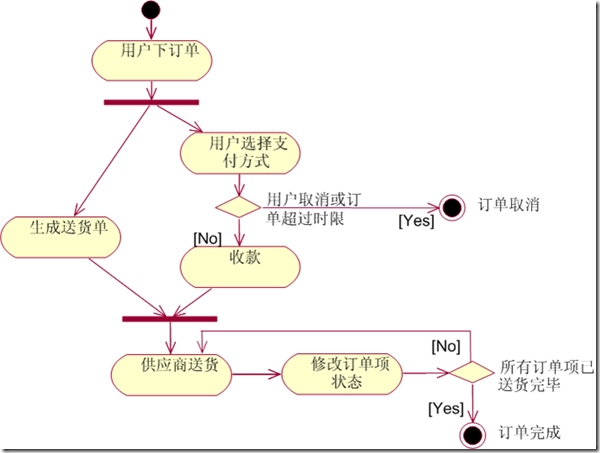
•**分支与监护条件**：**分支是用菱形表示的**，它有一个进入转换（箭头从外指向分支符号），一个或多个离开转换（箭头从分支符号指向外）。而**每个离开转换上都会有一个监护条件**，用来表示满足什么条件的时候执行该转换。

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_8.png)

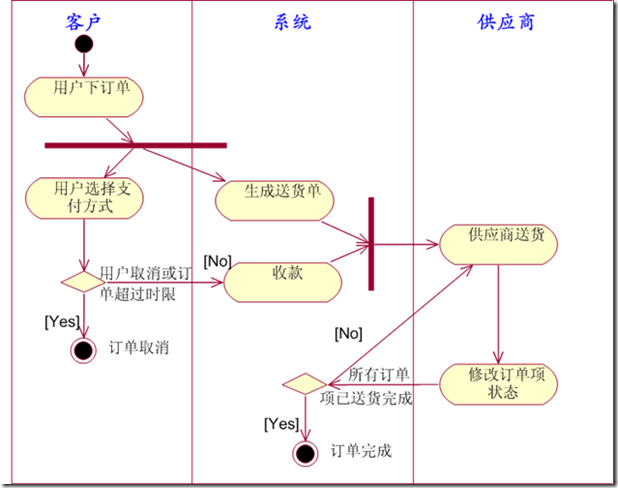
•**分岔与汇合**：

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_10.png)

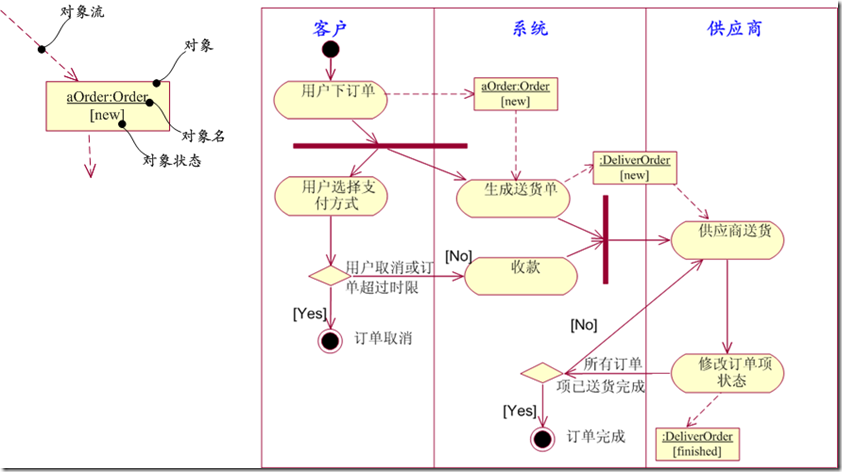
修改后的简单活动图

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_12.png)

**带泳道的活动图**

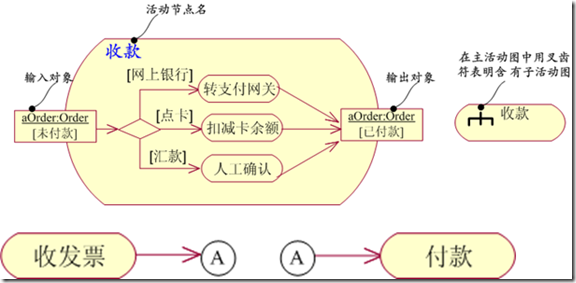
[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_14.png)

**带对象流的活动图**

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_16.png)

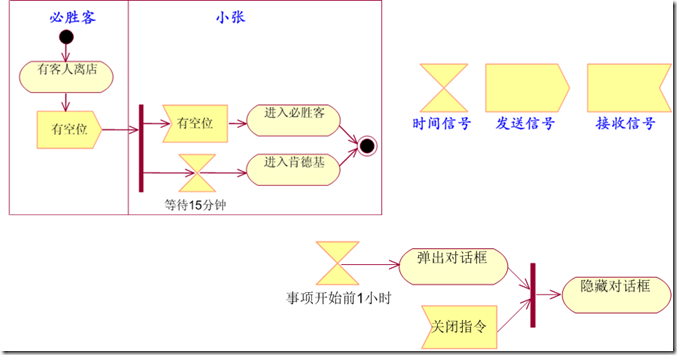
复杂活动图

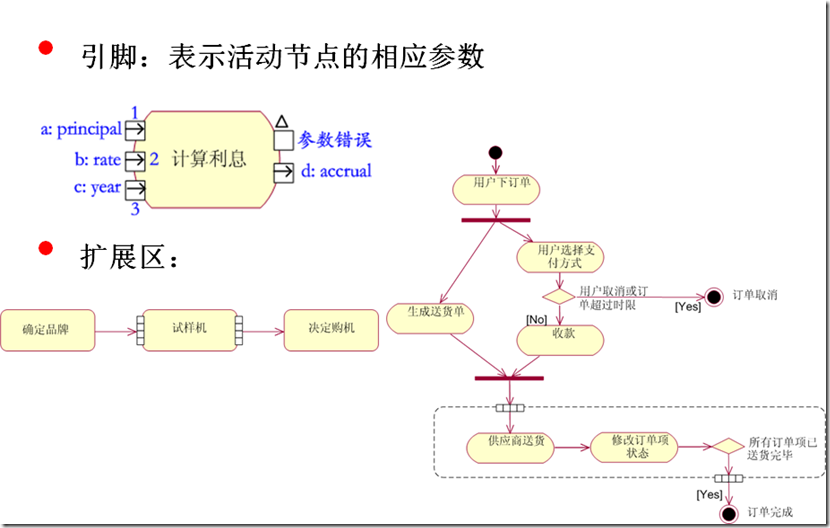
•**辅助活动图：**

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_18.png)

•汇合描述：当汇合的所有入流均到点汇合点时，就将执行汇合点指向的活动节点。但是有些时候，你希望对其做一些约束，这时就可以借助**汇合描述**来完成。**汇合描述实际上是一个约束，其格式就是“{约束条件}”。**

•发送信号与接收信号：

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_20.png)

[](http://images.cnblogs.com/cnblogs_com/TerryFeng/WindowsLiveWriter/UML_AFD4/image_22.png)

**•如何绘制活动图**

绘制活动图

•“活动图” 比较直观易懂；与传统的流程图十分的相近，只要能够读懂活动图，就不难画出活动图

•绘制时首先**决定是否采用泳道**：主要根据活动图中是否要**体现出活动的不同实施者**

•然后**尽量使用**分支、分岔和汇合等基本的建模元素来描述活动控制流程

•如果需要，加入对象流以及对象的状态变化，利用一些高级的建模元素（如辅助活动图、汇合描述、发送信号与接收信号、引脚、扩展区）来表示更多的信息

•活动图的建模**关键是表示出控制流**，其它的建模元素都是围绕这一宗旨所进行的补充

**工作流程，控制流程，业务流程中使用。**

• •活动图应用说明

活动图应用说明

•对工作流建模：用于业务建模的时候，每一条泳道表示一个职责单位，该图能够有效地体现出所有职责单位之间的**工作职责**，**业务范围及之间的交互关系**、信息流程

建模时应遵循以下策略：

•为工作流建立**一个焦点**，除非你所涉及的系统很小，否则不可能在一张图中显示出系统中所有的控制流

•选择对全部工作流中的一部分**有高层职责的业务对象**，并为每个重要的业务对象创建一条泳道

•识别工作流初始节点的**前置条件**和活动终点的**后置条件**，这可有效地实现对工作流的边界进行建模。

•从该工作流的初始节点开始，说明随时间发生的动作和活动，并在活动图中把它们表示成活动节点

•将**复杂的活动或多次出现的活动集合归到一个活动节点**，并通过辅助活动图或子活动图来表示它们

•找出连接这些活动节点的**转换**，**首先从工作流的顺序开始**，**然后考虑分支**，**接着再考虑分岔和汇合**

•如果工作流中**涉及重要的对象**，则也可以将它们加入到活动图中

•若工作流中有多次启用的，则可采用展开区表示

•**对操作建模：每一个对象占据一个泳道，而活动则是该对象的成员方法**

•建模时应遵循以下策略：   
-- 收集操作所涉及的抽象概念，包括操作的参数、返回类型、所属类的属性以及某些邻近的类   
-- 识别该操作的初始节点的前置条件和活动终点的后置条件。也要识别在操作执行过程中必须保持的信息   
-- 从该操作的初始节点开始，**说明随着时间发生的活动**，并在活动图中将它们表示为活动节点   
-- 如果需要，使用分支来说明条件语句及循环语句   
-- 仅当这个操作属于一个主动类时，才在必要时用分岔和汇合来说明并行的控制流程

构件图

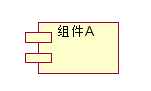
## 2.5 构件图

类是最基础的“模块化”元素，它封装了属性和成员的方法，就像是物理世界中的“分子”。但是，对于复杂的软件系统而言，往往拥有成百上千的各种类。因此，类的粒度太小了，引入更粗的粒度的概念—“构件”

**构件是系统中可替换的物理部分，它包装了实现而且遵从并提供一组接口的实现**。

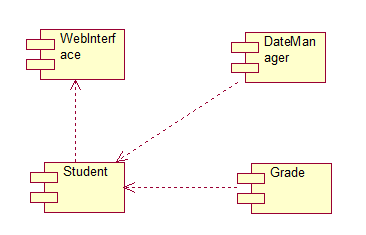
    通俗的说，**构件是系统设计的一个模块化元素，它隐藏了内部的实现，对外提供一组外部接口**。在系统中，**满足相同接口的组件可以自由地替换**。

1、构件的表示方法



和类的名称相近，构件的名称也是一个正文字符串，它可以是简单名，也可以是带路径的全名。

2、构件图实例：



## 2.6 部署图

1、  **部署图描述了一个系统运行时的硬件节点**，在这些节点上运行的软件构件将在何处物理运行以及它们将如何彼此通信的静态视图。

   部署图包括两种基本模型元素：节点和节点间的连接。每个模型中，仅包含一个部署图。**节点包括两种类型：处理器和设备。**

   处理器指本身具有计算能力且能执行各各软件的节点，如服务器。

处理器具有处理能力，所以在描述处理器方面应当包含了处理器的调度和进程。

调度指在处理器处理其进程中为实现一定的目的而对共同使用的资源进行时间分配。调度方式包含：抢占，无优先级，循环，算法控制，手动执行。进程表示一个单独的控制纯种，是系统中一个重量级的并发和执行单元。

   设备指本身不具备处理能力的节点，如打印机。

   连接用来表示两个节点之间的硬件连接。节点之间的连接可以通过光缆直接进行，或通过卫星等方式非直接连接，通常连接都是双向的。连接用实线表示，实线上可加连接名和构造型。

   系统开发人员和部署人员可以利用部署图去了解系统的物理运行情况。如果，开发的软件系统只需在一台计算机上运行，且使用的标准设备，则不需要为它画出系统部署图。部署图只需要给那些复杂的物理运行情况进行建模。

部署图显示了系统的硬件，安装在硬件上的软件，用于连接硬件的各种协议和中间件等。

2、 部署模型的目的：

描述一个具体应用的主要部署结构，通过对各种硬件，在硬件中的软件以及各种连接协议的显示，可以很好的描述系统是如何部署的；平衡系统运行时的计算资源分布；可以通过连接描述组织的硬件网络结构或者是嵌入式系统等具有多种硬件和软件相关的系统运行模型。

3、部署图实例：

