面向对象考点汇总

面向对象基本概念、面向对象分析、设计、测试。

UML建模：事务、关系、图、视图。

设计模式：创建型、结构型、行为型

对于设计模式考察最多，分为三类题型：
1.纯定义-直接考定义原文，通过关键字记忆理解 -最简单
2.文字场景-通过文字描述使用场景判断 -简单
3.图形场景-通过给出设计模式图判断 -有难度

案例考点：三种类，UML关系判断，图填空，设计模式应用。

设计模式

创建型设计模式	定义	记忆关键字
Abstract Factory 抽象工厂模式	提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类	抽象接口
Builder 构建器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示	类和构造分离
Factory Method 工厂方法模式	定义一个创建对象的接口，但由子类决定需要实例化哪一个类。使得子类实例化过程推迟	子类决定实例化
Prototype 原型模式	用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象	原型实例，拷贝
Singleton 单例模式	保证一个类只有一个实例，并提供一个访问它的全局访问点	唯一实例

口诀：单抽元件（建）厂

设计模式

结构型设计模式	定义	记忆关键字
Adapter 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	转换，兼容接口
Bridge 桥接模式	将类的抽象部分和它的实现部分分离开来，使它们可以独立的变化	抽象和实现分离
Composite 组合模式	将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性	整体-部分，树形结构
Decorator 装饰模式	动态的给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活	附加职责
Façade 外观模式	定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用	对外统一接口
Flyweight 享元模式	提供支持大量细粒度对象共享的有效方法	细粒度，共享
Proxy 代理模式	为其他对象提供一种代理以控制这个对象的访问	代理控制

口诀：外侨（桥）组员（元）戴（代）配饰。

设计模式

行为型设计模式	定义	记忆关键字
Chain of Responsibility 职责链模式	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求	传递请求、职责、链接
Command 命令模式	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作	日志记录、可撤销
Interpreter 解释器模式	给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子	解释器，虚拟机
Iterator 迭代器模式	提供一种方法来顺序访问一个聚合对象中的各个元素而不需要暴露该对象的内部表示	顺序访问，不暴露内部
Mediator 中介者模式	用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立的改变对象间的交互	不直接引用

口诀：观摩（模）对（迭）策，责令解放（访），戒（介）忘台（态）。

设计模式

行为型设计模式	定义	记忆关键字
Memento 备忘录模式	在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态	保存，恢复
Observer 观察者模式	定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新	通知、自动更新
State 状态模式	允许一个对象在其内部状态改变时改变它的行为	状态变成类
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	算法替换
Template Method 模板方法模式	定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤	
Visitor 访问者模式	表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作。	数据和操作分离

考试真题

35-37、创建型模式支持对象的创建，该模式允许在系统中创建对象，而不需要在代码中标识出特定的类型，这样用户就不需要编写一些列相关或相互依赖的对象在不指定具体类的情况下。（　）模式为创建一系列相关或相互依赖的对象提供了一个接口，（　）模式将复杂对象的构建与其表面相分离，这样相同的构造过程可以创建不同的对象；（　）模式允许对象在不了解要创建对象的确切类以及如何创建细节的情况下创建自定义对象。

A、prototyke　　B、Abstract Factoty　　C、Builder　　D、Singleron
A、prototyke　　B、Abstract Factoty　　C、Builder　　D、Singleron
A、 prototyke　　B、Abstract Factoty　　C、Builder　　D、Singleron

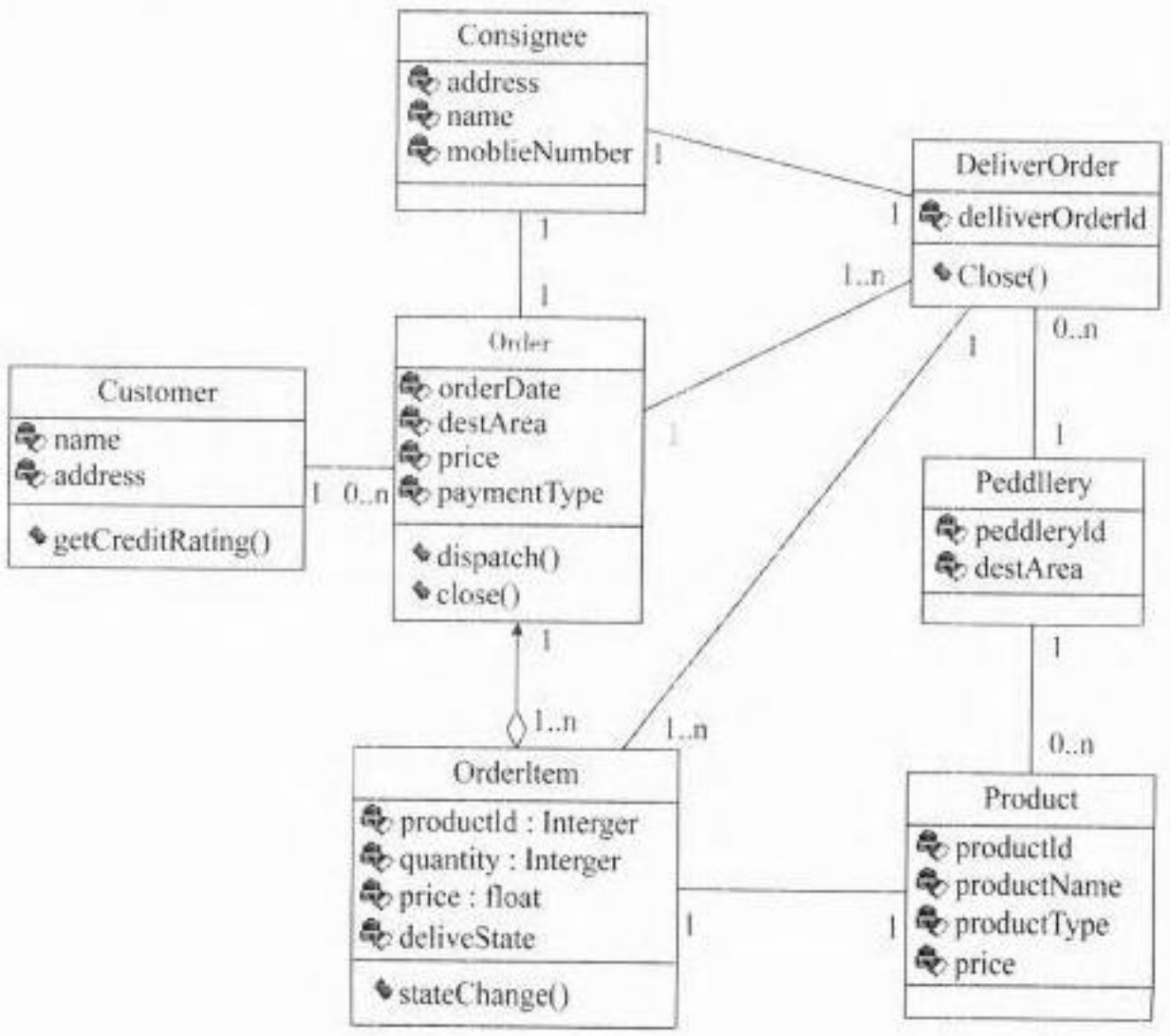
54-57. 设计模式按照目的可以划分为三类，其中，（54）模式是对对象实例化过程的抽象。例如（55）模式确保一个类只有一个实例，并提供了全局访问入口；（56）模式允许对象在不了解要创建对象的确切类以及如何创建等细节的情况下创建定义对象；（57）模式将复杂对象的构建与其表示分离。

A. 创建型　　B. 结构型　　C. 行为型　　D. 功能型
A. Facade　　B. Builder　　C. Prototype　　D. Singleton
A. Facade　　B. Builder　　C. Prototype　　D. Singleton
A. Facade　　B. Builder　　C. Prototype　　D. Singleton

考试真题

44-45. 一个完整的软件系统需从不同视角进行描述，下图属于软件架构设计中的 （44） ，用于（45）视图来描述软件系统。

A. 对象图　　B. 时序图
C. 构件图　　D. 类图
A. 进程　　B. 开发
C. 物理　　D. 用户



知识点

逻辑视图。主要支持系统的功能需求，即系统提供给最终用户的服务。

开发视图。也称为模块视图，在UML中被称为实现视图，它主要侧重于软件模块的组织和管理。通过系统I/O关系的模型图和子系统图来描述，其中，类图属于开发视图。

进程视图。侧重于系统的运行特性，主要关注一些非功能性需求，例如系统的性能和可用性等。强调并发性、分布性、系统集成性和容错能力。定义了逻辑视图中的各个类的操作具体是在哪一个线程中被执行的。

物理视图。在UML中被称为部署视图，它主要考虑如何把软件映射到硬件上，它通常要考虑到解决系统拓扑结构、系统安装和通信等问题。

场景。可以看作是那些重要系统活动的抽象，它使四个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。对于于UML中的用例视图。

- 逻辑视图（logicalview），设计的对象模型（使用面向对象的设计方法时）。
- 过程视图（processview），捕捉设计的并发和同步特征。
- 物理视图（physicalview），描述了软件到硬件的映射，反映了分布式特性。
- 开发视图 Cdevelopmentview），描述了在开发环境中软件的静态组织结构。

考试真题

体系结构模型的多视图表示是从不同的视角描述特定系统的体系结构。著名的 4+1 模型支持从（44）描述系统体系结构。

A.逻辑视图、开发视图、物理视图、进程视图、统一的场景
B.逻辑视图、开发视图、物理视图、模块视图、统一的场景
C.逻辑视图、开发视图、构件视图、进程视图、统一的场景
D.领域视图、开发视图、构件视图、进程视图、统一的场景

面向对象的分析模型主要由顶层架构图、用例与用例图和（ ）构成：设计模型则包含以（ ）表示的软件体系机构图、以交互图表示的用例实现图、完整精确的类图、描述复杂对象的（ ）和用以描述流程化处理过程的活动图等。

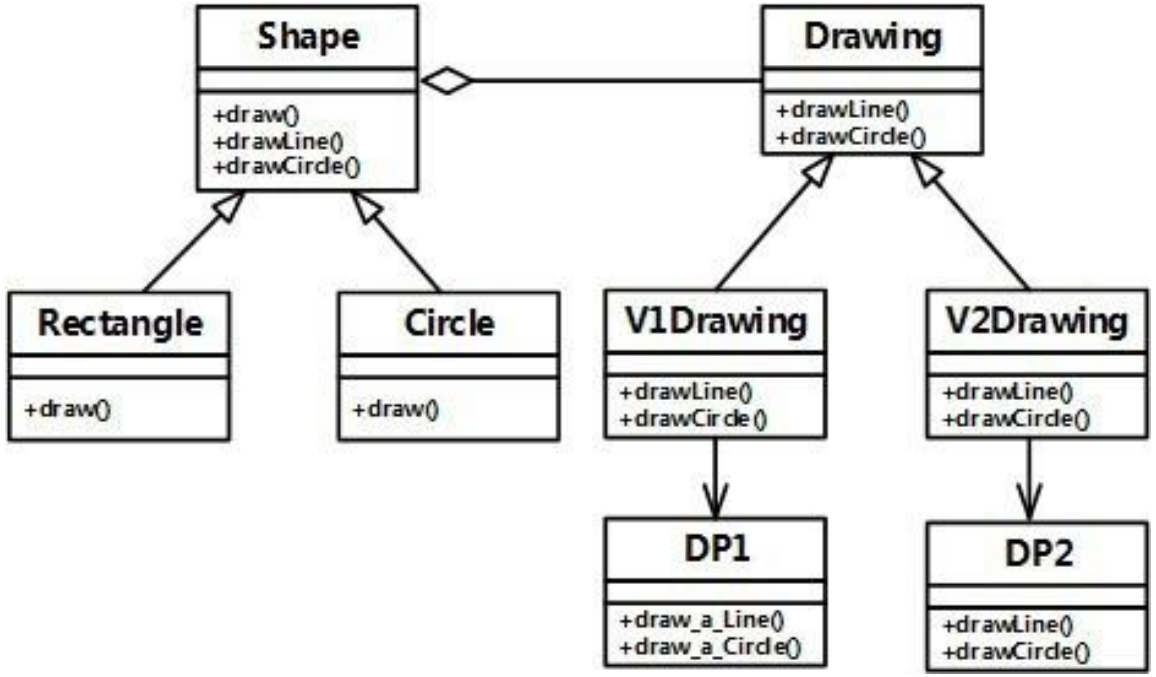
A. 数据流模型　　B.领域概念模型　C.功能分解图　　D.功能需求模型
A. 模型视图控制器　　B.组件图　　C.包图　　D.2层、3层或N层
A. 序列图　　B.协作图　　C.流程图　　D.状态图

用例（use case）用来描述系统对事件做出响应时所采取的行动。用例之间是具有相关性的。在一个会员管理系统中，会员注册时可以采用电话和邮件两种方式。用例“会员注册”和“电话注 册”、“邮件注册”之间是（ ）关系。

A. 包含（include） B.扩展(extend) C.泛化（generalize） D.依赖（depends on）

考试真题

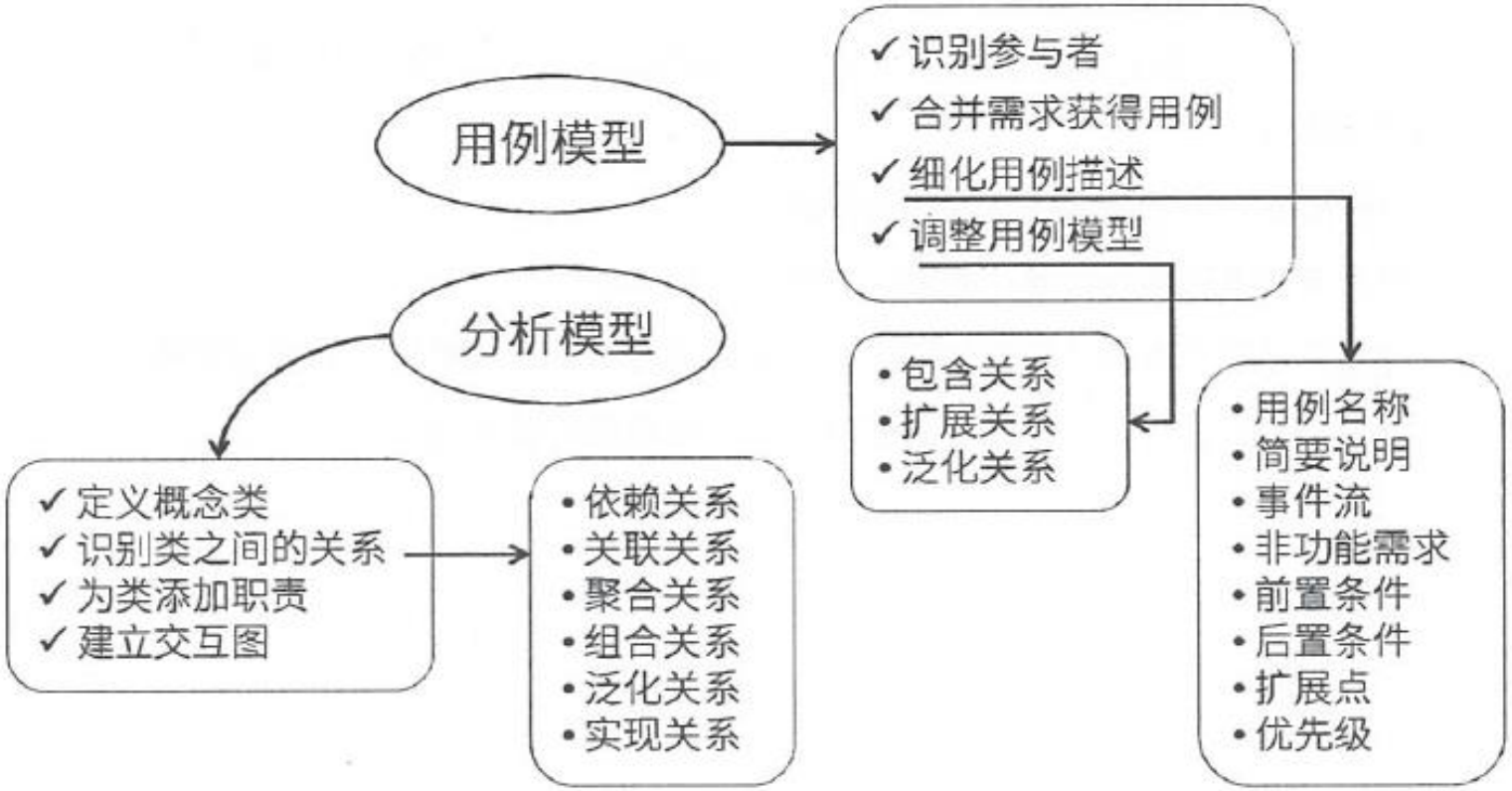
某软件公司欲开发一个绘图软件，要求使用不同的绘图程序绘制不同的图形。在明确用户需求后， 该公司的架构师决定采用Bridge模式实现该软件，并设计UML类图如下图所示。图中与Bridge模式 中的“Abstraction”角色相对应的类是（ ），与“Implementor”角色相对应的类是（ ）。



A. Shape　　B.Drawing
C.Rectangle　　D.V2Drawing
A. Shape　　B.Drawing
C.Rectangle　　D.V2Drawing

知识点

面向对象需求分析



知识点

面向对象的设计原则：
（1）单一责任原则。就一个类而言，应该仅有一个引起它变化的原因。即，当需要修改某个类的时候原因有且只有一个，让一个类只做一种类型责任。
（2）开放－封闭原则。软件实体（类、模块、函数等）应该是可以扩展的，即开放的；但是不可修改的，即封闭的。
（3）里氏替换原则。子类型必须能够替换掉他们的基类型。即，在任何父类可以出现的地方，都可以用子类的实例来赋值给父类型的引用。
（4）依赖倒置原则。抽象不应该依赖于细节，细节应该依赖于抽象。即，高层模块不应该依赖于低层模块，二者都应该依赖于抽象。
（5）接口分离原则。不应该强迫客户依赖于它们不用的方法。接口属于客户，不属于它所在的类层次结构。即：依赖于抽象，不要依赖于具体，同时在抽象级别不应该有对于细节的依赖。这样做的好处就在于可以最大限度地应对可能的变化。
上述（1）～（5）是面向对象方法中的五大原则。除了这五大原则之外，Robert C. Martin提出的面向对象设计原则还包括以下几个。
（6）重用发布等价原则。重用的粒度就是发布的粒度。
（7）共同封闭原则。包中的所有类对于同一类性质的变化应该是共同封闭的。一个变化若对一个包产生影响，则将对该包中的所有类产生影响，而对于其他的包不造成任何影响。
（8）共同重用原则。一个包中的所有类应该是共同重用的。如果重用了包中的一个类，那么就要重用包中的所有类。
（9）无环依赖原则。在包的依赖关系图中不允许存在环，即包之间的结构必须是一个直接的五环图形。
（10）稳定依赖原则。朝着稳定的方向进行依赖。
（11）稳定抽象原则。包的抽象程度应该和其稳定程度一致。

考试真题

在面向对象设计的原则中、（ ）原则是指抽象不应该依赖予细节，细节应该依赖于抽象，即应针对接口编程，而不是针对实现编程。
A．开闭 B.里氏替换 C.最少知识 D.依赖倒置

在面向对象设计中，（ ）可以实现界面控制、外部接口和环境隔离。（ ）作为完成用例业务的责任承担者，协调、控制其他类共同完成用例规定的功能或行为。
A．实体类 B.控制类 C.边界类 D.交互类
A．实体类 B.控制类 C.边界类 D.交互类

基于UML的需求分析过程的基本步骤为：利用（ ）表示需求；利用（ ）表示目标软件系统的总体架构。
A．用例及用例图 B.包图及类图 C.剧情及序列图 D.组件图及部署图
A．用例及用例图 B.包图及类图 C.剧情及序列图 D.组件图及部署图

UML关系

◆**依赖**：一个事物的语义依赖于另一个事物的语义的变化而变化
◆**关联**：是一种结构关系，描述了一组链，链是对象之间的连接。分为组合和聚合，都是部分和整体的关系，其中组合事物之间关系更强。两个类之间的关联，实际上是两个类所扮演角色的关联，因此，两个类之间可以有多个由不同角色标识的关联。
◆**泛化**：一般/特殊的关系，子类和父类之间的关系
◆**实现**：一个类元指定了另一个类元保证执行的契约。

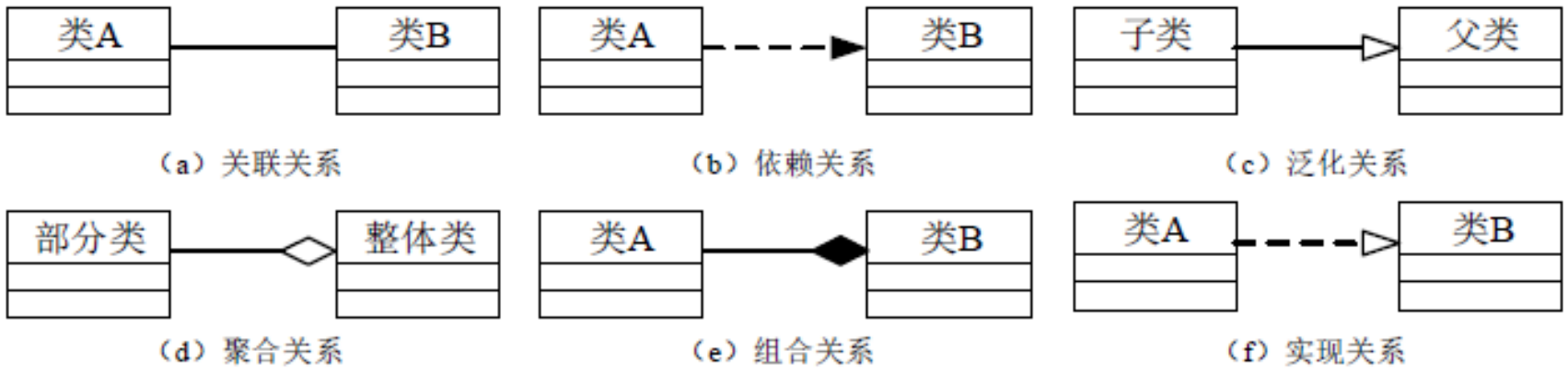


图 11-14 类之间的关系表示

UML图

◆**类图**：静态图，为系统的静态设计视图，展现一组对象、接口、协作和它们之间的关系。UML类图如下：

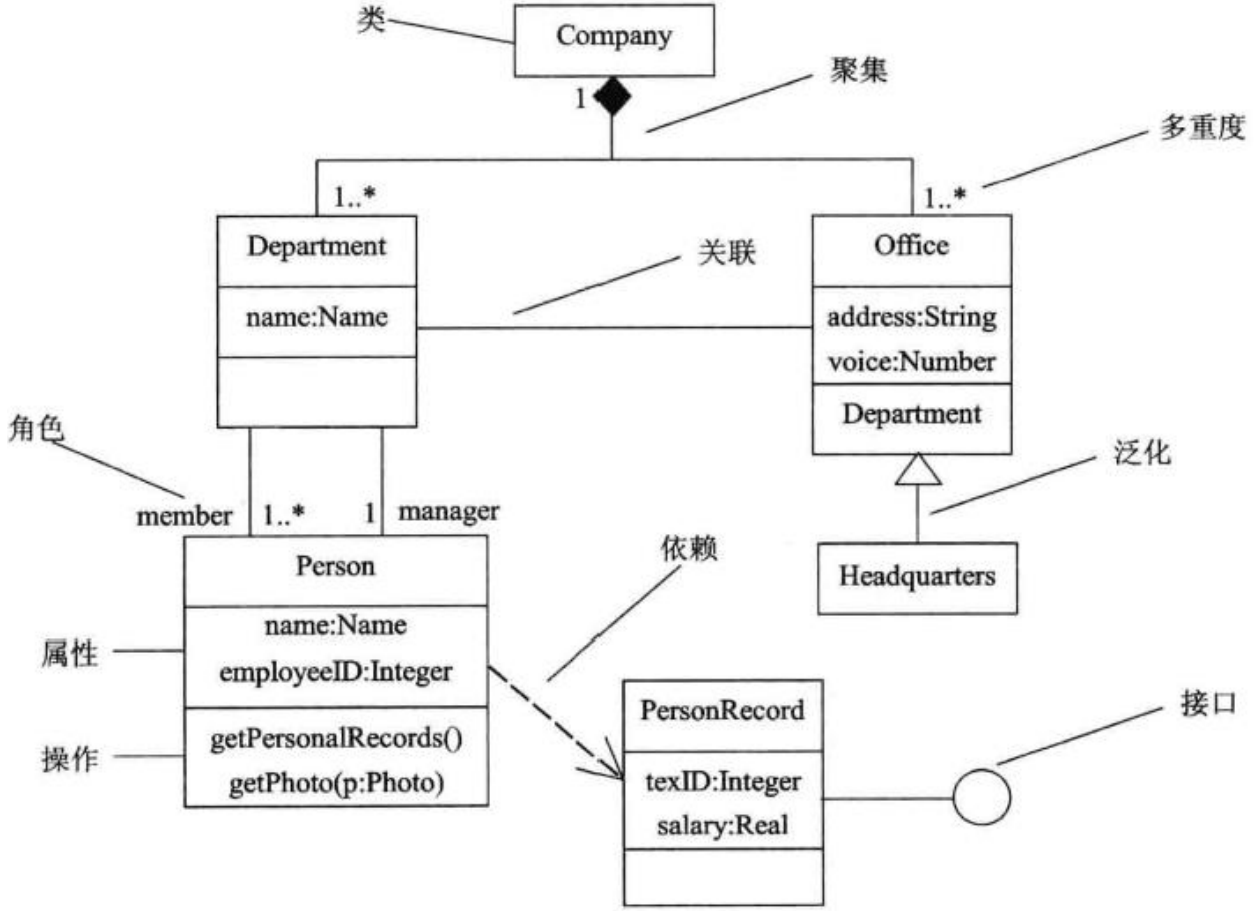


图 10-11　UML 类图

UML图

◆**用例图**：静态图，展现了一组用例、参与者以及它们之间的关系。用例图中的参与者是人、硬件或其他系统可以扮演的角色；用例是参与者完成的一系列操作，用例之间的关系有扩展、包含、泛化。如下：

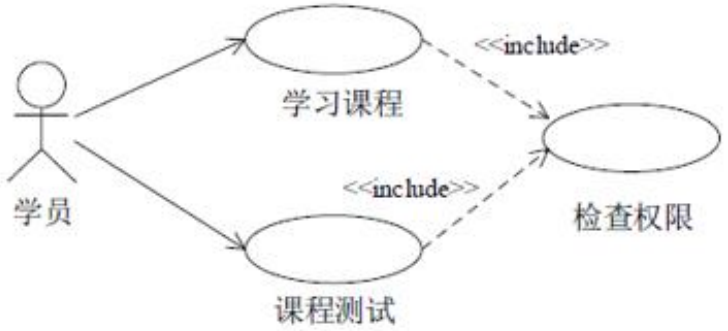


图 11-11 包含关系的例子

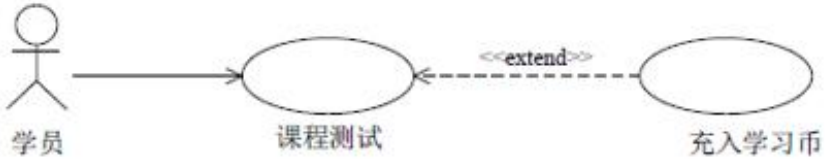
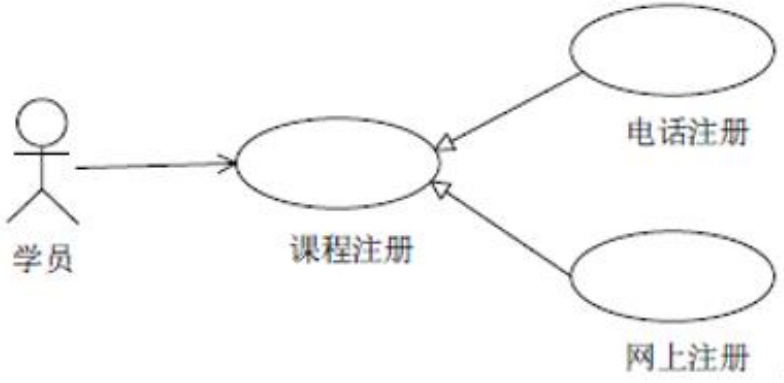
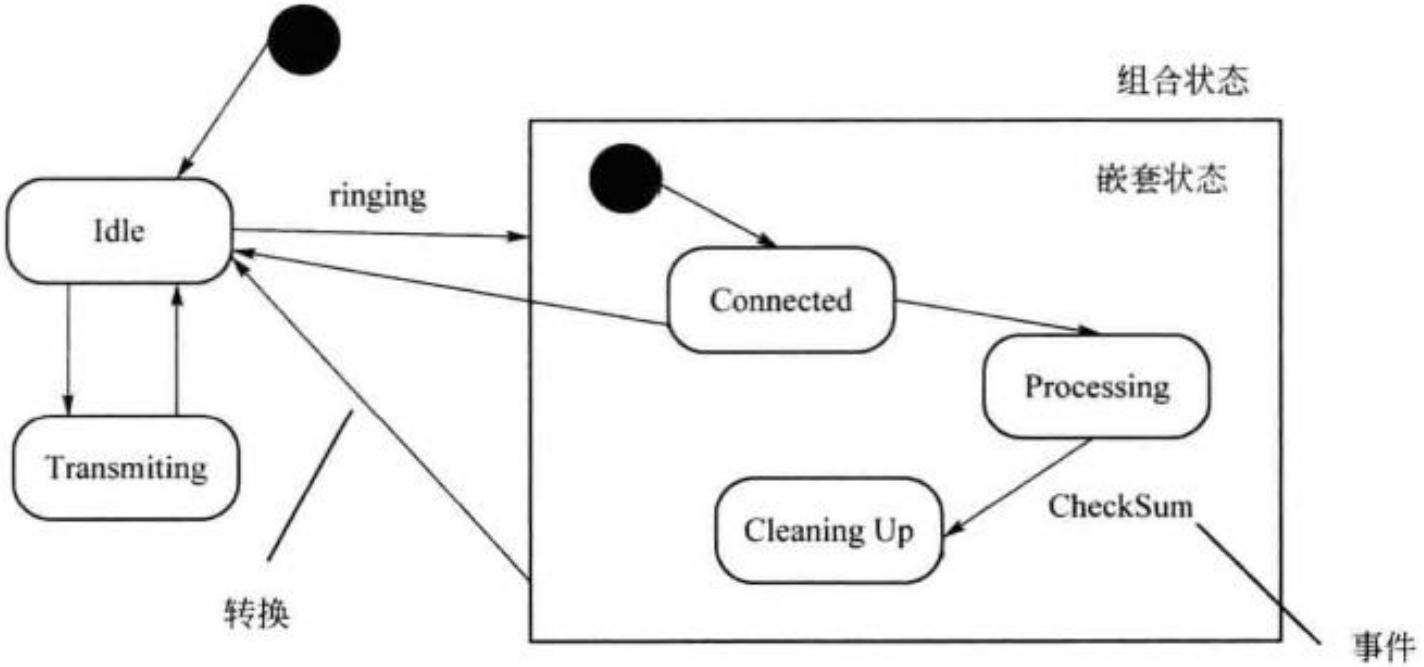


图 11-12 扩展关系的例子

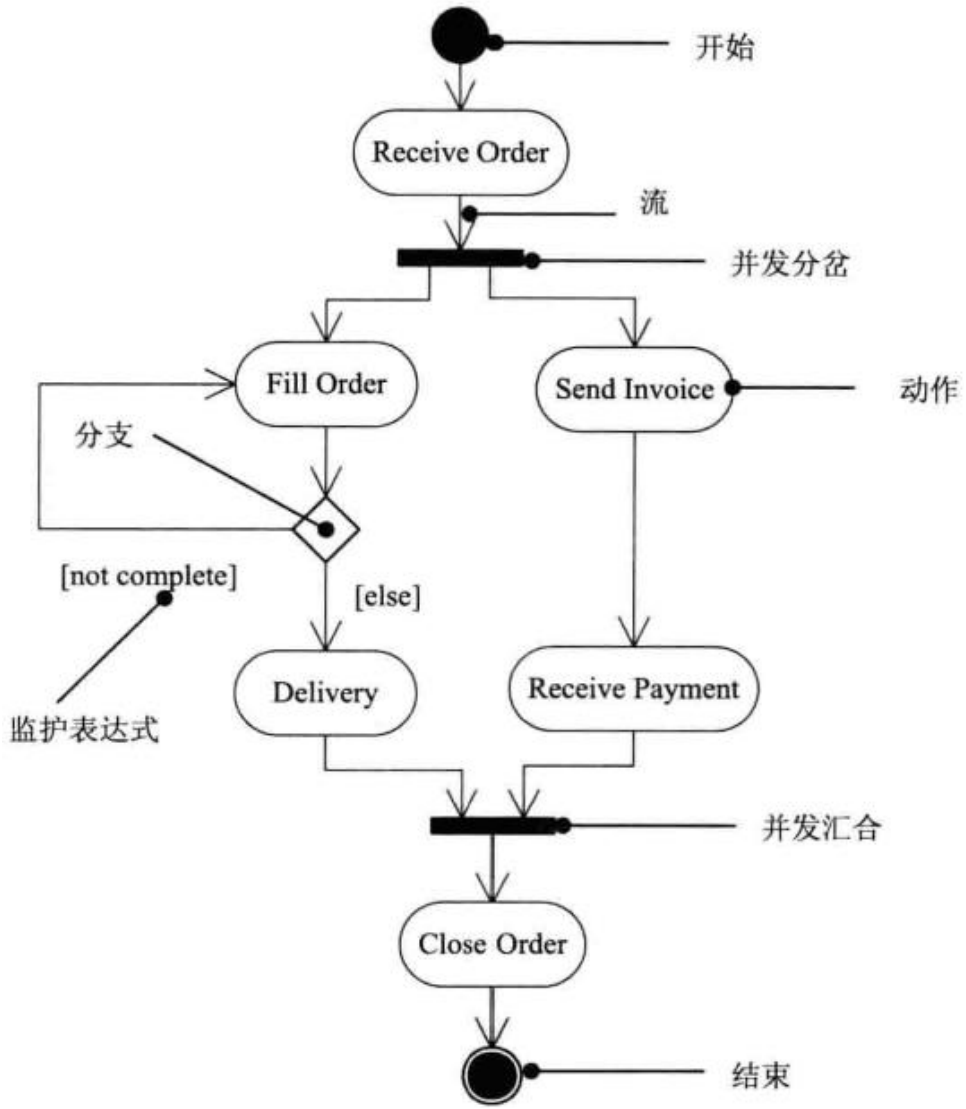


UML图

◆**状态图**：动态图，展现了一个状态机，描述单个对象在多个用例中的行为，包括简单状态和组合状态。转换可以通过事件触发器触发，事件触发后相应的监护条件会进行检查。状态图中转换和状态是两个独立的概念，如下：图中方框代表状态，箭头上的代表触发事件，实心圆点为起点和终点。



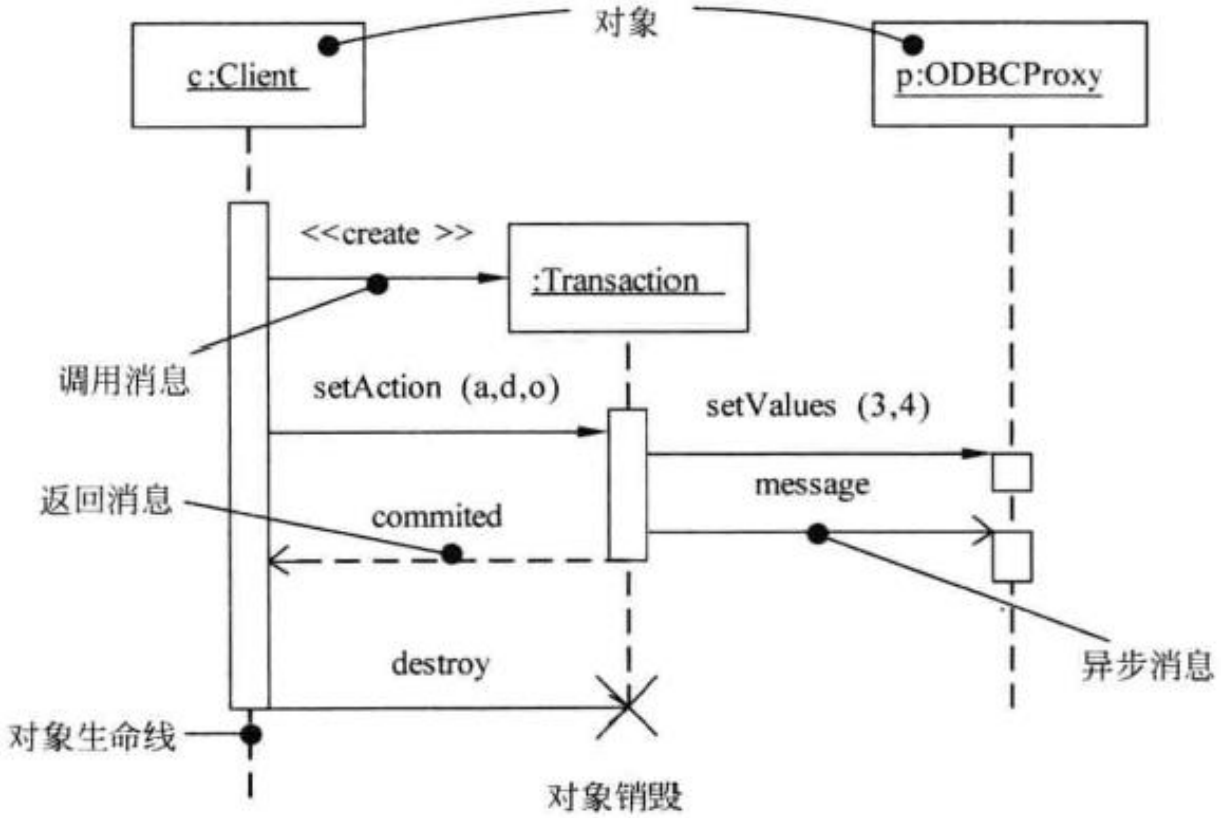
UML图



◆**活动图**：动态图，是一种特殊的状态图，展现了在系统内从一个活动到另一个活动的流程。活动的分岔和汇合线是一条水平粗线。牢记下图中并发分岔、并发汇合、监护表达式、分支、流等名词及含义。每个分岔的分支数代表了可同时运行的线程数。活动图中能够并行执行的是在一个分岔粗线下的分支上的活动

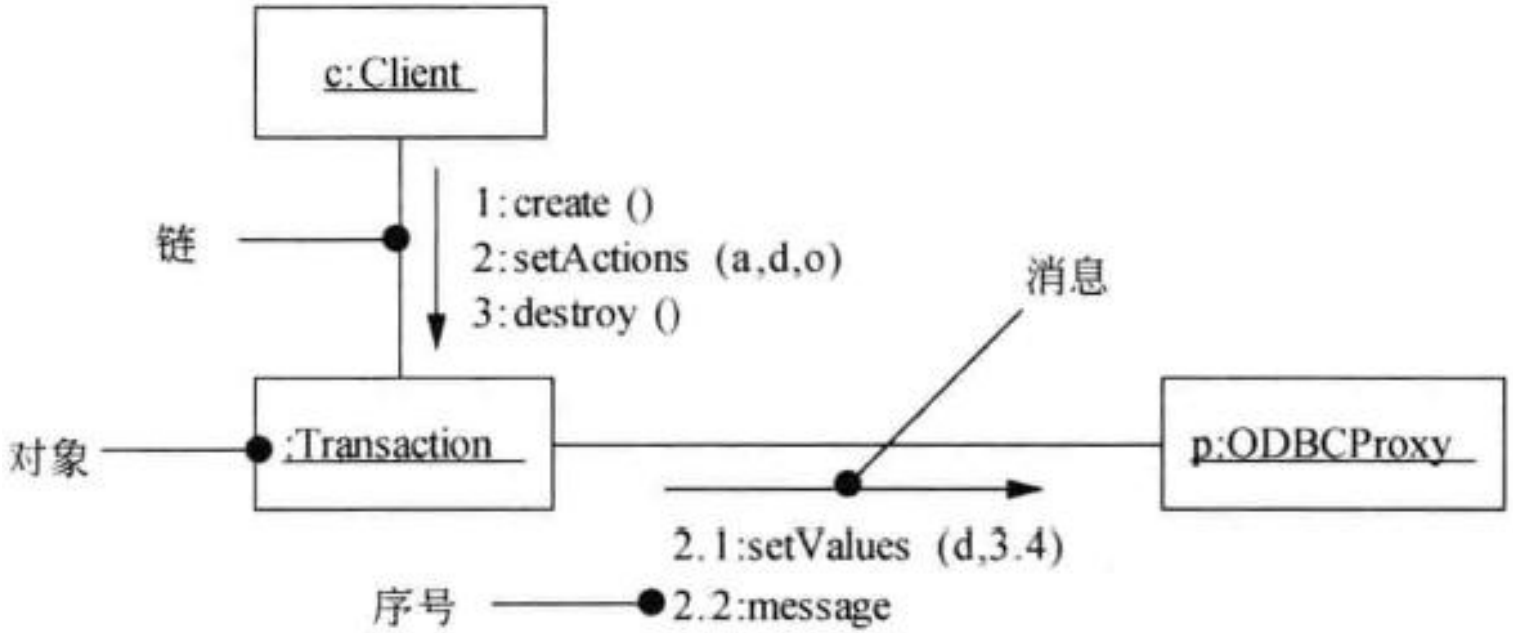
UML图

◆**序列图**：即顺序图，动态图，是场景的图形化表示，描述了以时间顺序组织的对象之间的交互活动。有**同步消息**（进行阻塞调用，调用者中止执行，等待控制权返回，需要等待返回消息，用实心三角箭头表示），**异步消息**（发出消息后继续执行，不引起调用者阻塞，也不等待返回消息，由空心箭头表示）、**返回消息**（由从右到左的虚线箭头表示）三种。如下：



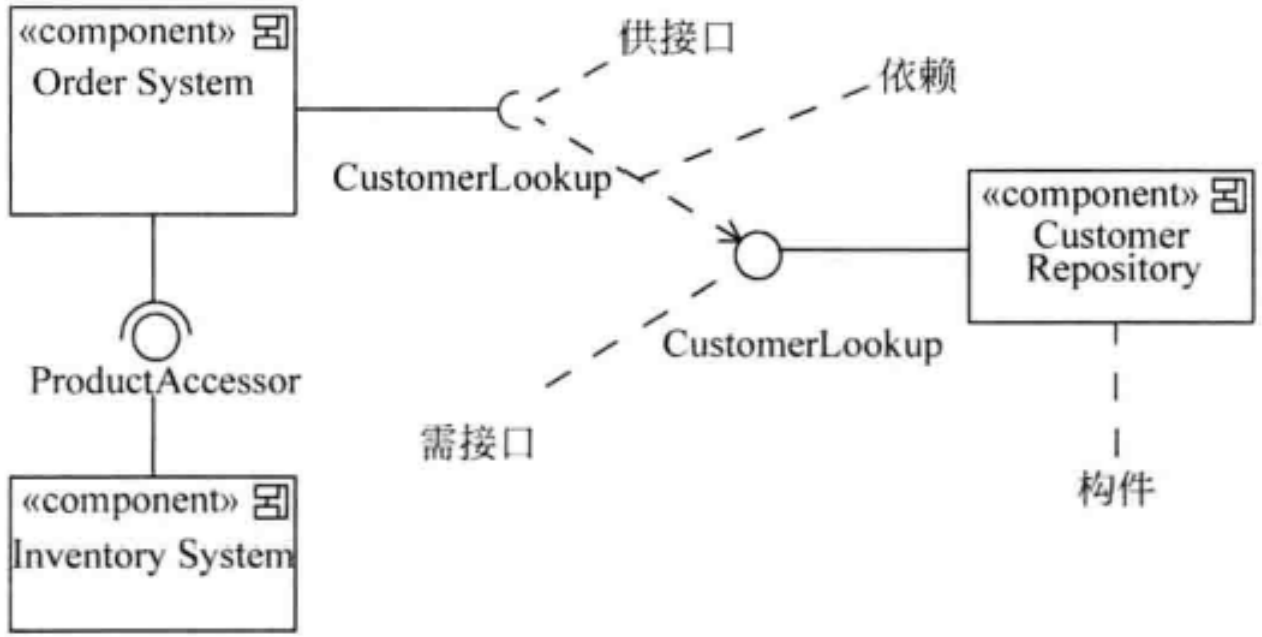
UML图

◆**通信图**：动态图，即协作图，强调参加交互的对象的组织。如下：



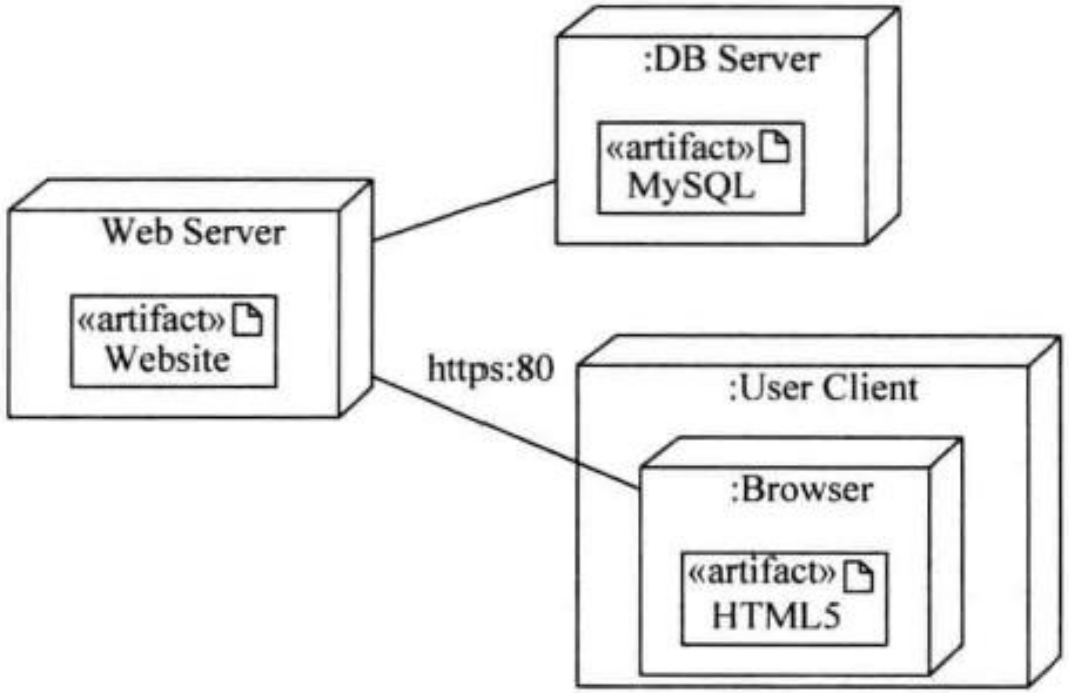
UML图

◆**构件图（组件图）**：静态图，为系统静态实现视图，展现了一组构件之间的组织和依赖。如下：



UML图

◆**部署图**：静态图，为系统静态部署视图，部署图物理模块的节点分布。它与构件图相关，通常一个结点包含一个或多个构件。其依赖关系类似于包依赖，因此部署组件之间的依赖是单向的类似于包含关系。如下：



案例真题

阅读以下关于软件系统设计与建模的叙述，在答题纸上回答问题 1 至问题 3.

【说明】

某医院拟委托软件公司开发一套预约挂号管理系统，以便为患者提供更好的就医体验，为医院提供更加科学的预约管理。本系统的主要功能描述如下:

(a)注册登录，(b)信息浏览，(c)账号管理，(d)预约挂号，(e)查询与取清预约，(F)号源管理，(g)报告查询，(h)预约管理，(i)报表管理和(j)信用管理等。

【问题 1】(6 分)

若采用面向对象方法对预约挂号管理系统进行分析，得到如图 2-1 所示的用例图。请将合适的参与者名称填入图 2-1中的(1)和(2)处，使用题干给出的功能描述(a)~(j)，完善用例(3)~(12)的名称，将正确答案填在答题纸上。

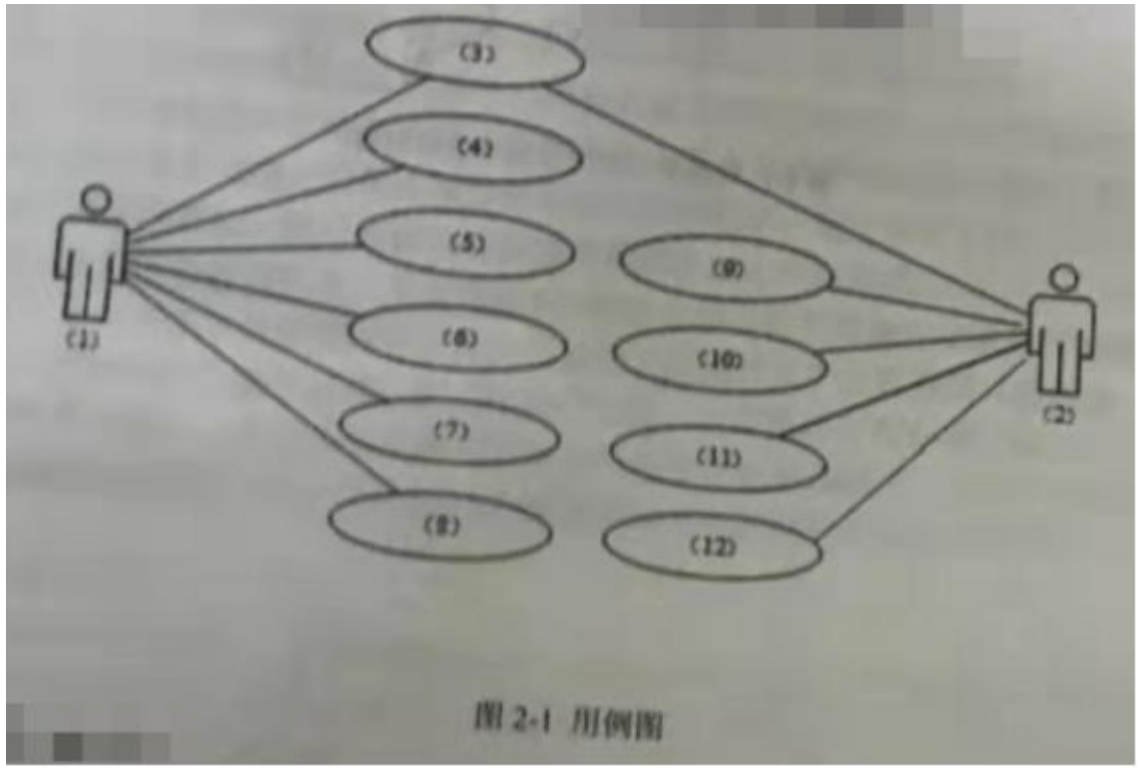


图 2-1 用例图

案例真题

【问题 2】(10分)

预约人员(患者)登录系统后发起预约挂号请求，进入预约界面。进行预约挂号时使用数据库访问类获取医生的相关信息，在数据库中调用医生列表，并调取医生出诊时段表，将医生出诊时段反馈到预的界面，并显示给预约人员;预约人员选择医生及就诊时间后确认预的，系统返回网预约结果，并向用户显示是否预约成功。

采用面向对象方法对预约挂号过程进行分析，得到如图 2-2 所示的顺序图，使用题干中给出的描述，完善图 2-2 中对象(1)，及消息(2)~(4)的名称，将正确答案填在普题纸上请简要说明在描述对象之间的动态交互关系时，协作图与顺序图存在哪些区别。

【问题 3】(9分)

采用面向对象方法开发软件，通常需要建立对象模型、动态模型和功能模型，请分别介绍这3种模型，并详细说明它们之间的关联关系，针对上述模型，说明哪些模型可用于软件的需求分析?

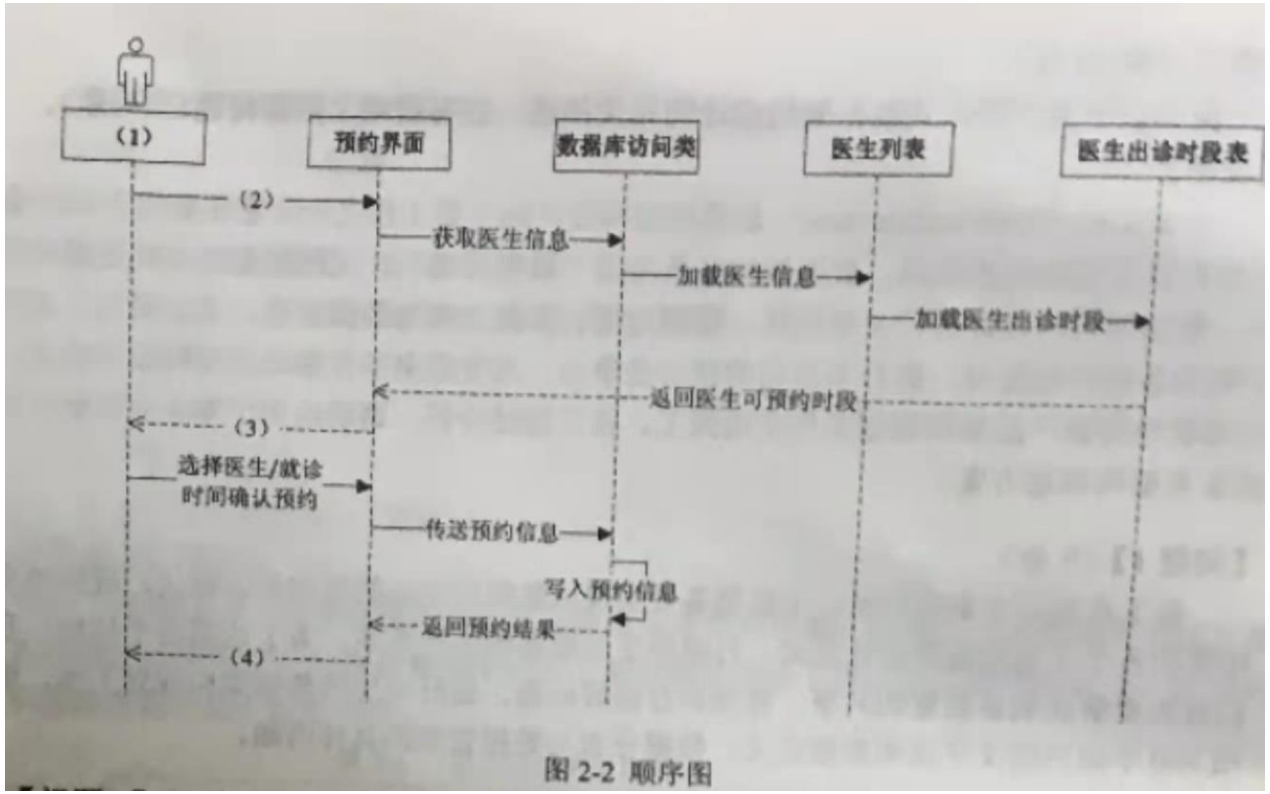


图 2-2 顺序图

案例真题

答案：
【问题1】
（1）系统管理员（2）患者
（3）a　（4）c（5）f（6）h（7）i（8）j
（9）b（10）d（11）e（12）g
（4）-（8）答案可互换。（9）-（12）答案可互换。
【问题2】
（1）预约人员（2）发起预约挂号请求（3）显示医生出诊时段（4）显示是否预约成功
顺序图强调交互的消息时间顺序。
协作图强调接受和发送消息的对象的结构组织，强调通信的方式。
【问题3】
对象模型描述系统中对象的静态结构、对象之间的关系、属性和操作，主要用对象图来实现。
动态模型描述与时间和操作顺序有关的系统特征，例如，激发事件、事件序列、确定事件先后关系的状态等，主要用状态图来实现。
功能模型描述一个计算如何从输入值得到输出值，它不考虑计算的次序，主要用DFD来实现。
功能模型指发生了什么，动态模型确定什么时候发生，而对象模型确定发生的客体。
对象设计建立基于分析模型的设计模型并考虑实现细节，可用于软件的需求分析。

谢谢！