# 软件工程概论8

1. 八、面向对象设计
   1. 1.面向对象设计准则
      1. (1) 模块化
         1. 传统的面向过程方法中的模块通常是函数、过程及子程序等，而面向对象方法中的模块则是类、对象、接口、构件等。
         2. 在面向过程的方法中，数据及在数据上的处理是分离的；而在面向对象方法中，数据及其上的处理是封装在一起的，具有更好的独立性，也能够更好地支持复用。
      2. (2) 抽象
         1. 面向对象方法不仅支持过程抽象，而且支持数据抽象。类实际上就是一种抽象数据类型。可以将类的抽象分为规格说明抽象及参数化抽象。
         2. 类对外开放的公共接口构成了类的规格说明，即协议。这种接口规定了外部可以使用的服务，使用者无需知道这些服务的具体实现算法。通常将这类抽象称为规格说明抽象。
         3. 参数化抽象是指当描述类的规格说明时并不具体指定所要操作的数据类型，而是将数据类型作为参数。
      3. (3) 信息隐藏
         1. 在面向对象方法中，信息隐藏通过对象的封装性实现。对于类的用户来说，属性的表示方法和操作的实现算法都应该是隐藏的。
      4. (4) 弱耦合
         1. 耦合是指一个软件结构内不同模块之间互连的紧密程度。在面向对象方法中，对象是最基本的模块，因此，耦合主要指不同对象之间相互关联的紧密程度。
      5. (5) 强内聚
         1. 内聚衡量一个模块内各个元素彼此结合的紧密程度。在面向对象设计中存在以下3种内聚：
         2. (1) 服务内聚：一个服务应该完成一个且仅完成一个功能。
         3. (2) 类内聚：设计类的原则是，一个类应该只有一个用途，它的属性和服务应该是高内聚的。类的属性和服务应该全都是完成该类对象的任务所必需的，其中不包含无用的属性或服务。如果某个类有多个用途，通常应该把它分解成多个专用的类。
         4. (3) 一般—特殊内聚：设计出的一般—特殊结构，应该符合多数人的概念，更准确地说，这种结构应该是对相应的领域知识的正确抽取。
      6. (6) 可重用
         1. 软件重用是提高软件开发生产率和目标系统质量的重要途径。
         2. 重用基本上从设计阶段开始。重用有两方面的含义：
         3. 一是尽量使用已有的类(包括开发环境提供的类库，及以往开发类似系统时创建的类)，
         4. 二是如果确实需要创建新类，则在设计这些新类的协议时，应该考虑将来的可重复使用性。
   2. 2.体系结构模块及依赖性
      1. 在面向对象软件中，常见的软件模块有类、接口、包和构件。
      2. 在设计阶段我们往往关注类、接口和包，在实现阶段关注构件，而在部署阶段则关注构件的部署，也就是将构件部署在哪些结点上。
      3. 1.类及其依赖性
         1. 1.类
            1. 在面向对象的程序设计中，类和接口是程序的基本组成单元。
            2. 一个典型程序需要界面类专门负责表示用户界面信息，需要数据库类负责与数据库进行交互，需要有业务逻辑类负责算法计算等。
            3. 在计算机程序中，要设计和实现的所有类都具有唯一的名字，在不同的阶段或从不同的角度可以将它们称为设计类、实现类、系统类、应用类等。
         2. 2．继承依赖性
            1. 依赖性管理中最棘手的问题是由于继承所引起的依赖性。继承是一种在父类和子类之间共享属性和行为的方式，所以运行时可以用一个子类对象代替其父类对象。

（1）多态继承

根据为请求提供服务的对象不同可以得到不同的行为，这种现象称为多态。在运行时对类进行实例化，并调用与实例化对象相应的方法，称为动态绑定、后期绑定或运行时绑定。相应地，如果方法的调用是在编译时确定的，则称为是静态绑定、前期绑定或编译时绑定。多态并不是伴随着继承而出现。如果在子类中不覆盖父类中的任何方法，就不会产生多态行为。很明显，继承会带来类和方法之间的依赖性。继承带来的依赖性有编译时继承依赖性和运行时继承依赖性。

2）无多态继承

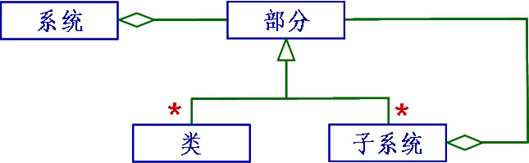
使用继承最简单的方式是子类不覆盖从父类继承来的方法，这样就不存在多态性继承问题。虽然无多态的继承有时并不是十分有用，但理解和管理起来是最容易的。

（3）扩展继承和约束继承

扩展继承是指子类继承父类的属性，并且提供额外属性来增强类定义。子类是父类的一种，如果子类覆盖了父类的方法，那么被覆盖的方法应该实现该方法的定义，并且能够在子类的语境中工作。

当一个类覆盖了继承来的方法，并对一些继承来的功能进行了限制，这时就产生了约束继承。这时，子类不再是父类的一种。有时，限制会造成继承方法的完全禁止。当方法的实现是空时，就会发生这种情况。

* + - 1. 3．交互依赖性
         1. 交互依赖性也称为方法依赖性，是通过消息连接产生的。
    1. 2.接口及其依赖性
       1. 1．接口
          1. 在UML2.0中，接口是不可直接实例化的特性集合的声明，即其对象不能直接实例化，需要通过类来实现，实现接口的类需要实现接口中声明的方法。
          2. 接口与抽象类有相似之处，抽象类是至少包含一个没有实现的方法的类，如果在一个抽象类中所有的方法都没有实现，则称其为纯抽象类，从这一点上，接口和纯抽象类似乎没有区别。但实际上，接口和抽象类还是有着本质的区别。在只支持单继承的语言中，一个类只能有一个直接父类，但是却可以实现多个接口。
       2. 2．实现依赖性
          1. 一个类可以实现多个接口，由类实现的接口集合称为该类的供给接口。在UML 2.0中，将一个类和该类实现的接口之间的依赖性称为实现依赖性
       3. 3．使用依赖性
          1. 一个接口可以为其他类或接口提供服务，同时也可能需要其他接口的服务。一个接口所需要的其他接口所提供的服务称为这个类的需求接口。需求接口详细说明一个类或接口需要的服务，从而可以为其客户提供服务。在UML2.0中，通过类（接口）和它所需接口之间的依赖关系来说明需求接口，这称为使用依赖性。
    2. 3.包及其依赖性
       1. 1. 包
          1. 包（package）又可称为层或子系统，是表示组织类的一种方式，用于划分应用程序的逻辑模型。包是高度相关的类的聚合，这些类本身是内聚的，但相对于其他聚合来说又是松散耦合的。包可以嵌套。外层包可以直接访问包括在它的嵌套包中的任何类。包还可以导入其他包
       2. 2．包依赖性
          1. 本质上，两个包之间的依赖性来自于两个包中类之间的依赖性。类之间的循环依赖性是个特别棘手的问题，好在大多数情况下可以通过重新设计避免循环依赖性。
    3. 4.构件及其依赖性
       1. 一个构件可以是一个编译的类，可以是一组编译的类，也可以是其他独立的部署单元
       2. 为了能够支持复用，软件构件应具有以下特性：
          1. （1）独立部署单元：一个构件是独立部署的，意味着它必须能与它所在的环境及其他构件完全分离。
          2. （2）作为第三方的组装单元：构件必须具备很好的内聚性，必须封装它的实现，并且只通过良好定义的接口与外部环境进行交互。
          3. （3）一个构件不能有任何（外部的）可见状态：即构件不能与自己的拷贝有所区别。
  1. 3.系统分解
     1. （1）子系统和类
        1. 在大型和复杂的软件系统情形，首先根据需求的功能模型（用例模型），将系统分解成若干个部分，每一部分又可分解为若干子系统或类，每个子系统还可以由更小的子系统或类组成，如图所示。



* + 1. （2）服务和子系统接口
       1. 服务是一组有公共目的的相关操作。而子系统则通过给其他子系统提供服务来发挥自己的能力。与类不同的是，子系统不要求其他子系统为它提供服务。
       2. 供其他子系统调用的某个子系统的操作集合就是子系统的接口。
       3. 子系统的接口包括操作名、操作参数类型及返回值。
       4. 面向对象的系统设计主要关注每个子系统提供服务的定义，即枚举所有的操作、操作参数和行为。
    2. （3）子系统之间的两种交互方式
       1. 客户-供应商关系：在这种关系中，客户子系统调用供应商子系统，后者完成某些服务工作并返回结果。使用这种交互方案，作为客户的子系统必须了解作为供应商的子系统的接口，而后者却无须了解前者的接口。
       2. 平等伙伴关系：在这种关系中，每个子系统都可能调用其他子系统，因此每个子系统都必须了解其他子系统的接口。与第一种方案相比，这种方案中，子系统间的交互更加复杂。
       3. （4）组织系统的两种方案
          1. 分层组织：这种组织方案把软件系统组织成一个层次系统，每层是一个子系统。上层在下层的基础上建立，下层为实现上层功能而提供必要的服务。每一层内所包含的对象，彼此间相互独立，而处于不同层次上的对象，彼此间往往有关联。
          2. 块状组织：这种组织方案把软件系统垂直地分解成若干个相对独立的、弱耦合的子系统，一个子系统相当于一块，每块提供一种类型的服务。
          3. 混合使用层次结构和块状结构，可以成功地由多个子系统组成一个完整的软件系统。
  1. 4 .问题域部分的设计
     1. 典型的面向对象系统一般由三层组成，即数据库层、业务逻辑层及用户界面层
     2. 可能对面向对象分析所得出的问题域模型做以下方面的补充或调整。
        1. （1）调整需求。有两种情况会导致修改通过面向对象分析所确定的系统需求：
           1. 一是用户需求或外部环境发生变化；
           2. 二是分析员对问题理解不透彻，导致分析模型不能完整、准确地反映用户的真实需求。
           3. 从类库选择已有的类，从供应商那里购买商业外购构件，从网络、组织、小组或个人那里搜集适用的遗留软构件，把它们增加到问题域部分的设计中去。
        2. (2) 复用已有的类
           1. 在被复用的已有类和问题域类之间添加泛化（一般化∕特殊化）关系，继承被复用类或构件属性和方法。
           2. 标出在问题域类中因继承被复用的已有类或构件而成为多余的属性和服务。
           3. 修改与问题域类相关的关联。
        3. (3) 把问题域类组合在一起
        4. (4) 增添泛化类以建立类间的协议
        5. (5) 调整继承的支持级别
           1. 1) 针对单继承语言的调整。
           2. 2) 针对无继承语言的调整。
        6. (6) 改进性能
           1. 1) 如果类之间经常需要传送大量消息，可合并相关的类，使得通信成为对象内的通信，而不是对象之间的通信，或者使用全局数据作用域，打破封装的原则，以减少消息传递引起的速度损失。
           2. 2) 增加某些属性到原来的类中，或增加低层的类，以保存暂时结果，避免每次都要重复计算造成速度损失。
        7. (7) 存储对象
           1. 通常的作法是，每个对象将自己传送给数据管理部分，让数据管理部分来存储对象本身。
  2. 5.人机交互部分的设计
     1. 用户界面设计步骤
        1. (1) 从系统的输入、输出及与用户的交互中获得信息，定义界面对象和行为（操作）。
        2. (2) 定义那些导致用户界面状态发生变化的事件，对事件建模。
        3. (3) 描述最终向用户展示的每一个界面的状态，。
        4. (4) 简要说明用户如何从界面提供的界面信息来解释系统状态。
     2. 界面设计目标
        1. (1) 简单性：尽量做到适度和简单，不要在页面上提供太多的东西。
        2. (2) 一致性：这一设计目标几乎适用于设计模型的每一个元素。
        3. (3) 确定性：Web应用系统的美学、界面和导航设计必须与将要构造的应用系统所处的领域保持一致。
        4. (4) 健壮性：在已经建立的确定性的基础上，Web应用系统通常会给用户明确的“承诺”。
        5. (5) 导航性：我们已经在前面提及了导航应该简单和一致，也应该以直观的和可预测的方式来设计。也就是说，用户不必搜索导航链接和帮助就知道如何使用Web应用系统。
        6. (6) 视觉吸引：在所有类型的软件中，Web应用系统毫无疑问是最具有视觉效果的、最生动的、也是最具有审美感的。
        7. (7) 兼容性：Web应用系统会应用于不同的环境(例如，不同的硬件、Internet连接类型、操作系统、浏览器)，并且必须互相兼容。
     3. 界面设计工作流
        1. (1) 回顾那些在分析模型中的信息，并根据需要进行优化。
        2. (2) 开发Web应用系统界面布局的草图。
        3. (3) 将用户目标映射到特定的界面行为。
        4. (4) 定义与每个行为相关的一组用户任务。
        5. (5) 为每个界面行为设计情节串联图板屏像。
        6. (6) 利用从美学设计中的输入来优化界面布局和情节串联图板。
        7. (7) 明确实现界面功能的界面对象。
        8. (8) 开发用户与界面交互的过程表示。
        9. (9) 开发界面的行为表示法。
        10. (10) 描述每种状态的界面布局。
        11. (11) 优化和评审界面设计模型。
  3. 6.任务管理部分的设计
     1. (1) 识别事件驱动任务
     2. (2) 识别时钟驱动任务
        1. 当系统运行时，这类任务的工作过程如下：
        2. 任务设置了唤醒时间后进入睡眠状态；
        3. 等待来自系统的一个时钟中断，一旦接收到这种中断，任务就被唤醒，并做它的工作，通知有关的对象，然后该任务又回到睡眠状态。
     3. (3) 识别优先任务
        1. 根据处理的优先级别来安排各个任务。优先任务可以满足高优先级或低优先级的处理需求。
        2. 高优先级：某些服务具有很高的优先级，为了在严格限定的时间内完成这种服务，可能需要把这类服务分离成独立的任务。
        3. 低优先级：与高优先级相反，有些服务是低优先级的，属于低优先级处理（通常称为后台处理）。设计时可能用额外的任务把这样的处理分离出来。、
     4. (4) 识别关键任务
        1. 关键任务是有关系统成功或失败的关键处理，这类处理通常都有严格的可靠性要求。
        2. 在设计过程中可能用额外的任务把这样的关键处理分离出来，以满足高可靠性处理的要求。
        3. 对高可靠性处理应该精心设计和编码，并且应该严格测试。
     5. (5) 识别协调任务
     6. (6) 审查每个任务
        1. 要使任务数保持到最少。
        2. 对每个任务要进行审查，确保它能满足一个或多个选择任务的工程标准——事件驱动、时钟驱动、优先任务∕关键任务或协调者。
     7. (7) 定义每个任务
        1. 1) 它是什么任务。首先要为任务命名，并对任务做简要描述。为面向对象设计部分的每个服务增加一个新的约束——任务名。如果一个服务被分裂，交叉在多个任务中，则要修改服务名及其描述，使每个服务能映射到一个任务。
        2. 2) 如何协调任务。定义每个任务如何协调工作。指出它是事件驱动的，还是时钟驱动的；对于事件驱动的任务，描述触发该任务的事件；对时钟驱动的任务，描述在触发之前所经过的时间间隔，同时指出它是一次性的，还是重复的事件间隔。
        3. 3) 如何通信。定义每个任务如何通信，任务从哪里取数据及往哪里送数据。
  4. 7.数据管理部分的设计
     1. 在映射时可以按下面的规则进行：
        1. (1) 一个普通的类可以映射为一个表或多个表，当分解为多个表时，可以采用横切和竖切的方法。
           1. 竖切常用于实例较少而属性很多的对象，一般是现实中的事物，将不同分类的属性映射成不同的表。通常将经常使用的属性放在主表中，而将其他一些次要的属性放到其他表中。
           2. 横切常常用于记录与时间相关的对象，如成绩记录、运行记录等。由于一段时间后，这些对象很少被查看，所以往往在主表中只记录最近的对象，而将以前的记录转到对应的历史表中。
        2. (2) 关联关系的映射
           1. 一对一关联的映射：对于一对一关联，可以在两个表中都引入外键，这样两个表之间可以进行双向导航。也可以根据具体情况，将类组合成一张单独的表。
           2. 一对多关联的映射：可以将关联中的“一”端毫无变化地映射到一张表，将关联中表示“多”的端上的类映射到带有外键的另一张表，使外键满足关系引用的完整性。
           3. 多对多关联的映射：由于记录的一个外键最多只能引用另一条记录的一个主键值，因此关系数据库模型不能在表之间直接维护一个多对多联系。为了表示多对多关联，关系模型必须引入一个关联表，将两个类之间的多对多关联转换成表上的两个一对多关联。
        3. (3) 继承关系的映射：通常使用以下两种方法来映射继承关系。
           1. 将基类映射到一张表，每个子类映射到一张表。在基类对应的表中定义主键，而在子类定义的表中定义外键。
           2. 将每个子类映射到一张表，没有基类表。在每个子类的表中包括基类的所有属性。这种方法适用于子类的个数不多，基类属性比较少的情况。
  5. 8.对象设计
     1. 对象设计过程包括使用模式设计对象、接口规格说明、对象模型重构、对象模型优化4组活动。
        1. (1) 使用模式设计对象：设计者可以选择合适的设计模式，复用已有的解决方案，以提高系统的灵活性，并确保在系统开发过程中，特定类不会因要求的变化而被修改。
        2. (2) 接口规格说明：在系统设计中所标识的子系统功能，都需要在类接口中详细说明，包括操作、参数、类型规格说明和异常情况等。
        3. (3) 对象模型重构：重构的目的是改进对象设计模型，提高该模型的可读性和扩展性。如将两个相似的类归并为一个类；将没有明显活动特征的类转为属性；将复杂的类分解为简单的类；重新组合类和操作来增进封装性和继承性等。
        4. (4) 对象模型优化：优化活动是为了改进对象设计模型，以实现系统模型中的性能要求。包括选择更好的算法、提高系统执行的速度、更好地使用存储系统、减少连接中的重数来提高查询的速度、为了增加效率而增加额外的连接、改变执行的顺序等。
     2. 设计模式包括4个要素：
        1. (1) 名字：用来将一设计模式与其他设计模式区分开。
        2. (2) 问题描述：用来描述该设计模式适用于何种情况。通常设计模式所解决的问题是对可更改性、可扩展性设计目标、以及非功能性需求的实现。
        3. (3) 解决方案：描述解决该问题所需要的、结合在一起的类和接口的集合。
        4. (4) 结果：描述将要解决设计目标的协议和可供选择的办法。
     3. 几种常用的优化方法包括：
        1. (1) 增加冗余关联以提高访问效率
        2. (2) 调整查询次序
        3. (3) 保留派生属性