面向对象技术

内容

- 面向对象方法论
- 面向对象基本概念
- 面向对象分析、设计与编程

内容

- 面向对象方法论
- 面向对象基本概念
- 面向对象分析、设计与编程

面向对象

面向对象是一种认识世界看待事物的方式，它认为：

- 世界由对象组成；
- 每个对象拥有自己的属性和操作，通过调用其操作可改变其属性状态，某种程度上是自成一体的，自包含的，自治的；
- 一个对象可通过发送消息来调用另一对象的操作，互相作用、交互、影响、合作；
- 对象组成的世界就这样存在、运作。

面向对象

“面向对象”是从结构组织的角度去模拟客观世界的一种方法，这种方法的基本着眼点是构成客观世界的那些成分----对象。

用“面向对象”的观点去认识客观世界，用“面向对象”的方法去模拟客观世界，这就构成了“面向对象”的完整含义。

面向对象软件

面向对象应用到软件领域：

- 软件系统由对象组成，对象有自己的属性和操作；用户发出服务请求，对象接收到操作请求后调用相关操作，改变自己的数据状态，并可通过向其它对象发送请求与其它对象交互，因而影响其它对象的数据状态，最终满足服务请求。
- 系统可看作一个对象网络，根据用户请求调用相关操作，改变对象网络的状态，从而满足客户请求。

面向对象软件

- 面向对象就是基于对象概念，以对象为中心，以类和继承为构造机制，充分利用多态和动态连接提供灵活性，来认识、理解、刻划客观世界，设计、构建相应的软件系统。

面向过程与面向对象

面向过程方法和面向对象方法：

- 面向过程方法中的结构化分析（SA）

  侧重于对系统进行功能分解

- 面向对象分析（OOA）

  分析以对象为中心，侧重于从现实对象的角度出发去研究和理解问题，在分析之初并不考虑具体实现

- 以图书馆信息系统为例

  SA：借阅登记、逾期罚款登记和新书录入登记等

  OOA：图书馆、管理员、借阅者和书等

面向过程与面向对象

面向过程方法和面向对象方法适用范围：

- SA建模技术一般适用于以下情况：
  - 客户对其需求非常明确
  - 业务流程定义非常完备，不会经常改动
- OOA建模技术一般适用于以下情况：
  - 软件将用面向对象的语言来编程
  - 客户需求陈述的不清楚
  - 从以往经验看，客户会频繁要求增加新功能
  - 要开发的系统很复杂
  - ……

面向过程与面向对象

面向过程方法和面向对象方法的特点：

- 面向过程方法
  - 重点放在数据结构、算法和执行步骤上
  - 数据和操作分离
  - 过程一般难以重用
  - 缺乏表达力强的可视化建模技术
- 面向对象方法
  - 系统围绕对象来构造
  - 对象把数据和行为结合起来
  - 对象之间彼此发送消息
  - 对象一般易于重用
  - 问题的可视化模型可方便地演化为解决方案的模型

面向对象方法论

整个系统的数据和操作分解封装到对象中，可以在自治对象的基础上理解系统，而不是在基本数据和操作的基础上，所以**提高了抽象级别，有助于处理复杂性**。

对象、类、接口、继承、多态等机制有助于处理复杂性多变性，以及提高可重用性等好处。

面向对象方法论

面向对象方法解决的两个经典问题：

- 多数方法论建立分离的信息和功能模型，忽略行为模型。
- 如何从分析平滑过渡到设计的问题。

面向对象方法论

面向对象的基础在于将客观世界中的应用问题看成是由实体及其相互关系组成，那么就可以将与某一应用问题有关的实体抽象为解空间中的对象。

面向对象开发方法通过提供对象、对象间信息传递等语言机制让软件开发人员在解空间中直接模拟问题空间中的对象及其行为，从而削减了语义断层，拉近了问题空间与解空间的距离，从而简化了软件工程师为问题寻找解的工作，并为软件开发活动提供了直观自然的语言支持和方法学指导。

面向对象方法的优越性

- 简化软件开发过程
- 改善软件结构
- 支持软件复用

面向对象方法的优越性

- 简化软件开发过程

面向对象方法的概念体系简单、直观、自然，支持分析、设计、实现和测试等软件开发的主要阶段。

软件开发工程中不同角色得以使用统一的概念交流和协作。

相邻阶段间软件产品的衔接转换也更容易。

面向对象方法的优越性

- 改善软件结构

面向对象软件系统可视作一群相对独立自治的类，它们各司其职，仅在必要时类的实例对象才会通过消息请求其它对象提供帮助。

面向对象方法通过对属性和操作的封装实现信息隐藏 。外界对类的内部数据的访问只能通过该类的公开接口进行，因此对该类而言是可控的。

面向对象方法还通过多态机制适当分离类的对外接口和内部实现，使得类的使用者得以使用统一的接口触发不同的功能执行，并且保证内部实现的变更不会导致接口使用方式的修改。

面向对象方法的优越性

- 支持软件重用

面向对象方法的主要概念及原则与软件重用的要求十分契合，如对象和类、抽象、封装、继承、聚合、粒度控制、多态。

面向对象的分析设计方法支持软件重用。面向对象方法从面向对象的编程发展到面向对象的分析与设计，使这种支持能够从软件生命周期的前期阶段开始发挥作用，从而达到了较高的级别。

面向对象支持软件重用

- 支持软件重用（续）

面向对象方法的一个重要优点是可以在整个软件生命周期达到概念、原则、术语及表示法的高度一致，使OO方法不但能在各个级别支持软件重用，而且能对各个级别的重用形成统一的、高效的支持，达到良好的全局效果。

做到这一点的必要条件是从OOA就把支持软件重用作为一个重点问题来考虑。

运用OOA方法所定义的对象类具有适合作为可重用组件的许多特征，OOA结果对问题域的良好映射，使同类系统的开发者容易从问题出发，在已有的OOA结果中发现不同粒度的可重用组件。

内容

- 面向对象方法论
- 面向对象基本概念
- 面向对象分析、设计与编程

面向对象的基本概念

面向对象=对象+类+继承+消息通信

—Peter Coad, Edward Yourdon

面向对象的基本概念

面向对象的基本概念包括：

对象、类、继承、聚合、多态、消息、抽象、封装、接口……

对象

一个对象是：

- 具有明确定义的边界和唯一标识(identity)的一个实体(entity)。
- 状态和行为的一个封装体。
- 类的一个实例(instance)。

对象

- 识别一个对象必须要能区别其他对象，主要是区别一个对象的边界和标识。
- 计算机软件在运行时刻，每个对象都有自己的标识，称为OID(Object Identity)，是面向对象语言系统赋予的内部标识。

对象

- 一个对象包含了属于自己的一些特定状态描述。对象的状态可用属性来表示。
- 对象的行为用于管理维护对象的状态，可能是改变状态，也可能读取状态。一种行为可用一个操作来表示，定义一个操作需要一个名字加上一组可能的形式参量。
- 一个对象的属性和操作统称为该对象的特征(feature)。

对象

- 一个对象是其所属类的一个实例。
- 计算机所实现的对象都来自于特定的类。一个对象是类实例化的结果。

对象

- 面向对象主要体现为人们观察分析世界的一种思维方式，将客观世界中的各个事物都看做是对象，承认客观对象具有自己的规律。
- 首先识别对象的标识和边界，以区分各个对象。其次识别对象的各种状态和行为，用属性来描述状态，用操作来描述行为。最后，为对象确定它所属的类。

类

一个类是：

- 一组对象的抽象描述；
- 这组对象具有相同的特征、约束和语义规范。

类

- 一个类相对于其对象是抽象的，而对象是具体的实例。

一个类必须先描述一个名字，这个名字应来自客观世界中的某种存在形式。类中并不描述单个对象的性质的值，而应描述该类所有对象共有哪些属性和操作。类不仅要抽象概括已出现的对象，而且要描述将来出现的对象。

类

- 一个类相对于其对象而言是静态的，而对象是动态的，仅在运行时刻存在。

在运行时刻，类不改变，对象是由类经实例化创建出来的，然后调用其操作而改变其状态、执行特定计算并返回结果，最后被撤销。这被称为对象的生命周期。

继承

- 继承性反映自然的分类结构。通常的说法是特殊类“是一种(is-a-kind-of)”一般类。
- 它表示类之间的内在联系以及对属性和操作的共享，即子类可以沿用父类的某些特征。子类也可以具有自己独有的属性和操作。

继承

- 继承增加了软件重用的机会。

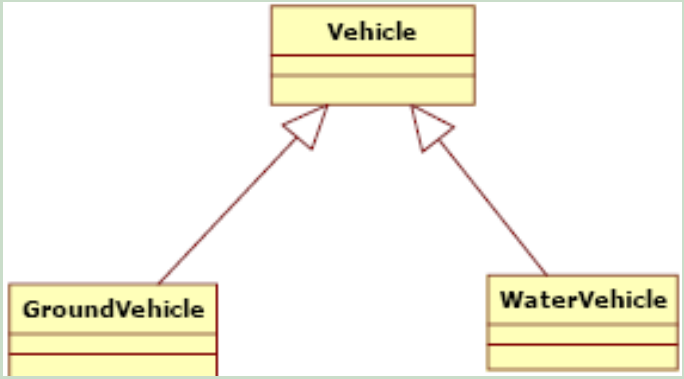
继承是类与类之间的一种关系，它使开发人员在已有类的基础上可定义和实现新的类。继承是利用可重用软件构造系统的有效机制。继承能有效地支持类的重复使用，当需要在系统中增加新特征时，所需的新代码最少。

继承

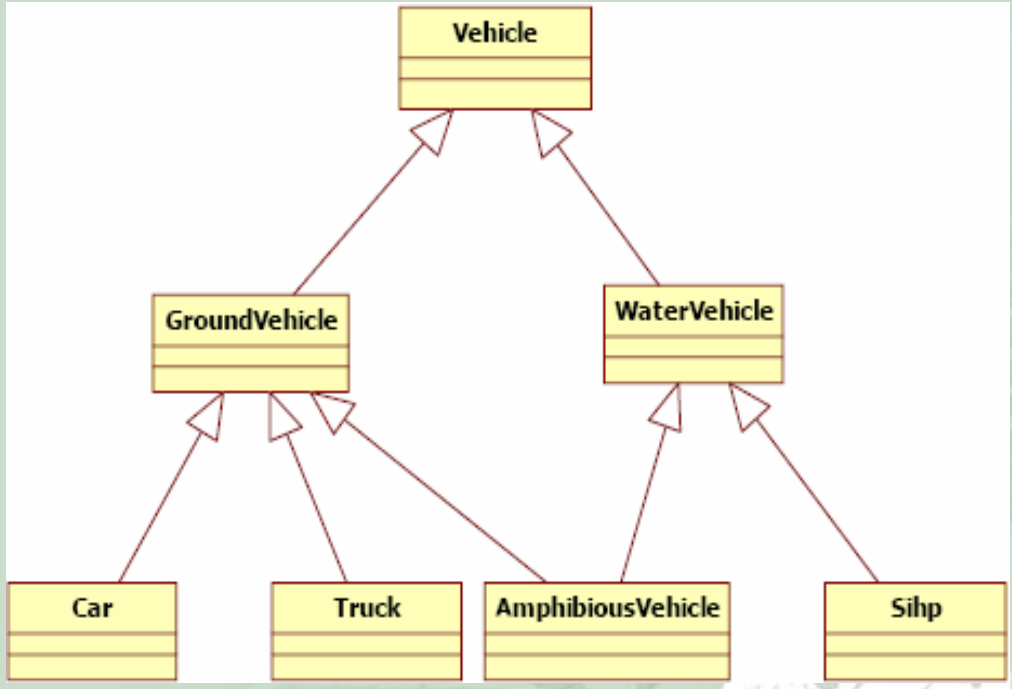
- 继承表示较一般的类与较特殊的类之间的关系。在此关系中，一般性的类(general class)通常被称为“超类”、“基类”、“父类”等，而特殊类(specifc class)被称为“子类”、“衍类”、“派生类”等。
- UML类图中用一个三角箭头和实线从子类指向超类。UML中将这种关系称为“泛化”或“一般化”。泛化关系可嵌套形成层次结构。

继承

- 单继承


Vehicle
GroundVehicle
WaterVehicle

多继承


Vehicle
GroundVehicle
WaterVehicle
Car
Truck
AmphibiousVehicle
Sihp

继承

- 要建立一个正确的继承关系需满足“里氏替代原则”：凡是需要一个超类对象的地方，而实际提供了一个子类对象，总应该是可行的。如果不能满足此原则，那么这个继承关系就有问题了。

继承

- 在继承结构中，超类自己并不知道它有什么子类，而子类必须知其超类，所以子类依赖于超类。特殊性依赖一般性，具体依赖抽象，这是面向对象设计的基本原则之一。

继承

- 在继承结构中，当一个子类实例化创建一个对象时，该子类的所有超类也必须实例化。

继承

- 继承性允许子类中说明与其超类同名的属性。当通过对象引用来访问某个性质时，在编译时刻需要一个查找过程，从引用变量的说明类型开始，按继承层次自下向上查找。

继承

- 继承性也允许子类说明与其超类具有相同的操作。这有两种目的：
- 其一是“实现”。超类仅说明抽象的操作而未提供实现方法，子类要按相同的基调来提供具体的“实现”。
- 其二是“改写”。超类虽提供了一个实现，但子类需要根据自己的特殊情况给出不同的实现，以“改写”继承而来的行为。

消息

- 消息(Message)
- 是指向对象发出的服务请求
- 包括：提供服务的对象的标识、服务标识、输入信息和回答信息等
- 对象间的消息传递是面向对象方法的基本原则
- 消息的类型
- 同步消息：请求者需要等待响应者返回
- 异步消息：请求者不需要等待响应者返回，发出消息后可以继续自己的后续工作

消息

消息传递是对象与其外部世界相互关联的唯一途径。对象可以向其他对象发送消息以请求服务，也可以响应其他对象传来的消息，完成自身固有的某些操作，从而服务于其他对象。

抽象

- 抽象的结果反映出事物重要、本质和显著的特征。抽象的过程强调被抽象事物的重要共性，而忽略不重要的差异。
- 在面向对象方法中，抽象活动主要抽取事物的结构特征和行为特征。抽象的结果有赖于特定的领域、特定的视角。

抽象

- 抽象是人类在认识复杂现象本质的过程中最强有力的思维工具。
- 抽象对认知求解过程的最大贡献在于帮助我们获取问题和方案相通的本质。
- 根据面向对象的基本思想，对问题领域中关键事物的抽象会相对稳定地延续到求解领域。

封装

- 封装是一种自然的构造，目的是保护内部；同时也是一种人为的构造，目的是方便使用。
- 我们经常使用一个遥控器或者一个简单操作面板来操纵一台空调机。一台空调机就是被封装起来的一个单元，这种封装具有保护内部、方便操作、使用安全等好处。

封装

- 封装就是把相关概念组成一个单元，然后可通过一个名字来引用它。
- 面向对象封装就是把表示状态的各个性质和对状态的各个操作包装成对象类型，使得对对象状态的存取只能通过封装提供的接口来进行，只保留有限的对外接口与外界联系。

多态

- 多态的一般性含义是某一论域中的一个元素可以有多种具体解释。

多态

- 由继承性可知，每个子类对象都是其超类的一个对象。这意味着，子类的每个对象都属于其直接或间接的多个超类。这样一个对象的所属类型具有多态性。

多态

- 在一个对象上调用一个操作具有多态性。在继承结构中，子类可重新改写继承而来的部分操作的实现，而保持其签名不变。被改写的方法在调用时可以在更抽象的超类上进行，而真正执行的是更具体的子类所提供的实现。

多态

- 这样我们就可以在超类上抽象地描述一种行为规范，或者提供一种比较通用的实现方法，而把行为的具体过程留给子类来提供，或者子类改写继承而来的实现方法。
- 如此可兼顾统一性、规范性和灵活性、可扩展性，这正是抽象编程的核心所在，很多软件框架就是以此而建立的。
- 显然，一个软件系统中，抽象越丰富，适应变化的灵活性就越强。

接口

- 除类之外，接口(interface)也是一种类型。
- 接口是一种抽象的类型，不能直接实例化。一个接口仅说明一组抽象的行为规范，作为一种职责约定或者服务规范，而不说明行为如何实现。接口通常仅说明一组相关的操作，而不说明这些操作的实现方法。

接口

- 在C++程序中，往往把包含一组公共纯虚函数的类作为接口。Java语言有专门的接口类型，说明一组公共的抽象方法和静态变量。UML提供了接口类型。

接口

- 一个接口可由多个类提供实现，而一个类可实现多个接口。这种实现接口的类被称为“实现类”。实现类应按照操作的签名和语义来提供具体的实现方法。实现类的对象都属于其类所实现的接口，所以这些对象也被称为该接口的实例。

接口

- 接口的好处体现在接口的使用方，称为客户程序。客户程序如果知道一个对象是某个接口的实例，就可以调用该接口上的操作作用于该对象，而不需要知道该对象属于哪个实现类，也不需要知道该操作的具体实现。这样改变实现类或者实现类的内部改变实现方法，都不会影响客户程序的设计。

接口

- 接口主要的用途就是表示各种服务规范，隔离客户程序与提供服务的实现类，给客户程序仅提供少量的公共操作，使客户程序的设计更方便，不必为细节所累，而且接口的实现也方便更改，不影响客户程序。
- 如果不用接口的话，客户程序就要直接依赖于提供服务的实现类，当实现类更改时，客户程序就不得不更改。
- 所以接口的好处主要是降低耦合。松耦合是良好设计的一个基本要求。

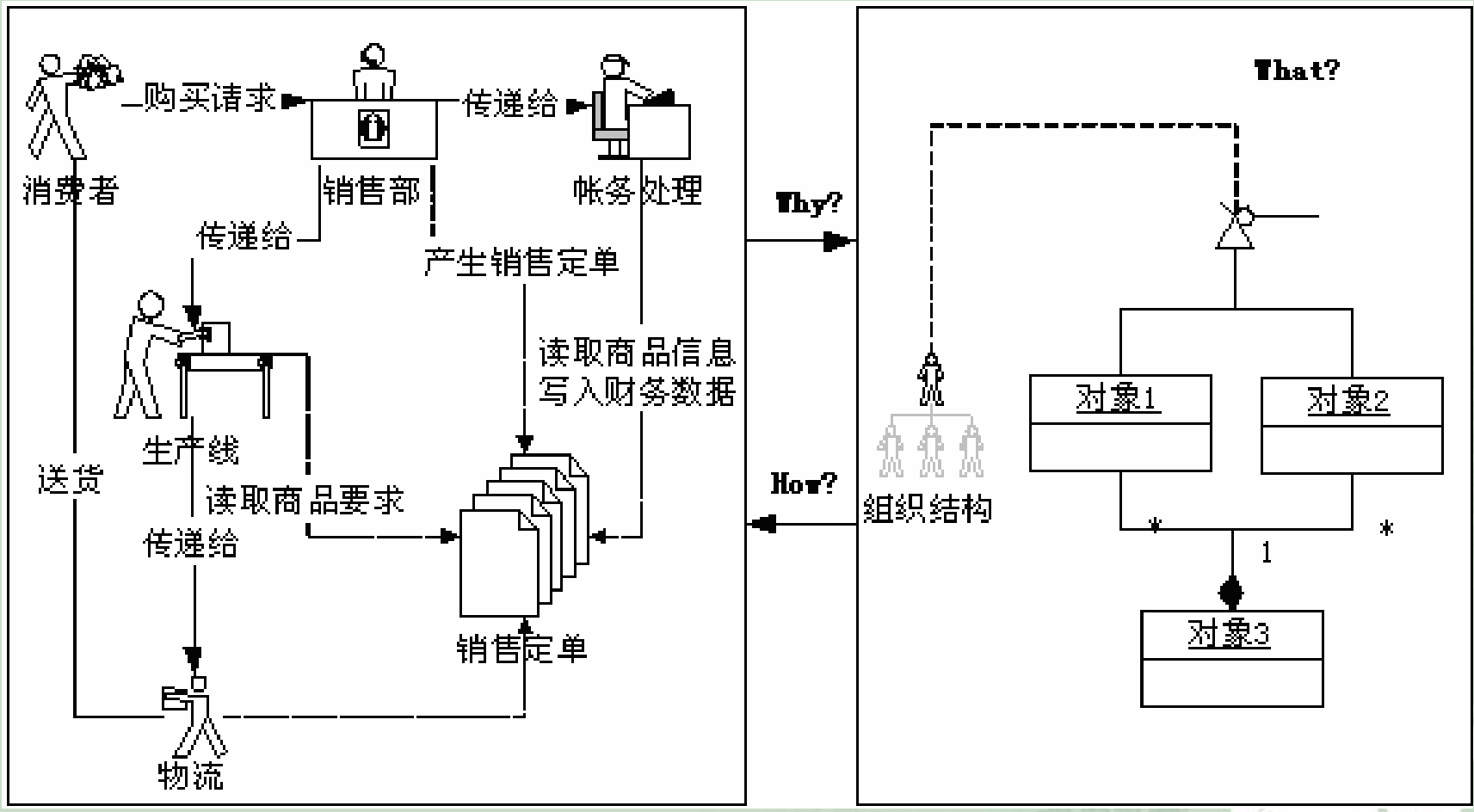
接口

- 在复杂系统设计中接口扮演着重要角色。
- 接口如果改变，不仅会影响实现类的设计，也会导致客户程序的改变。
- 功能明确定义而且稳定的接口是良好设计的关键，使实现类与客户程序之间能相互协调，减少冲突，简化设计。

内容

- 面向对象方法论
- 面向对象基本概念
- 面向对象分析、设计与编程

面向对象的困难


购买请求
传递给
消费者
销售部
帐务处理
传递给
产生销售定单
读取商品信息
写入财务数据
生产线
送货
读取商品要求
传递给
销售定单
物流
Why?
How?
What?
组织结构
对象1
对象2
对象3
*
*
1

面向对象的困难

- 计算机中实现的对象是对客观对象的一种模拟。
- 我们往往需要从分析客观对象入手，建立业务模型或分析模型，再建立设计模型，逐步抽象到计算机能识别的类。
- 在计算机运行时刻，这些类进行实例化创建计算机对象，人与对象进行交互，对象之间进行交互，最终完成计算并得到结果，这个结果对于客观对象具有模拟作用。

面向对象的困难

- 这样一个过程应建立在尊重客观事实，正确反映客观规律的基础上。
- 如果计算机中的对象能够正确反映客观对象的规律，就能较好地满足用户的需求，所实现的软件就具有较长的生存周期。

面向对象的困难

- 但是，人类的认知能力、当前软件实现能力都是有限的。我们所识别的对象可能与客观对象之间有偏差，也可能不完备。我们所设计实现的类也可能出现错误，计算机对象所提供的状态和行为可能与用户的要求有差距。
- 这意味着从认识客观对象到建立类是一个循环往复的过程，而不是一蹴而就。

面向对象分析

- 面向对象分析
- 研究问题域和客户需求，发现问题域中与系统职责有关的对象及其特征和相互关系
- 建立一个能直接映射到问题领域、符合客户需求的OOA模型
- 面向问题

面向对象分析

面向对象分析常用方法：

- 名词/动词分析
- 分类表
- 领域分析
- CRC分析
- ……

面向对象设计

- 面向对象设计
- 以OOA模型为基础，按照实现的要求进行设计决策
- 建立一个能满足客户需求且可完全实现的OOD模型
- 面向方案

面向对象设计

面向对象设计最困难的部分是将系统分解成对象集合。因为要考虑许多因素：封装、粒度、依赖关系、灵活性、性能、演化、重用等，它们都影响着系统的分解，并且这些因素通常还是相互冲突的。

面向对象设计

面向对象设计的基本原则：

- 单一职责原则：对一个类而言应该仅有一个引起它变化的原因。OOD的实质就是合理地进行类的职责分配。
- 开闭原则：软件实体应该可以扩展但不可以修改。
- Liskov替换原则：子类型必须能够替换其父类型。
- 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。
- 接口分离原则：多个专用接口优于一个单一通用接口。不要强迫客户依赖于他们不使用的方法。
- 良性依赖原则：不会在实际中造成危害的依赖关系都是良性依赖。

面向对象设计

GRASP（通用职责分配软件模式）：

- 信息专家
- 创建者
- 低耦合
- 高内聚
- 控制器
- 多态
- 中介
- 纯虚构
- 受保护变化

面向对象设计

GOF设计模式：

创建型模式--抽象工厂、生成器 、工厂方法、原型、单件；

结构型模式--适配器、桥接、组成、装饰、外观、享元、代理；

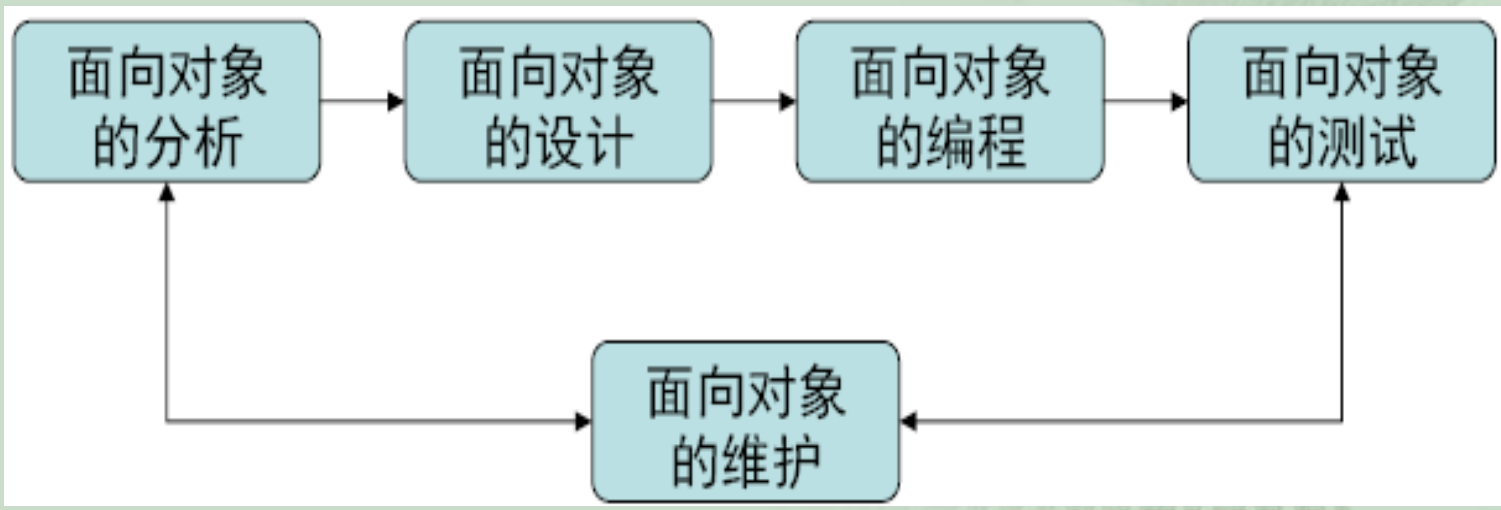
行为模式--职责链、命令、解释器、迭代器、中介者、备忘录、观察者、状态、策略、模板方法、访问者。

面向对象编程语言

- Simula,1967
- Smalltalk,20世纪60年代
- C++
- Java
- C#
- Ada

面向对象编程语言

- 以下观点已被普遍接受
- 编程并不是全部，更不是根源
- 需求分析与设计显得相当重要
- 软件开发的重点不应只对准编程阶段



面向对象编程语言

- OOP层面关注的是类、成员变量、成员函数、代码重用、名字空间……
- OOA、OOD层面关注的是类、职责、变化点隔离、交互机制、状态、角色、耦合度、内聚度、可重用性、可扩展性、可维护性……

面向对象建模

模型能反映客观世界的基本规律，也能描述将要设计实现的软件，所以建立模型对于分析和设计具有重要意义。

面向对象与UML

UML在系统建模中的作用：

- 在每个阶段都可以用到
- 建立起从概念模型直至可执行体之间明显的对应关系
- 面向对象的几个关键特征在UML中均可表示

内容

- 面向对象方法论
- 面向对象基本概念
- 面向对象分析、设计与编程