# 重点

## 多态

动态绑定是实现多态的基础

### 几类多态

一般将多台分为通用多态和特殊多态。通用多态包括参数多态和包含多态。  
参数多态采用参数化模板，通过给出不同的类型参数，使得一个结构有多种类型。  
包含多态同样的操作可用于一个类型及其子类型。（注意是子类型，不是子类。）包含多态一般需要进行运行时的类型检查。如Pascal中的子界。  
特殊多态包括强制多态和过载多态。  
强制多态编译程序通过语义操作，把操作对象的类型强行加以变换，以符合函数或操作符的要求。程序设计语言中基本类型的大多数操作符，在发生不同类型的数据进行混合运算时，编译程序一般都会进行强制多态  
过载多态是一种特定的多态，指同一个名（操作符、函数名）在不同上下文中可代表不同的含义。

## 设计模式

适配器

装饰

观察者

状态模式

策略模式

生成器模式

### 介绍

抽象工厂模式（Abstract Factory）: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。  
构建器模式（Builder）：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。  
工厂方法模式（Factory Method）：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。工厂方法使得子类实例化的过程推迟。  
原型模式（Prototype）：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。  
单例模式（Singleton）: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。  
适配器模式（Adapter）：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。  
桥接模式（Bridge）：将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化。  
组合模式（Composite）：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。  
装饰模式（Decorator）：动态地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。  
外观模式（Facade）: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。  
享元模式（Flyweight）：提供支持大量细粒度对象共享的有效方法。  
代理模式（Proxy）：为其他对象提供一种代理以控制这个对象的访问。  
职责链模式（Chain of Responsibility）：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。  
命令模式（Command）：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。  
解释器模式（Interpreter）：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。  
迭代器模式（Iterator）：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。  
中介者模式（Mediator）：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，还可以独立地改变对象间的交互。  
备忘录模式（Memento）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。  
观察者模式（Observer）：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。  
状态模式（State）：允许一个对象在其内部状态改变时改变它的行为。  
策略模式（Strategy）：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。  
模板方法模式（Template Method）：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。  
访问者模式（Visitor）：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

## UMl

活动图

状态图

等等

线表示啥

组件图

用例图

### 各种图

（1）类图（class diagram）。类图描述一组类、接口、协作和它们之间的关系。在OO系统的建模中，最常见的图就是类图。类图给出了系统的静态设计视图，活动类的类图给出了系统的静态进程视图。  
（2）对象图（object diagram）。对象图描述一组对象及它们之间的关系。对象图描述了在类图中所建立的事物实例的静态快照。和类图一样，这些图给出系统的静态设计视图或静态进程视图，但它们是从真实案例或原型案例的角度建立的。  
（3）构件图（component diagram）。构件图描述一个封装的类和它的接口、端口，以及由内嵌的构件和连接件构成的内部结构。构件图用于表示系统的静态设计实现视图。对于由小的部件构建大的系统来说，构件图是很重要的。构件图是类图的变体。  
（4）组合结构图（composite structure diagram）。组合结构图描述结构化类（例如，构件或类）的内部结构，包括结构化类与系统其余部分的交互点。组合结构图用于画出结构化类的内部内容。  
（5）用例图（use case diagram）。用例图描述一组用例、参与者及它们之间的关系。用例图给出系统的静态用例视图。这些图在对系统的行为进行组织和建模时是非常重要的。  
（6）顺序图（sequence diagram，序列图）。顺序图是一种交互图（interaction diagram），交互图展现了一种交互，它由一组对象或参与者以及它们之间可能发送的消息构成。交互图专注于系统的动态视图。顺序图是强调消息的时间次序的交互图。  
（7）通信图（communication diagram）。通信图也是一种交互图，它强调收发消息的对象或参与者的结构组织。顺序图和通信图表达了类似的基本概念，但它们所强调的概念不同，顺序图强调的是时序，通信图强调的是对象之间的组织结构（关系）。在UML 1.X版本中，通信图称为协作图（collaboration diagram）。  
（8）定时图（timing diagram，计时图）。定时图也是一种交互图，它强调消息跨越不同对象或参与者的实际时间，而不仅仅只是关心消息的相对顺序。  
（9）状态图（state diagram）。状态图描述一个状态机，它由状态、转移、事件和活动组成。状态图给出了对象的动态视图。它对于接口、类或协作的行为建模尤为重要，而且它强调事件导致的对象行为，这非常有助于对反应式系统建模。  
（10）活动图（activity diagram）。活动图将进程或其他计算结构展示为计算内部一步步的控制流和数据流。活动图专注于系统的动态视图。它对系统的功能建模和业务流程建模特别重要，并强调对象间的控制流程。  
（11）部署图（deployment diagram）。部署图描述对运行时的处理节点及在其中生存的构件的配置。部署图给出了架构的静态部署视图，通常一个节点包含一个或多个部署图。  
（12）制品图（artifact diagram）。制品图描述计算机中一个系统的物理结构。制品包括文件、数据库和类似的物理比特集合。制品图通常与部署图一起使用。制品也给出了它们实现的类和构件。  
（13）包图（package diagram）。包图描述由模型本身分解而成的组织单元，以及它们之间的依赖关系。  
（14）交互概览图（interaction overview diagram）。交互概览图是活动图和顺序图的混合物。