### 面向对象基础

类与对象（核心）

1类是一组具有相同属性和方法的对象抽象。一个类中的每个对象都是这个类的实体。

2类的静态成员与一般成员

类的静态成员与一般成员不同，静态成员与对象的实例无关，只与类本身有关，它们一般用来实现类的封装的功能和数据。但不包括特定对象的功能和数据。

1. 普通数据成员属于类的具体对象，对象创建才会分配内存，静态成员属于类，即使没有任何对象创建，类的静态数据成员变量也会存在。
2. 外部访问静态成员只能通过类名访问。
3. 类的静态成员无法直接访问普通数据成员（可以通过类的指针间接访问），而类的任 何成员函数都可以访问该类的静态数据成员。
4. 类的静态方法只能访问该类的静态成员。

3类的分类

边界类：边界类描述系统外部环境和系统内部运作的交互，工作在参与者和系统之间，而边界对象表示的为一个交互接口。

抽象类：

实体类：存储和管理系统内部信息，可以有行为，甚至很复杂的行为，但行为必须与代表的实体有关。实体类独立于系统外部环境。

控制类：秒偶数特定用例的控制行为，也特定的用例实现密切相关，可以说他就是执行用例实现，控制类可以有效的降低边界类和实体类的耦合，使系统对于外部环境的变化具有良好的适应性。

4对象是指一组属性及这组属性上的专用操作的封装体。它由对象标识（名称）

属性（状态 数据 成员变量 另一对象） 服务（操作 行为 方法）三要素组成。对象被封装为一个整体，以接口形式对外提供服务。

多态（核心）

在面向对象技术中，对象在受到消息后予以反应，不同的对象收到同一消息可产生不同的效果，这一现象称为多态。参数多态和包含多态称为通用多态，过载多态和强制多态称为特定多态。

参数多态：同一对象或函数或过程能以一致的形式用于不同的类型。

包含多态：包含多态最常见的是子类型化，一个类型是另一类型的子类型。

过载多态：同一变量来表示不同的功能，通过上下文来确定一个类所代表的的概念。

强制多态：通过语义操作把一个变元类型加以变换，以符合函数要求，类型的变换可以在编译时隐式进行，也可以在动态运行时进行。

封装（核心）

封装性是一种信息隐蔽技术，使得系统分析员能够清晰的标明它们所提供的服务界面，而用户和应用程序员只看得见对象提供的操作功能，而看不见数据或操作代码细节。封装就是将抽象的数据与行为相结合。

重载（核心）

重载，在一个类定义中，可以编写几个同名的方法，但是只要它们的签名[参数列表](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=74271600&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)不同，Java就会将它们看做唯一的方法。简单的说，一个类中的方法与另一个方法同名，但是[参数表](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=2556542&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)不同，这种方法称之为重载方法。重载有四种形式的多态：[虚函数](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=577043&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)多态、模板多态、重载与转换。

继承与泛化（核心）

## 继承 当多个类出现部分相同的实例变量和方法时,就考虑用继承

继承的使用:将多个类中相同的实例变量的方法提出来写成一个公共的父类

泛化 表示类与类之间的继承关系，接口与接口之间的继承关系，或类对接口的实现关系。

一般化的关系是从子类指向父类的，与继承或实现的方法相反。

|  |  |  |
| --- | --- | --- |
| 基类 |  | 派生类  （派生类对象是否可访问基类方法） |
| 公有成员 私有成员 保护成员 | 公有派生  它类  对象访问 | 公有成员 私有成员 保护成员  可访问 不可访问 不可访问 |
| 公有成员 私有成员 保护成员 | 公有派生  己类内部成员访问 | 公有成员 私有成员 保护成员  可访问 不可访问 可访问 |
| 公有成员 私有成员 保护成员 | 私有派生  它类  对象访问 | 私有成员 私有成员 私有成员  不可访问 不可访问 不可访问 |
| 公有成员 私有成员 保护成员 | 私有派生  己类内部成员访问 | 私有成员 私有成员 私有成员  可访问 可访问 可访问 |
| 公有成员 私有成员 保护成员 | 保护派生  它类  对象访问 | 保护成员 私有成员 保护成员  不可访问 不可访问 不可访问 |
| 公有成员 私有成员 保护成员 | 保护派生  己类内部成员访问 | 保护成员 私有成员 保护成员  可访问 不可访问 可访问 |

---------------------------------------------------------------------------------------------------------------------------------

例如已知有两个类O P，类O中定义了两个方法，一个私有方法F1，一个公有方法F2；

类P定义一个公有方法；类P为类O的派生类。

继承方式 class:private O{.......} 则类P的对象不可以访问F1，F2。

class:public O{.......} 则类P的对象不可以访问F1，可以访问F2。

class:protected O{.......} 则类P的对象不可以访问F1，可以访问F2。

构造函数与析构函数（核心）

|  |  |  |
| --- | --- | --- |
|  | 构造函数 | 析构函数 |
| 定义 | 构造函数与类名相同，在建立对象时自动调用。 | 析构函数是对象小时前的瞬间自动调用的函数。 |
| 特点 | 1. 构造函数的函数名必须与定义它的类同名。 2. 构造函数没有返回值，不可以在前面加void 3. 构造函数被声明定义为公有函数。 4. 构造函数在建立对象时由系统自动调用。 | 1. 析构函数没有任何参数，不能被重载，但可以是虚函数，一个类一个析构函数。 2. 析构函数没有返回值。 3. 析构函数名与类名相同，但在类名前加逻辑非运算符~ 以示区别构造函数。 4. 析构函数用户自己定义，对象消失系统自动调用，若无定义析构函数，自动生成不做事情的默认析构函数。 |

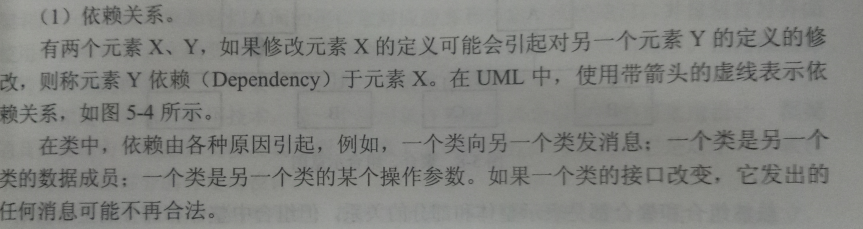
模板类（核心）

模板是根据参数类型[生成函数](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=406157&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)和类的机制（有时称为“参数决定类型”），通过使用模板，可以只设计一个类来处理多种类型的数据，而不必为每一种类型分别创建类。

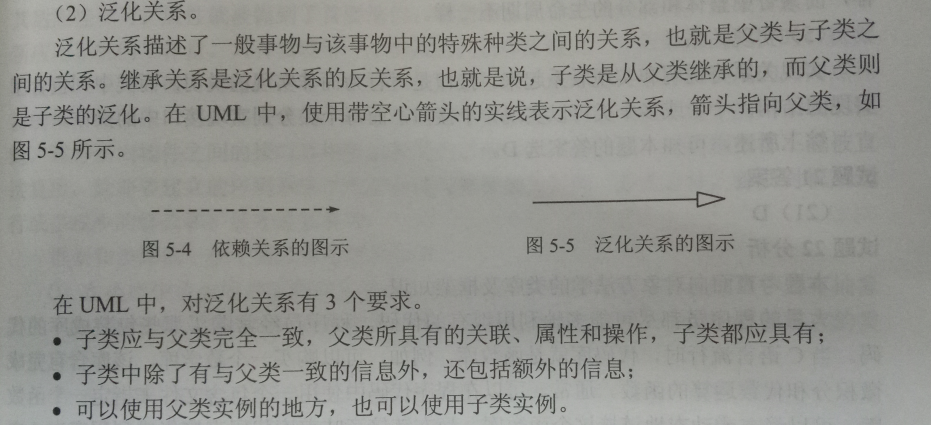
类与类的关系（核心）



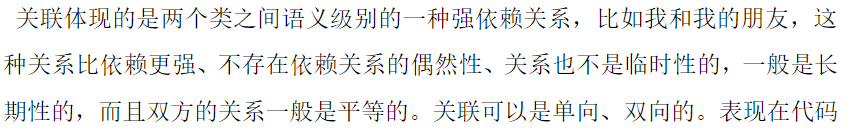
1. 依赖



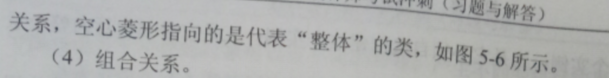
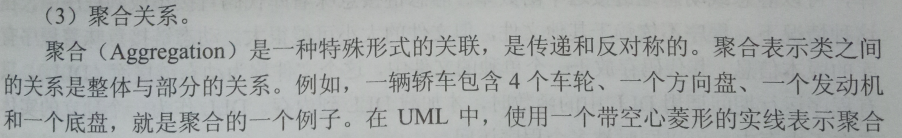
1. 泛化



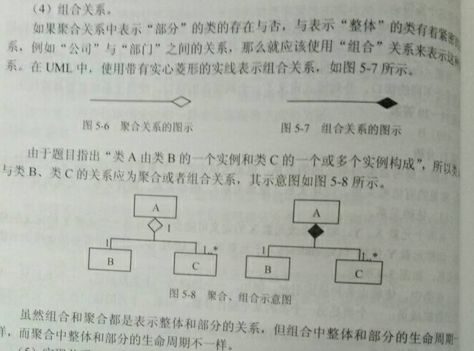
1. 关联



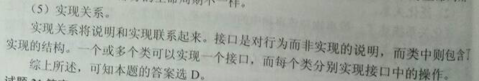
1. 聚合



1. 组合



1. 实现



用例与用例的关系（核心）

1. 包含：当从两个和两个以上的用例提取公共行为，使用包含关系，提取的公共用例称为抽象用例，而原始用例称为基本用例或基础用例。抽象用例必须是基础用例的一部分。
2. 扩展：如果一个用例明显的混合两种或多种情景，即可能发生多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例。抽象用例不一定是基础用例的一部分。
3. 泛化：当多个用例拥有一种类似的结构和行为的时候，可以将他们的共性抽象为父用例，其它关系作为泛化关系的子用例。

### 面向对象分析与设计基础

面向对象方法

(1).Booch方法

Booch最先描述了面向对象的软件开发方法的基础问题，指出面向对象开发是一种根本不同于传统的功能分解的设计方法。面向对象的软件分解更接近人对客观事务的理解，而功能分解只通过问题空间的转换来获得。

(2).OMT方法

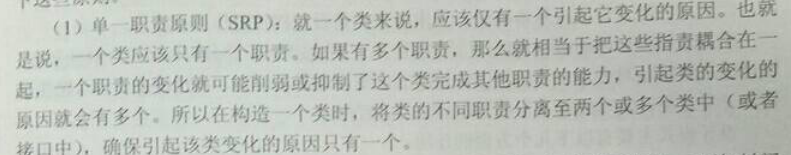
OMT方法是1991年由James Rumbaugh等5人提出来的，其经典著作为"面向对象的建模与设计"。

(3).Coad方法

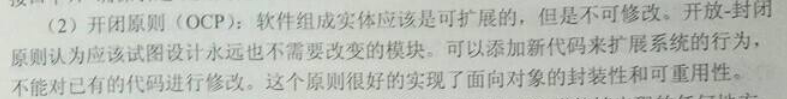
Coad方法是1989年Coad和Yourdon提出的面向对象开发方法。该方法的主要优点是通过多年来大系统开发的经验与面向对象概念的有机结合，在对象、结构、属性和操作的认定方面，提出了一套系统的原则。该方法完成了从需求角度进一步进行类和类层次结构的认定。尽管Coad方法没有引入类和类层次结构的术语，但事实上已经在分类结构、属性、操作、消息关联等概念中体现了类和类层次结构的特征。

面向对象七大原则

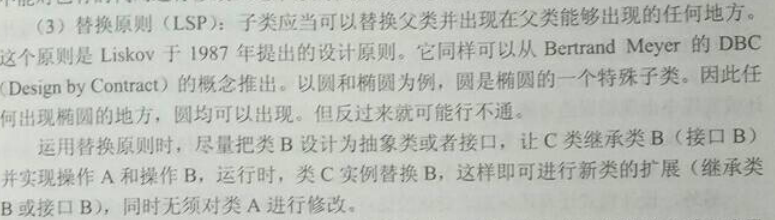
1. 单一职责原则



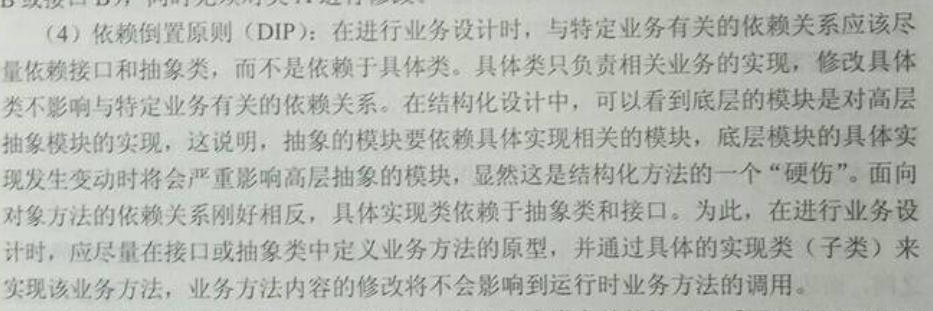
1. 开闭原则



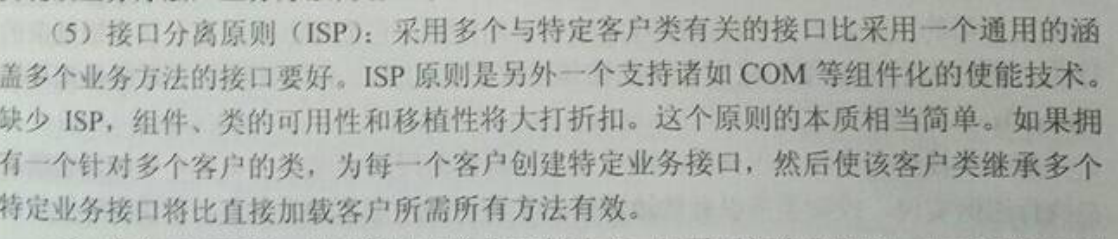
1. 李氏替换原则



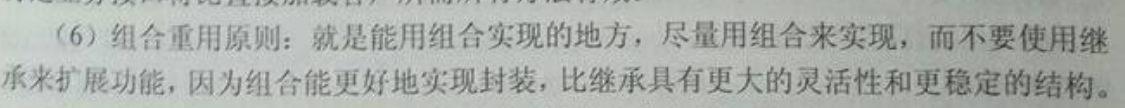
1. 依赖倒置原则



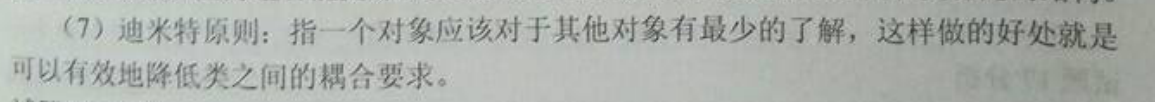
1. 接口隔离原则



1. 组合重用原则



1. 迪米特原则



面向对象分析与设计

面向对象分析：建模系统功能；发现并确定业务对象；组织对象并确定其关系。

面向对象设计：设计确定对象间的关系 设计确定对象间的通信方式（消息模式）

面型对象的优势

1、易维护

采用面向对象思想设计的结构，可读性高，由于继承的存在，即使改变需求，那么维护也只是在局部模块，所以维护起来是非常方便和较低成本的。

2、质量高

在设计时，可重用现有的，在以前的项目的领域中已被测试过的类使系统满足业务需求并具有较高的质量。

3、效率高

在软件开发时，根据设计的需要对现实世界的事物进行抽象，产生类。使用这样的方法解决问题，接近于日常生活和自然的思考方式，势必提高软件开发的效率和质量。

4、易扩展

由于继承、封装、多态的特性，自然设计出高内聚、低耦合的系统结构，使得系统更灵活、更容易扩展，而且成本较低。

### UML基础

UML概念

UML有四种事物 结构事物 分组事物 行为事物 注释事物

结构事物是UML模型中的名词，模型的静态部分，描述概念或物理元素。

分组事物是UML模型中的组织部分，由模型分解成的盒子。

行为事物是UML模型中的动词，模型的动态部分，描述跨越时间和空间。

注释事物是UML模型中解释部分，这些注释事物用来描述 说明和标注模型的任何元素。

---------------------------------------------------------------------------------------------------------------------------------

UML系统五大视图

逻辑视图：也称为设计视图，设计模型在架构方面具有重要意义的部分，类 子系统 包 用例实现的子集。

进程视图：是可执行线程和进程为活动类的建模，是逻辑视图依次执行实例，描述并发与同步结构。

实现视图：实现视图对组成基于系统的物理代码的文件和构件进行建模。

部署视图：部署视图把构件部署到一组物理结点上，表示软件到硬件的映射和分布结构。

用例试图：是最基本的需求分析模型。

---------------------------------------------------------------------------------------------------------------------------------

在UML中，把组织模型的组块称为包，包是用于把元素组织成组的通用机制，有助于组织模型中的元素。使得用户能够控制对包的访问。

接口是描述某个类或构件的一个服务操作集。接口是来为类或构件提供功能 操作 行为的。

构件是为接口提供实现的物理部分。

UML常见图

结构性视图（静态）结构领域主要是对系统中的结构成员及相互关系进行描述

行为性视图（动态）行为领域则描述系统随时间变化的行为。

---------------------------------------------------------------------------------------------------------------------------------

结构性视图

（核心）类图：类图描述一组类 接口 协作和他们之间的关系。类图给出系统的静态设计视图，活动类图给出系统的静态进程视图。

对象图：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。这些图给出系统的静态设计图或静态进程视图。但对象图是从真实案例或原型案例角度建立。

包图：包图描述由模型本身分解而成的组织单元，以及他们之间的依赖关系。

组合结构图：组合结构图描述结构化类（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。

构件图：构件图描述一个封装的类和它的接口 端口 以及内嵌的构件和连接构件的内部结构。构件图用于表示系统的静态设计实现视图。构件图是类图的变体。

部署图：部署图描述对运行时的处理结点及在其中生存的构件配置，部署图给出了架构的静态部署视图，通常一个结点包括一个或多个部署图。

制品图：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，制品图经常和部署图一起使用，制品也给出它们实现的类和构件。

行为性视图

（核心）用例图：通常描述一组用例 参与者与它们之间的关系。用例图给出系统的静态用例视。

这些图对系统的行为进行组织和建模时非常重要。

（核心）顺序图：顺序图一种交互图，交互图展现一种交互，由一组对象或参与者以及他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。顺序图强调消息的时间次序的交互图。

通信图：通信图是一种交互图，强调收发消息的对象或参与者的结构组织。该图反映了对象之间的消息交互，与顺序图相似但不同，协作图不但描述了对象之间的交互还描述交互对象之间的链接关系。，即通信图同时反映了系统的动态和静态特征。通信图（是协作图）

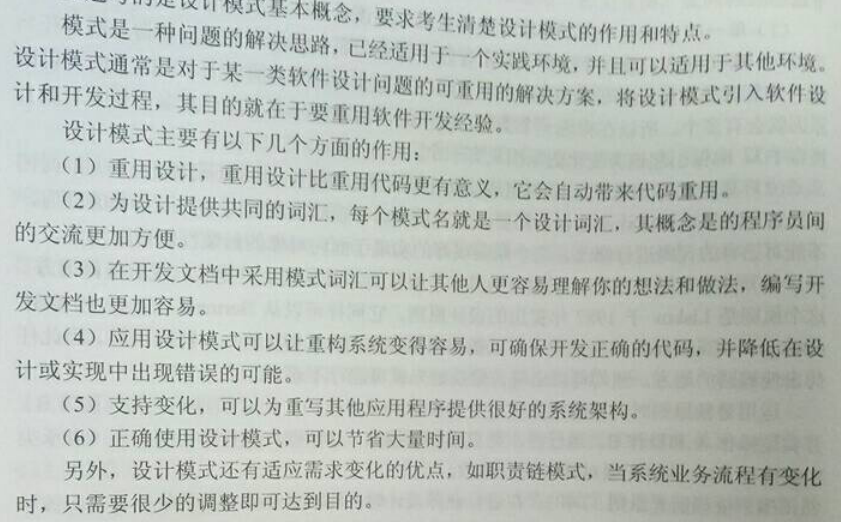
定时图：定时图也是一种交互图，强调消息跨越不同对象或参与者的实际时间，而不是仅仅只关心消息的相对时间。

状态图：状态图描述一个状态机，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，它对接口 类或协作的行为建模很重要。它强调事件导致的对象行为，有助于反应式系统建模。

活动图：活动图将进程或其他计算机结构展示为计算机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，对系统的功能建模和业务流程建模很重要，强调对象间控制流程。

交互概览图：交互概览图是活动图和顺序图的混合物。

设计模式（核心）



设计模式分为 创建型 结构型 行为性。

创建型：工厂方法 抽象工厂 单例 构建 原型

结构型：适配器 合成 装饰 代理 享元 门面 桥接

行为型: 模板 迭代器 责任链 命令 备忘录 状态 访问者 解释器 调停者 观察者。

（1）适配器（Adapter）适配器设计模式的意图是将一个类的接口转换为客户所希望的接口。使得原本不兼容的类一起工作。适用于想使用一个已存在但接口不合适的类。

（2）原型设计模式（Prototype）的意图啊用原型实例指定创建对象的种类，并且通过复制这些原型创建新对象。

（3）迭代器设计模式（Iterator）的意图是提供一种顺序访问一个聚合对象中的各个元素，而不暴露该对象内部表示。

（4）观察者设计模式（Observer）定义对象间一对多的依赖关系，当一对象发生改变时，所有依赖它的对象都将发生改变。并自动更新。

（5）桥接模式意图是将抽象部分与实现部分分离，独立的变化。适用于

（1）避免抽象方法和实现方法捆绑一起。

（2）类的抽象以及它的实现都应该可以通过生成子类的方法扩充。

（3）对一个抽象的实现方法的修改应对客户不影响，对应客户的代码不变。

（4）想在多个对象间共享实现，要求客户不知道

（6）单例模式确保一个类只有一个实例，自行实例化，并向整个系统提供这个实例。

（7）合成设计模式组合多个对象形成树型结构，以表示整体----部分的结构层次。

合成模式对单个对象和多个对象具有一致性。

（8）门面模式（外观模式）提供一个接口访问子系统多个不同的接口。

（9）装饰模式的意图是动态的给一个对象增加职责，不改变原类文件和使用继承的情况下，动态扩展一个对象的功能。适用于

（1）在不影响其他对象的前提下，以动态透明的方式给单个对象添加职责。

（10）代理模式的意图是为其它对象提供一个代理或地方以控制对这个对象的访问。

（11）享元模式的意图是运用共享技术有效的支持大量细粒度的对象。

（12）策略模式的意图是定义一系列的算法，把它们封装起来，并且使他们可以相互替换。本模式使得算法独立于使用它的客户而变化。

---------------------------------------------------------------------------------------------------------------------------------

Abstract Factory（抽象工厂模式）：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

Adapter（适配器模式）：将一个类的接口转换成客户希望的另外一个接口。A d a p t e r模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

Bridge（桥接模式）：将抽象部分与它的实现部分分离，使它们都可以独立地变化。

Builder（建造者模式）：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

Chain of [Responsibility](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=1654456&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)（职责链模式）：为解除请求的发送者和接收者之间耦合，而使多个对象都有机会处理这个请求。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它。

Command（命令模式）：将一个请求封装为一个对象，从而使你可用不同的请求对客户进行[参数化](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=38199498&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)；对请求排队或记录请求日志，以及支持可取消的操作。

Composite(组合模式)：将对象组合成[树形结构](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7714336&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)以表示“部分-整体”的层次结构。它使得客户对单个对象和复合对象的使用具有一致性。

Decorator(装饰模式)：动态地给一个对象添加一些额外的职责。就扩展功能而言， 它比生成子类方式更为灵活。

Facade（外观模式）：为子系统中的一组接口提供一个一致的界面， F a c a d e模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

设计模式Factory Method(工厂模式)：定义一个用于创建对象的接口，让子类决定将哪一个类实例化。Factory Method使一个类的实例化延迟到其子类。

[Flyweight](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10549158&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)（享元模式）：运用共享技术有效地支持大量[细粒度](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=70897420&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)的对象。

Interpreter（解析器模式）：给定一个语言, 定义它的文法的一种表示，并定义一个解释器, 该解释器使用该表示来解释语言中的句子。

Iterator（迭代器模式）：提供一种方法顺序访问一个聚合对象中各个元素, 而又不需暴露该对象的内部表示。

Mediator（中介模式）：用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

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。这样以后就可将该对象恢复到保存的状态。

Observer（观察者模式）：定义对象间的一种[一对多](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7746200&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)的依赖关系,以便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刷新。

[Prototype](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=45373&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)（原型模式）：用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象。

Proxy（代理模式）：为其他对象提供一个代理以控制对这个对象的访问。

Singleton（单例模式）：保证一个类仅有一个实例，并提供一个访问它的全局访问点。 单例模式是最简单的设计模式之一，但是对于Java的开发者来说，它却有很多缺陷。在本月的专栏中，David Geary探讨了单例模式以及在面对[多线程](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=452880&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)（multithreading)、类装载器（classloaders）和[序列化](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=54553&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)(serialization)时如何处理这些缺陷。

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。对象看起来似乎修改了它所属的类。

Strategy（策略模式）：定义一系列的算法,把它们一个个封装起来, 并且使它们可相互替换。本模式使得算法的变化可独立于使用它的客户。

Template Method（模板方法模式）：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可[重定义](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=73746426&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)该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。它使你可以在不改变各元素的类的前提[下定义](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7705050&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)作用于这些元素的新操作。