行为型模式

模板方法模式

- 定义：在一个方法中定义一个算法骨架，并将某些步骤推迟到子类中实现。模板方法模式可以让子类在不改变算法整体结构的情况下，重新定义算法中的某些步骤
- 适用场景：一次性实现一个算法的不变的部分，并将可变的行为留给子类来实现
- 适用场景：各子类中公共的行为被提取出来并集中到一个公共父类中，从而避免代码重复
- 优缺点
  - 优点：提高复用性
  - 优点：提高扩展性
  - 优点：符合开闭原则
  - 缺点：继承关系自身缺陷，如果父类添加新的抽象方法，所有子类都有改一遍
- 源码应用
  - JDK：AbstractQueuedSynchronizer
  - JDK：AbstractExecutorService
  - Servlet：HttpServlet

策略模式

- 定义：定义一族算法类，将每个算法分别封装起来，让它们可以互相替换。策略模式可以使算法的变化独立于使用它们的客户端
- 适用场景：系统有很多类，它们的区别仅在于行为不同
- 适用场景：一个系统需要动态地在几种算法中选择一种
- 优缺点
  - 优点：算法可以自由切换
  - 优点：避免使用多重条件判断
  - 优点：开闭原则
- 源码应用
  - JDK：Comparator
  - Spring Core：Resource
  - Spring Security：SecurityContextHolderStrategy

观察者模式

- 定义：在对象之间定义一个一对多的依赖，当一个对象状态改变的时候，所有依赖的对象都会自动收到通知
- 适用场景：关联行为场景，建立一套触发机制
- 优缺点
  - 优点：观察者和被观察者是抽象耦合的
  - 优点：支持广播通信
  - 缺点：观察者之间有过多的细节依赖，提高时间消耗及程序复杂度
  - 缺点：使用要得当，要避免循环调用
- 源码应用
  - Spring Boot：监听器模式
  - Guava：EventBus、@Subscribe

责任链模式

- 定义：将请求的发送和接收解耦，让多个接收对象都有机会处理这个请求。将这些接收对象串成一条链，并沿着这条链传递这个请求，直到链上的某个接收对象能够处理它为止
- 适用场景：一个请求的处理需要多个对象当中的一个或几个协作处理
- 优缺点
  - 优点：请求的发送者和接收者（请求的处理）解耦
  - 优点：对象不需要知道链的结构
  - 优点：责任链可以动态组合
  - 缺点：不能保证请求一定被接收
  - 缺点：系统性能将受到一定影响，而且在进行代码调试时不太方便，可能会造成循环调用
- 源码应用：Tomcat：Filter

状态模式

- 定义：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
- 适用场景：一个对象存在多个状态（不同状态下行为不同），且状态可相互转换
- 优缺点
  - 优点：将不同的状态隔离
  - 优点：把各种状态的转换逻辑，分布到 State 的子类中，减少相互间依赖
  - 优点：增加新的状态非常简单
  - 缺点：使用不当将导致程序结构和代码的混乱
  - 缺点：不符合开闭原则

解释器模式

- 定义：为某个语言定义它的语法表示，并定义一个解释器用来处理这个语法
- 适用场景：某个特定类型问题发生频率足够高
- 优缺点
  - 优点：语法由很多类表示，容易改变及扩展此“语言”
  - 缺点：当语法规则数目太多时，增加了系统复杂度
- 源码应用
  - JDK：Pattern
  - Spring：ExpressionParser

访问者模式

- 定义：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类的前提下定义作用于这些元素的新操作
- 适用场景：一个数据结构（List/Set/Map等）包含很多类型对象
- 优缺点
  - 优点：优秀的扩展性
  - 优点：符合单一职责原则
  - 缺点：具体元素对访问者公布细节，违反了迪米特原则
  - 缺点：具体元素变更比较困难
  - 缺点：违反了依赖倒置原则
- 源码应用
  - JDK：FileVisitor
  - Spring IOC：BeanDefinitionVisitor

迭代器模式

- 定义：提供一种方法顺序访问一个聚合对象中的各种元素，而又不暴露该对象的内部表示
- 适用场景：访问一个集合对象的内容而无需暴露它的内部表示
- 适用场景：为遍历不同的集合结构提供一个统一的接口
- 优缺点
  - 优点：分离了集合对象的遍历行为
  - 缺点：类的个数成对增加
- 源码应用：JDK：Iterator

备忘录模式

- 定义：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
- 适用场景：保存及恢复数据相关业务场景
- 适用场景：后悔的时候，即想恢复到之前的状态
- 适用场景：游戏存档、博客暂存、工作流
- 优缺点
  - 优点：为用户提供一种可恢复机制
  - 优点：存档信息的封装
  - 缺点：资源占用

中介者模式

- 定义：定义了一个单独的对象，来封装一组对象之间的交互。将这组对象之间的交互委派给与中介对象交互，来避免对象之间的直接交互
- 适用场景：系统中对象之间存在复杂的引用关系，产生的相互依赖关系结构混乱且难以理解
- 适用场景：交互的公共行为，如果需要改变行为则可以增加新的中介者类
- 优缺点
  - 优点：将一对多转化程了一对一、降低程序复杂度
  - 优点：类之间解耦
  - 缺点：中介者会庞大，变得复杂难以维护

命令模式

- 定义：将请求封装为一个对象，这样可以使用不同的请求参数化其他对象，并且能够支持请求的排队执行、记录日志、撤销等功能
- 适用场景：请求的调用者和接收者需要解耦，使得调用者和接收者不直接交互
- 适用场景：需要抽象出等待执行的行为
- 优缺点
  - 优点：降低耦合
  - 优点：容易扩展新命令或者一组命令
  - 缺点：命令的无限扩展会增加类的数量，提高系统实现复杂度
- 源码应用：Hystrix：HystrixCommand