结构模式

结构型模式描述如何将类或对象按某种布局组成更大的结构。它分为类结构型模式和对象结构型模式，前者采用继承机制来组织接口和类，后者采用组合或聚合来组合对象。

由于组合关系或聚合关系比继承关系耦合度低，满足“合成复用原则”，所以对象结构型模式比类结构型模式具有更大的灵活性。

结构型模式分为以下 7 种

- 代理模式
- 适配器模式
- 装饰者模式
- 桥接模式
- 外观模式
- 组合模式
- 享元模式

代理模式

由于某些原因需要给某对象提供一个代理以控制对该对象的访问。这时，访问对象不适合或者不能直接引用目标对象，代理对象作为访问对象和目标对象之间的中介。

Java中的代理按照代理类生成时机不同又分为静态代理和动态代理。静态代理代理类在编译期就生成，而动态代理代理类则是在Java运行时动态生成。动态代理又有JDK代理和CGLib代理两种。

- 静态代理
- 动态代理
    - CGLIB代理
    - JDK代理

JDK代理和CGLIB代理

使用CGLib实现动态代理，CGLib底层采用ASM字节码生成框架，使用字节码技术生成代理类，在JDK1.6之前比使用Java反射效率要高。唯一需要注意的是，CGLib不能对声明为final的类或者方法进行代理，因为CGLib原理是动态生成被代理类的子类。

在JDK1.6、JDK1.7、JDK1.8逐步对JDK动态代理优化之后，在调用次数较少的情况下，JDK代理效率高于CGLib代理效率，只有当进行大量调用的时候，JDK1.6和JDK1.7比CGLib代理效率低一点，但是到JDK1.8的时候，JDK代理效率高于CGLib代理。所以如果有接口使用JDK动态代理，如果没有接口使用CGLIB代理。

动态代理和静态代理

动态代理与静态代理相比较，最大的好处是接口中声明的所有方法都被转移到调用处理器一个集中的方法中处理（InvocationHandler.invoke）。这样，在接口方法数量比较多的时候，我们可以进行灵活处理，而不需要像静态代理那样每一个方法进行中转。

如果接口增加一个方法，静态代理模式除了所有实现类需要实现这个方法外，所有代理类也需要实现此方法。增加了代码维护的复杂度。而动态代理不会出现该问题

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，使得原本由于接口不兼容而不能一起工作的那些类能一起工作。

适配器模式分为类适配器模式和对象适配器模式，前者类之间的耦合度比后者高，且要求程序员了解现有组件库中的相关组件的内部结构，所以应用相对较少些。

- **类适配器模式（耦合度高一些）**

  实现方式：定义一个适配器类来实现当前系统的业务接口，同时又继承现有组件库中已经存在的组件。

  类适配器模式违背了合成复用原则。类适配器是客户类有一个接口规范的情况下可用，反之不可用。

- **对象适配器模式（用的比较多）**

  实现方式：对象适配器模式可采用将现有组件库中已经实现的组件引入适配器类中，该类同时实现当前系统的业务接口。

- 静态代理和装饰者模式的区别：
  - 相同点

        - 都要实现与目标类相同的业务接口
        - 在两个类中都要声明目标对象
        - 都可以在不修改目标类的前提下增强目标方法
    - 不同点
        - 目的不同，装饰者是为了增强目标对象，静态代理是为了保护和隐藏目标对象
        - 获取目标对象构建的地方不同，装饰者是由外界传递进来，可以通过构造方法传递，静态代理是在代理类内部创建，以此来隐藏目标对象

装饰者模式

定义：指在不改变现有对象结构的情况下，动态地给该对象增加一些职责（即增加其额外功能）的模式

- 饰者模式可以带来比继承更加灵活性的扩展功能，使用更加方便，可以通过组合不同的装饰者对象来获取具有不同行为状态的多样化的结果。装饰者模式比继承更具良好的扩展性，完美的遵循开闭原则，继承是静态的附加责任，装饰者则是动态的附加责任。
- 装饰类和被装饰类可以独立发展，不会相互耦合，装饰模式是继承的一个替代模式，装饰模式可以动态扩展一个实现类的功能。

```
public interface IReportService {

    void generateReport();
}

public class CommonReportImpl implements IReportService {
    @Override
    public void generateReport() {
        System.out.println("常见病");
    }
}

public class VisionReportImpl implements IReportService {
    @Override
    public void generateReport() {
        System.out.println("视力筛查报告");
    }
}

public abstract class ReportDecorator implements IReportService {

    private final IReportService iReportService;

    public ReportDecorator(IReportService iReportService) {
```

```
        this.iReportService = iReportService;
    }

    @Override
    public void generateReport() {
        iReportService.generateReport();
    }
}

public class ImportReportDecorator extends ReportDecorator{
    public ImportReportDecorator(IReportService iReportService) {
        super(iReportService);
    }

    @Override
    public void generateReport() {
        System.out.println("导入报告");
        super.generateReport();
    }
}

public class Test {

    public static void main(String[] args) {

        CommonReportImpl iReportService = new CommonReportImpl();
        ReportDecorator reportDecorator = new ImportReportDecorator(iReportService);
        reportDecorator.generateReport();
    }
}
```

❓ **装饰者模式和代理模式的区别**

桥接模式

将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现，从而降低了抽象和实现这两个可变维度的耦合度。

- 桥接模式提高了系统的可扩充性，在两个变化维度中任意扩展一个维度，都不需要修改原有系统。
- 实现细节对客户透明
- 使用场景
    - 当一个类存在两个独立变化的维度，且这两个维度都需要进行扩展时。
    - 当一个系统不希望使用继承或因为多层次继承导致系统类的个数急剧增加时(类爆炸)。

    - 当一个系统需要在构件的抽象化角色和具体化角色之间增加更多的灵活性时。避免在两个层次之间建立静态的继承联系，通过桥接模式可以使它们在抽象层建立一个关联关系。

外观模式(门面模式)

又名门面模式，是一种通过为多个复杂的子系统提供一个一致的接口，而使这些子系统更加容易被访问的模式。该模式对外有一个统一接口，外部应用程序不用关心内部子系统的具体的细节，这样会大大降低应用程序的复杂度，提高了程序的可维护性。

优缺点

- 优点
    - 降低了子系统与客户端之间的耦合度，使得子系统的变化不会影响调用它的客户类。
    - 对客户屏蔽了子系统组件，减少了客户处理的对象数目，并使得子系统使用起来更加容易。
- 缺点
    - 不符合开闭原则，修改很麻烦

使用场景

- 对分层结构系统构建时，使用外观模式定义子系统中每层的入口点可以简化子系统之间的依赖关系。
- 当一个复杂系统的子系统很多时，外观模式可以为系统设计一个简单的接口供外界访问。
- 当客户端与多个子系统之间存在很大的联系时，引入外观模式可将它们分离，从而提高子系统的独立性和可移植性。

组合模式

又名部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式根据树形结构来组合对象，用来表示部分以及整体层次，这种类型的设计模式属于结构模式。他创建了对象组的树形结构。

- 透明组合模式

  透明组合模式，抽象根节点角色声明了所有用于管理成员对象的方法。这样做的好处是确保所有的构件类都有相同的接口。透明组合模式也是组合模式的标准形式。缺点是不够安全，因为叶子对象和容器对象本质上是有区别的，叶子对象不可能有下一层对象。编译阶段不会有异常，但是运行时就可能存在问题。

- 安全组合模式

  在安全组合模式中，在抽象构建角色中没有声明任何用于管理成员对象的方法，而是在树枝节点中声明并实现这些方法。安全组合模式的缺点是不够透明，因为叶子构建和容器构建具有不同的方法。却容器构建中用于管理成员对象的方法并没有在抽象构建类中定义，因此客户端不能完全针对抽象编程，必须哟呀区别的对待叶子、容器的构建。

优点

- 组合模式可以清楚的定义分层次的复杂对象，表示对象的全部或部分层次，他让客户端忽略了层次的差异，方便对整个层次结构进行控制。
- 客户端可以一直的使用一个组合结构或其中单个对象，不必关心处理的是单个对象还是整个结构，简化了客户端的代码
- 在组合模式中添加新的树枝节点和叶子结点都非常方便，无须对现有的类库进行修改，符合“开闭原则”
- 组合模式为树形结构的面向对象实现提供了一种灵活的解决方案，通过叶子结点和树枝节点的递归，可以形成复杂的树形结构，但是对树形结构的控制却非常简单。

享元模式

运用共享技术来有效的支持大量细粒度对象的复用。他通过共享已经存在的对象来大幅减少需要创建对象的数量、避免大量相似对象的开销，从而提高系统资源的利用率。

优缺点

- 优点
  - 减少对象的创建数量，节约系统资源，提高系统性能
  - 外部状态相对独立且不影响内部状态
- 缺点

- 为了使对象可以共享，需要将对象的部分外部化，分离内部和外部的状态，使程序逻辑复杂