[设计模式简介 2](#_Toc8549)

[设计模式的类型 2](#_Toc5594)

[创建型模式（Creational Patterns） 2](#_Toc3185)

[结构型模式（Structural Patterns） 2](#_Toc25278)

[行为型模式（Behavioral Patterns） 2](#_Toc20433)

[J2EE 设计模式 2](#_Toc7703)

[设计模式的六大原则 2](#_Toc627)

[工厂模式 3](#_Toc30914)

[抽象工厂模式 3](#_Toc14560)

[单例模式 3](#_Toc10197)

[建造者模式 3](#_Toc17341)

[原型模式 3](#_Toc4513)

[适配器模式 4](#_Toc28492)

[桥接模式 4](#_Toc15283)

[过滤器模式 4](#_Toc13304)

[组合模式 4](#_Toc18862)

[装饰器模式 4](#_Toc2092)

[外观模式 5](#_Toc7232)

[享元模式 5](#_Toc17932)

[代理模式 5](#_Toc22657)

# 设计模式简介

对接口编程而不是对实现编程。

优先使用对象组合而不是继承。

作用：重用代码、代码易于理解、保证代码可靠性。解耦？

## 设计模式的类型

### 创建型模式（Creational Patterns）

在创建对象的同时隐藏创建逻辑的方式，而不是使用 new 运算符直接实例化对象。

包括：工厂模式，抽象工厂模式，单例模式，建造者模式，原型模式，

### 结构型模式（Structural Patterns）

关注类和对象的组合。继承的概念被用来组合接口和定义组合对象获得新功能的方式。

包括：适配器模式，桥接模式，过滤器模式，组合模式，装饰器模式，外观模式，享元模式，代理模式

### 行为型模式（Behavioral Patterns）

关注对象之间的通信。

包括：责任链模式，命令模式，解释器模式，迭代器模式，中介者模式，备忘录模式，观察者模式，状态模式，空对象模式，策略模式，模板模式，访问者模式。

### J2EE 设计模式

关注表示层。由 Sun Java Center 鉴定。

MVC 模式，业务代表模式，组合实体模式，数据访问对象模式，前端控制器模式，截过滤器模式，服务定位器模式，传输对象模式。

## 设计模式的六大原则

开闭原则：对扩展开放，对修改关闭。实现热插拔，提高扩展性。

里氏代换原则：实现抽象的规范，实现子父类互相替换；

依赖倒转原则：针对接口编程，实现开闭原则的基础；

接口隔离原则：降低耦合度，接口单独设计，互相隔离；

迪米特法则，又称不知道原则：功能模块尽量独立；

合成复用原则：尽量使用聚合，组合，而不是继承；

# 工厂模式

通过工厂方法返回工厂接口实例；即一个方法可以返回一个基类、接口的所有子类对象实例。

通过反射实现对象实例化。

主要解决：主要解决接口选择的问题。

问题：

如何实现反射获取对象的类型检查？泛型参数同时传入类型T，限制Shape？

传入Class参数时，泛型参数无法实现同时传入类型T，范围约束Shape。即要么不能自动转换，要么不能检验输入类型。

# 抽象工厂模式

围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂。

抽象工厂为工厂基类，规定工厂方法；子类工厂继承抽象工厂，实现方法。通过多态调用实现实例获取。

主要解决：主要解决接口选择的问题。

问题：

使用一个工厂中多个工厂方法？代码耦合？

# 单例模式

一个类负责创建自己的对象，同时确保只有单个对象被创建。构造函数私有。

实现方式：懒汉模式、饿汉模式、双验锁模式、枚举模式。

主要解决：一个全局使用的类频繁地创建与销毁。

对象实例要求唯一性，或创建实例开销过大。

# 建造者模式

使用多个简单的对象构建成复杂对象。

主要解决：软件系统中复杂对象的创建工作。

将复杂对象分解为多个基本对象，通过组合实现不同复杂对象的创建。

# 原型模式

创建重复的对象，同时又能保证性能。

实现一个原型接口Cloneable，用于创建当前对象的克隆。

主要解决：在运行期建立和删除原型。

数据、资源的复用。

ThreadLocal，效率低？

# 适配器模式

作为两个不兼容的接口之间的桥梁。在现存对象中加入新功能的适配器实例，调用适配器实例方法实现新功能。解耦。不必重写代码，局部更新即可。

主要解决：软件系统中将一些"现存的对象"放到新环境中。

# 桥接模式

把抽象化与实现化解耦，使得二者可以独立变化。

分解变化维度，保证一个时刻只有一个维度变化。用一个接口一个维度，多个接口组合实现多维度。

主要解决：在有多种可能会变化的情况下，用继承会造成类爆炸问题，扩展起来不灵活。

# 过滤器模式

允许开发人员使用不同的标准来过滤一组对象，通过逻辑运算以解耦的方式连接起来。

结合多个标准来获得单一标准。分解标准、组合标准。

# 组合模式

处理树状结构。

把一组相似的对象当作一个单一的对象。

在对象内部增加List属性，代表下级对象；通过对List的操作完成对下级的操作。

# 装饰器模式

包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。

抽象的装饰类实现原类根接口，并引入接口的实例变量；具体装饰类实现功能扩展，在方法中通过实例变量调用原方法，并增加新逻辑。

没有继承原类，实现了原类的功能扩展。

# 外观模式

向现有的系统添加一个接口，供客户端访问，隐藏系统的复杂性。

主要解决：降低访问复杂系统的内部子系统时的复杂度，简化客户端与之的接口。

封装。

# 享元模式

减少创建对象的数量，以减少内存占用和提高性能。对象存在时复用对象，类似多个单例模式的集合。

关键代码：用 HashMap 存储这些对象。

# 代理模式

一个类代表另一个类的功能。

主要解决：在直接访问对象时会产生问题。

关键代码：实现与被代理类组合。将被代理类传入代理类中，而后调用代理类方法。

# 责任链模式

为请求创建了一个接收者对象的链。

在这种模式中，通常每个接收者都包含对另一个接收者的引用。如果一个对象不能处理该请求，那么它会把相同的请求传给下一个接收者，依此类推。

主要解决：职责链上的处理者负责处理请求。

关键代码：Handler 里面聚合它自己，在 HanleRequest 里判断是否合适，如果没达到条件则向下传递，向谁传递之前 set 进去。

拿到请求，判断自己是否处理；而后判断是否继续传递。

# 命令模式

一种数据驱动的设计模式。

请求以命令的形式包裹在对象中，并传给调用对象。调用对象寻找可以处理该命令的合适的对象，并把该命令传给相应的对象，该对象执行命令。

关键代码：三个角色：received命令执行对象，Command ，invoker命令对象入口

一个行为模式一个对象，请求中附带不同的行为实例。

# 解释器模式

评估语言的语法或表达式的方式。

自定义语法规则，应用少。

# 迭代器模式

用于顺序访问集合对象的元素。

通过内部类实现Iterator接口，在外部类提供getIterator()方法，获取Iterator。

主要解决：不同的方式来遍历整个整合对象。

# 中介者模式

通过中介通信，降低多个对象和类之间的通信复杂性。

主要解决：对象与对象之间存在大量的关联关系。

关键代码：对象之间的通信封装到一个类中单独处理。

# 备忘录模式

保存一个对象的某个状态，以便在适当的时候恢复对象。

关键代码：客户不与备忘录类耦合，与备忘录管理类耦合。备忘录类通过List存储备份状态实例，备忘录管理类操作备忘录，备份或者提取状态。

使用原型模式+备忘录模式，节约内存。

# 观察者模式

当对象间存在一对多关系时，则使用观察者模式。比如，当一个对象被修改时，则会自动通知它的依赖对象。

关键代码：在抽象类里有一个 ArrayList 存放观察者们。