设计模式按照使用场景可以分为三大类: 创建型模式（Creational Patterns）、结构型模式（Structural Patterns）、行为型模式（Behavioral Patterns）

创建型模式: 对对象的实例化过程进行抽象，这使得一个系统可以不用关心这些对象是如何创建，组合，呈现的，对于类创建模式来说通过使用继承改变实例化的类，对于对象创建模式来说通过使用代理来实例化所需要的对象。

工厂模式:

原型模式:

构建者模式:

单例模式:

结构型模式: 通过对多个类和对象进行组合得到复杂结构的类，一般使用继承继承或者成员变量引用形式来实现。

适配器模式:

桥接模式:

过滤器模式:

组合模式:

装饰者模式:

门面模式:

享元模式:

代理模式:

行为型模式: 行为模式不仅表达了对象和类，还表达了他们之间的交互，涉及到了对象和算法的分配。

责任链模式:

命令模式:

解析器模式:

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:

# 单例模式:

1. 意义:保证某个类在系统中是全局的并且只有一个实例
2. 要点: 一是单例模式的类只提供私有的构造函数，二是类定义中含有一个该类的静态私有对象，三是该类提供了一个静态的公有的函数用于创建或获取它本身的静态私有对象。
3. 应用实例:
4. 自己代码实现例子

//懒汉式实现单例模式等到调用的时候才进行实例化  
//饿汉式就是在类中一开始就已经实例化  
public class Ticket{  
 public int tickAmount;  
 //构造函数私有化  
 private Ticket() {  
 this.tickAmount=1000;  
 }  
 //定义静态类,但是没有实例化  
 private static Ticket *ticket* = null;//如果是饿汉式就是private static Ticket ticket = new Ticket();  
 //外部访问方法  
 public static Ticket getInstance() {  
 if (*ticket* == null) {  
 *ticket* = new Ticket();  
 }  
 return *ticket*;  
 }  
}

# 工厂模式:

1. 意义:
2. 要点:
3. 应用实例:SpringContext
4. 自己代码实现:

# 代理模式:

1. 意义:为其他对象提供一种代理以控制对这个对象的访问。
2. 要点:代理模式分为普通代理和动态代理
3. 应用实例:
4. 自己代码实现:

# 调停者模式:

1. 意义:用于模块之间的解耦,通过避免对象相显式的指向对方从而降低耦合.(对象的行为模式)
2. 要点:
3. **抽象调停者(Mediator)角色：**定义出同事对象到调停者对象的接口，其中主要方法是一个（或多个）事件方法。
4. **具体调停者(ConcreteMediator)角色：**实现了抽象调停者所声明的事件方法。具体调停者知晓所有的具体同事类，并负责具体的协调各同事对象的交互关系。
5. **抽象同事类(Colleague)角色：**定义出调停者到同事对象的接口。同事对象只知道调停者而不知道其余的同事对象。
6. **具体同事类(ConcreteColleague)角色：**所有的具体同事类均从抽象同事类继承而来。实现自己的业务，在需要与其他同事通信的时候，就与持有的调停者通信，调停者会负责与其他的同事交互。
7. 应用实例:ESB