# 策略模式Strategy

## 模式定义

## 核心思想

数据固定，目的固定，过程自定义

数据有了，目的明确，如何实现？

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

## 特点

## 不足之处

## 实际应用

## 其它

# 观察者模式Observer

## 模式定义

定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。

## 核心思想

## 模式涉及名词

1. 可观察者observable(主题subject)
2. 观察者observer
3. 注册register

## 结构图

## 设计原则体现

1. 为交互对象之间的松耦合而努力

可观察者与观察者都是使用接口，

## 使用步骤

1. 定义主题接口和观察者接口(针对接口编程)；
2. 具体主题实现类：
   * 1. 维护观察者注册信息，主要用于对观察者推送消息；
     2. 定义一个内容更状态，用于判断是否需要向观察者们推送消息；
3. 具体观察者实现类：维护一个主题对象，主要用于后期取消订阅主题方便；

## 特点

## 不足之处

## 实际应用

1. jdk的java.util.Observable和java.util.Observer
2. Jdk的Swing的JButton

## 其它

# 装饰器模式

## 模式定义

## 核心思想

当想要对已有的类进行功能增强时，可以定义类，将已有对象传入，基于已有的功能，并提供加强功能，那么自定义的该类成为装饰类。

一句话：构造函数进行传递，进行功能增强。

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

1. 在装饰类中创建需要被装饰的类的对象，并私有化
2. 装饰类通常会通过构造方法接受被装饰的对象，并基于被装饰的对象的功能，提供更强的功能。
3. 重写需要增强的方法，不需要增强的方法调用被装饰类的方法

## 特点

## 不足之处

## 实际应用

1. BufferReader缓冲区技术
2. 使用Filter过滤器解决get提交中文乱码问题时，通过对增强request的getParameter方法来实现

## 其它

### 装饰与继承的区别

1. 装饰者模式比继承要灵活，避免了继承体系的臃肿，而且降低了类与类之间的关系
2. 装饰类因为是增强已有对象，具备的功能和已有的是相同的，只不过是提供了更强的功能

所以一般装饰类与被装饰类通常是属于一个体系(并列关系，同属于一个父类或接口).

# 单例模式Singleton

## 模式定义

## 核心思想

目的：保证类对象的唯一性.

单态(Single)，保证一个类的对象，在内存中的唯一性.

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

1. 在本类的成员位置，创建私有的静态的本类实例
2. 私有无参构造方法
3. 定义public static的方法，返回本类对象.

## 特点

1. 饿汉式
   1. 优点：开发中常见，线程安全的,
   2. 缺点：不管是否使用就创建对象，造成内存空间浪费。
2. 懒汉式
   1. 缺点：对象延迟加载，会出现安全问题，对象不唯一

## 不足之处

## 实际应用

## 其它

### 饿汉式

优点：开发中常见，线程安全的,

缺点：不管是否使用就创建对象，造成内存空间浪费。

模板代码:

|  |
| --- |
| class Single{  private static Single single = new Single();  private Singele(){}  public static Single getInstance(){  return single;  } |

}

### 懒汉式

#### 写法1

代码实现:

|  |
| --- |
| //1.私有的静态的本类实例  private static JDBCUtils instance;  //2.私有无参构造方法  private JDBCUtils(){}  //3.提供获取本类实例的公共的静态方法  public static JDBCUtils getInstance(){  if (instance==null) {  instance = new JDBCUtils();  }  return instance;  } |

存在问题:

对象延迟加载，会出现安全问题，对象不唯一

#### 写法2

如何写一个安全高效的懒汉式？

安全：可以使用 Synchronized

代码实现:

|  |
| --- |
| public class JDBCUtils{  //1.私有的静态的本类实例  private static JDBCUtils instance;  //2.私有无参构造方法  private JDBCUtils(){}  //3.提供获取本类实例的公共的静态方法  public static JDBCUtils getInstance(){  if (instance==null) {  //本类的字节码对象在内存中唯一，保证了锁的唯一性  synchronized (JDBCUtils.class) {  if (instance==null) {  instance = new JDBCUtils();  }  }  }  return instance;  }  } |

### 使用枚举实现单例

代码实现：

|  |
| --- |
| public enum JDBCUtils {  UTILS;  } |

枚举本身就是唯一的。

获取实例的方式：JDBCUtils.UTILS

# 工厂模式

## 模式定义

## 核心思想

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

## 特点

## 不足之处

## 实际应用

## 其它

# 代理模式

## 模式定义

## 核心思想

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

## 特点

## 不足之处

## 实际应用

(1)request方法加强

(2)service层的事务控制提取

## 其它

# 适配器模式

## 模式定义

## 核心思想

适配器模式把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作。

适配器模式的宗旨是：保留现有类提供的服务，修改其接口使其能满足用户的需要。

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

## 特点

## 不足之处

## 实际应用

## 其它

# 模板

## 模式定义

## 核心思想

## 模式涉及名词

## 结构图

## 设计原则体现

## 使用步骤

## 特点

## 不足之处

## 实际应用

## 其它