decorator模式

场景

（1）假设我们手头已经有了一个类，可以执行一些功能

（2）但是我们还是希望对这个类的**功能做一些增强**，此时怎么办呢？基于已有的类的功能基础之上，再增强一些功能，可以做装饰

public interface **Component** {

   void operate();

}

public class **ConcreteComponent implements Component** {

   public void operate() {

       System.out.println("组件的功能");

  }

}

public class **Decorator implements Component** {

   protected Component component;

   public Decorator(Component component) {

       this.component = component;

  }

   public void operate() {

       System.out.println("实现某些动态加强功能");

       component.operate();

       System.out.println("实现某些动态加强功能");

  }

}

装饰器模式有一些非常经典的实现

1. 比如java的io体系，可以一层包装一层，一层包装一层，外面的一层，都会对立面的一层进行功能的增强。

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(“”)));

br.close();

（2）还有就是spring的aop，aop这块可以基于动态代理的理念，装饰我们的目标对象，然后加入事务控制，日志打印之类的功能。

public interface UserService {

   void save(User user);

}

@Transactional

public class UserServiceImpl implements UserService {

   public void save(User user) {

       // 执行插数据库的操作

  }

}

// spring aop

public class UserServiceImplProxy implements UserSerivce {

   private UserSerivce userService;

   public void save(User user) {

       // 执行操作之前打开事务

       connection.startTransaction();

       // 执行目标的功能

       userService.save(user);

       // 执行完之后提交事务

       connection.commitTransaction();

  }

}

我们本次会在电商业务设计的时候，结合具体业务，来考虑如何运用这个模式

853769620