adapter模式【适配新旧接口规范 --- 新旧接口的方法一样，但规范变了】

场景介绍：

（1）假设我们做了一个第一版的一个系统，这个系统里有一个接口和一个实现类

（2）接着我们开始做第二版的系统，这个系统我们定义了一个新的接口，和新的实现类

（3）但是我们同时在第二版的系统中，也要使用第一版系统中定义的那个老接口和老实现类

【本质是：将旧接口的使用方法和新接口的使用方法统一起来 ---- 通过实现同一个 Target 接口（在新方法中，调用老方法）】

Target oldObject = new Adapter(new Adaptee());

Target newObject = new TargetImpl();

oldObject.request();

newObject.request();

 【老接口】

public class Adaptee {

   public void specificRequest() {

       System.out.println("老的方法实现");

  }

}

【新接口】

public interface **Target** {

   void request();

}

【适配器类：实现了新接口，持有老接口实现类的引用】

public class **Adapter implements Target** {

   【持有老接口的实现】

   private Adaptee adaptee;

   public Adapter(Adaptee adaptee) {

       this.adaptee = adaptee;

  }

   public void request() {

       adaptee.spcificRequest();

  }

}

我们手头有有一套系统v1.0版本的老代码，现在要在这个基础之上构建v1.0的新代码。然后有了一个新的接口，需要让别人面向新接口开发，此时就只能开发适配器，适配v1.0版本的类，但是实现的是v2.0的接口，别人调用v2.0的接口，内部却是基于v1.0的老类实现的功能

在实际企业开发中的使用场景

这个模式一般是在系统不断升级的过程中使用，对已经写好的老的类，写一套适配器来适配老类，但是提供新的接口，这个我们在后面系统升级的时候，可以去实践。在项目阶段二的时候，会去实践的，真的有版本升级的时候，才可以完美的去演示这个模式的使用效果。

还有一种情况，是对于已有的第三方类库，比如redis的客户端，或者是elasticsearch的客户端，他们提供了一套API，但是我们这里的要求是需要面向我们这里的DAO接口来进行编程，此时可以写一个适配器，将比如redis客户端的接口适配到我们的接口，这个我们会在本次课程来实践。

比如我们的DAO接口，要求的接口风格都是：save、update、remove、list、get，这些方法风格

DAORedisImpl，redis客户端，本身提供的是get、set、mset、mget，一套接口；适配器，DAORedisImpl就是一个适配器，这个适配器实现的是我们的DAO接口，在我们的save、update、remove等方法中，去调用redis客户端的get、set、mset、mget等方法。

这个东西，我们会从项目阶段二开始，大量的使用，我们就会开始有很多的外部依赖，不只是mysql，mybatis的mapper去做，不太好实践。项目阶段一我们暂时可能就不实践，项目阶段二，大量的实践。

853769620