The Adaptor Pattern
（适配器模式）

主讲：顾祝燕
组员：耿惠、何振芬、朱金凤

ADAPTER—在朋友聚会上碰到了一个美女Sarah，从香港来的，可我不会说粤语，她不会说普通话，只好求助于我的朋友kent了，他作为我和Sarah之间的Adapter，让我和Sarah可以相互交谈了(也不知道他会不会耍我)

适配器（变压器）模式：把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口原因不匹配而无法一起工作的两个类能够一起工作。适配类可以根据参数返还一个合适的实例给客户端。

第八章　适配器模式

适配器模式（别名：包装器）
将一个类的接口转换成客户希望的另外一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

Adapter Pattern(Another Name: Wrapper)
Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

一、概述

适配器模式是将一个类的接口（被适配者）转换成客户希望的另外一个接口（目标）的成熟模式，该模式中涉及有目标、被适配者和适配器。适配器模式的关键是建立一个适配器，这个适配器实现了目标接口并包含有被适配者的引用。

二、适配器模式的结构与使用

模式的结构中包括三种角色：

- 目标（Target）
- 被适配者（Adaptee）
- 适配器（Adapter）

模式的UML类图

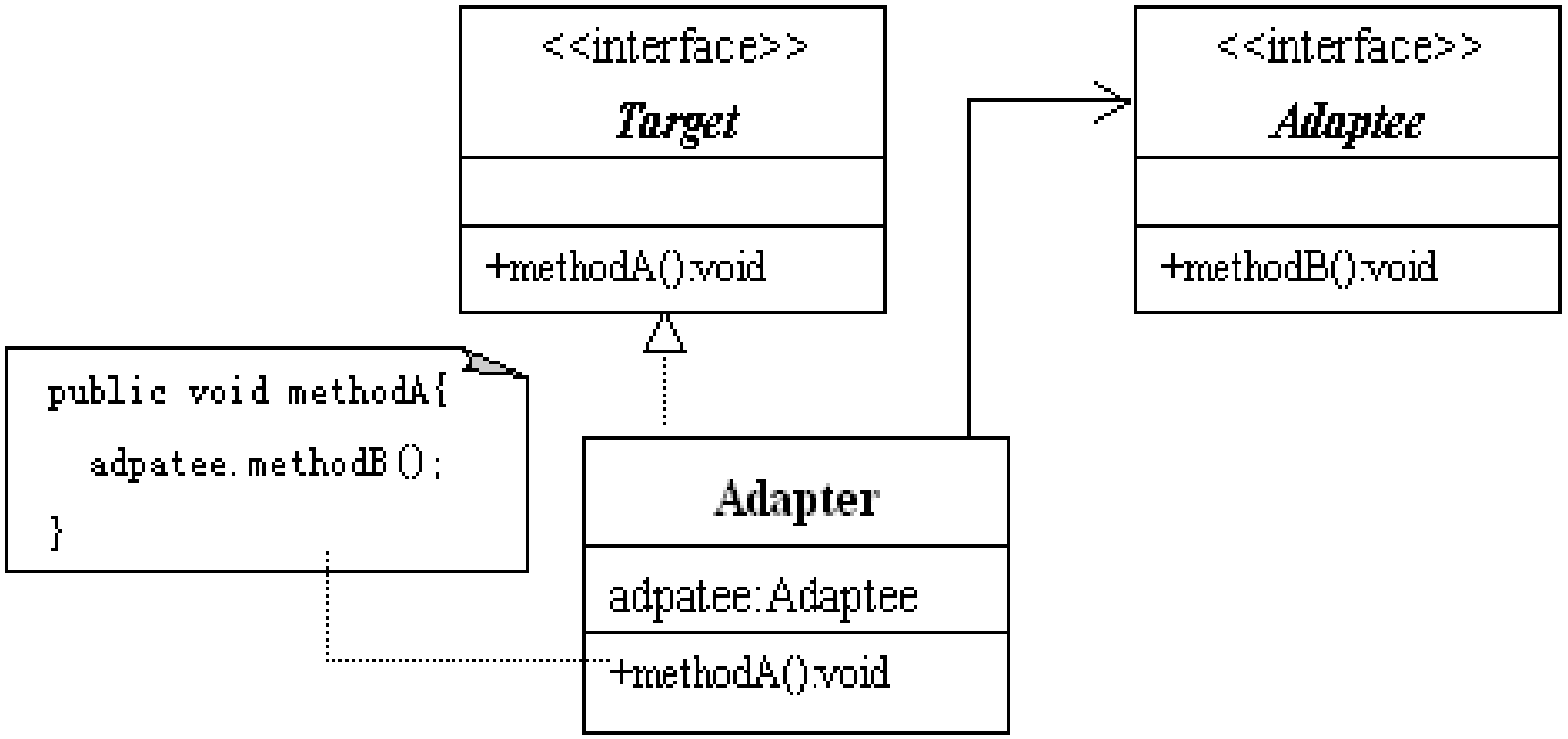


图 8.6　适配器模式的类图

模式的结构的描述与使用

1．目标（Target）：ThreeElectricOutlet.java

```
public interface ThreeElectricOutlet{
    public abstract void connectElectricCurrent();
}
```

2．被适配者（Adaptee）：TwoElectricOutlet.java

```
public interface TwoElectricOutlet{
    public abstract void connectElectricCurrent();
 }
```

3．适配器（Adapter） TreeElectricAdapter.java

```
public class TreeElectricAdapter implements ThreeElectricOutlet{
  TwoElectricOutlet outlet;
  TreeElectricAdapter(TwoElectricOutlet outlet){
     this.outlet=outlet;
  }
  public void connectElectricCurrent(){
     outlet.connectElectricCurrent();
  }
}
```

模式的结构的描述与使用

4．应用 Application.java_1

```
public class Application{
  public static void main(String args[]){
    ThreeElectricOutlet outlet;
    Wash wash=new Wash();
    outlet=wash;
    System.out.println("使用三相插座接通电流：");
    outlet.connectElectricCurrent();
    TV tv=new TV();
    TreeElectricAdapter adapter=new TreeElectricAdapter(tv);
    outlet=adapter;
    System.out.println("使用三相插座接通电流：");
    outlet.connectElectricCurrent();
  }
}
```

4．应用 Application.java_2

```
class Wash implements ThreeElectricOutlet{
  String name;
  Wash(){
    name="黄河洗衣机";
  }
  Wash(String s){
    name=s;
  }
  public void connectElectricCurrent(){
    turnOn();
  }
  public void turnOn(){
    System.out.println(name+"开始洗衣物。");
  }
}
```

4．应用 Application.java_3

```
class TV implements TwoElectricOutlet{
  String name;
  TV(){
    name="长江电视机";
  }
  TV(String s){
    name=s;
  }
  public void connectElectricCurrent(){
    turnOn();
  }
  public void turnOn(){
    System.out.println(name+"开始播放节目。");
  }
}
```

三、适配器模式的优点

- 目标（Target）和被适配者（Adaptee）是完全解耦的关系。
- 适配器模式满足“开-闭原则”。当添加一个实现Adaptee接口的新类时，不必修改Adapter，Adapter就能对这个新类的实例进行适配。

面向对象软件系统的适配问题

- 假设我们已经有一个软件系统，原来使用了一个第三方类库A。现在有一个新的第三方类库B，其功能等各方面都更加强大。我们希望用B来替换A，以改善我们的系统。但是B的接口与A不一样。那怎么办呢？

Adapter模式

- 定义
  - 将一个类的接口转换成客户端所期望的另一种接口，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。
- 别名
  - 包装器Wrapper

Adapter模式

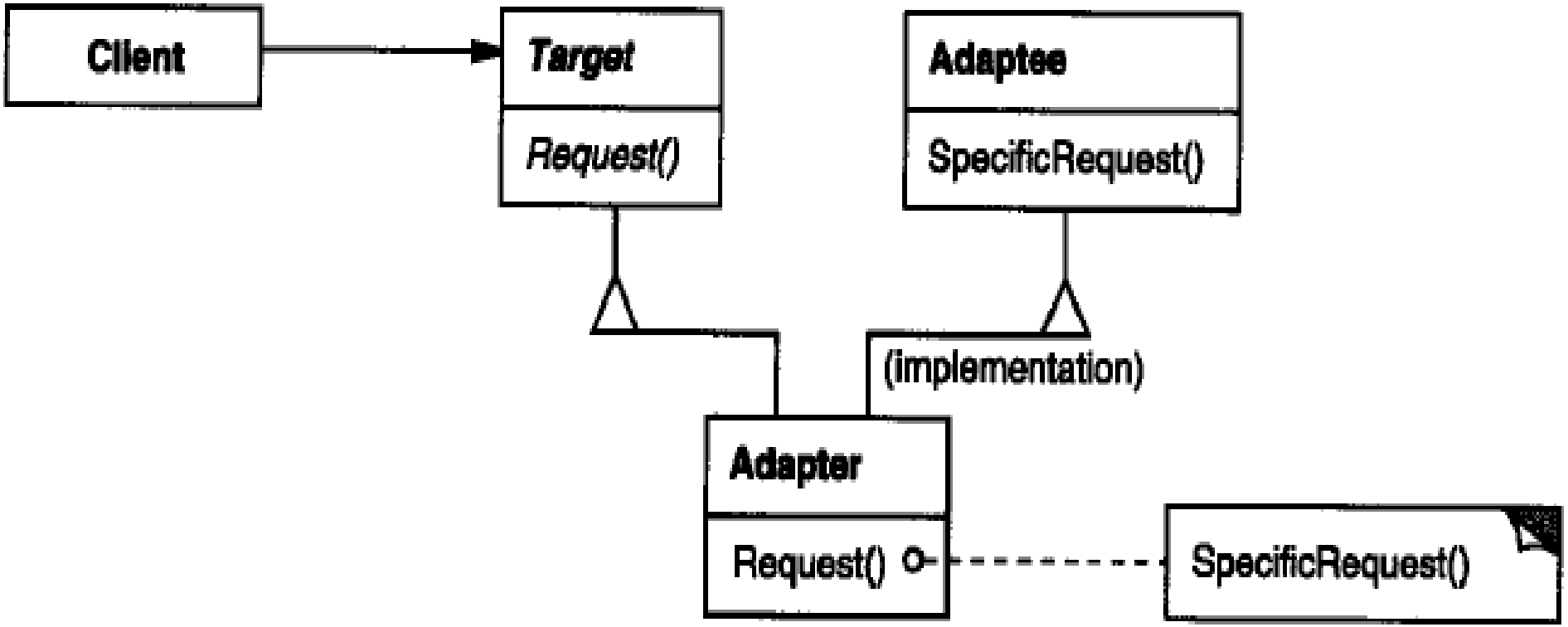
- 动机
  - 有时，为复用而设计的**工具箱类**不能够被复用的原因仅仅是因为它的接口与专业应用领域所需要的接口不匹配（名称不一样，参数不一样，等等）。
- 我们可以改变工具箱类使它兼容专业领域中的类的接口，但前提是必须有这个工具箱的源代码。然而即使我们得到了这些源代码，修改工具箱也是没有什么意义的；因为不应该仅仅为了实现一个应用，工具箱就不得不采用一些与特定领域相关的接口。动机（续）
  - 我们可以不用上面的方法，而定义一个适配器类，由它来适配工具箱的接口和专业应用的接口。我们可以用两种方法做这件事：
  - 1) 继承专业应用类的接口和工具箱类的实现。这种方法对应Adapter模式的类版本（多继承）
  - 2) 将工具箱类的实例作为适配器类的组成部分，并且使用工具箱的接口实现适配器类。这种方法对应Adapter模式的对象版本。

Adapter模式

- 适用性
  以下情况使用Ａｄａｐｔｅｒ模式
  - 你想使用一个已经存在的类，而它的接口不符合你的需求。
  - 你想创建一个可以复用的类，该类可以与其他不相关的类或不可预见的类（即那些接口可能不一定兼容的类）协同工作。

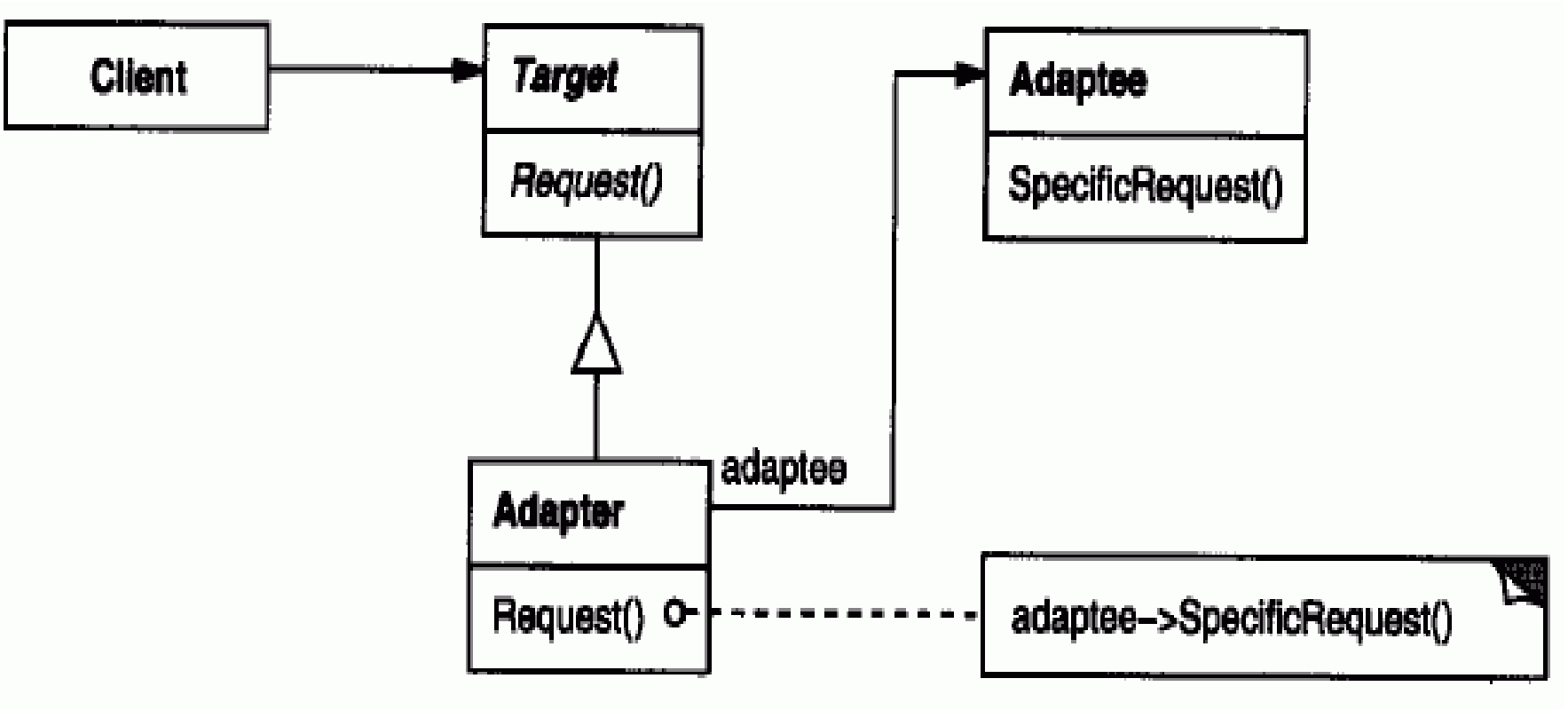
Adapter模式

- 结构（类版本）



Adapter模式

- 结构（对象版本）



基于类的Adapter模式

- 基于类的Adapter模式的一般结构如下：Adaptee类为Adapter的父类，Adaptee类为适配源，适配目标（接口）也是Adapter的父类；基于类的Adapter模式比较适合应用于Adapter想修改Adaptee的部分方法的情况。

基于对象的Adapter模式

- 基于对象的Adapter模式的一般结构如下：Adaptee类对象为Adapter所依赖，适配目标（接口）是Adapter的父类；
- 基于对象的Adapter模式比较适合应用于Adapter想为Adaptee添加新的方法的情况。但在Adaptee类的方法与Adapter类的方法不同名而实现相同功能的情况下，我们一般也使用基于对象的Adapter模式，

Adapter模式

- 参与者
  - Target
    - Client使用的与特定领域相关的“接口”。
  - Client
    - 与符合Target接口的对象协同的专业系统。
  - Adaptee
    - 一个已经存在的“接口”，它具有Client要求的功能但不符合Client的接口要求。这个接口需要适配。
  - Adapter
    - 对Adaptee的接口与Target接口进行适配

Adapter模式

- 协作
  - Client在Adapter实例上调用一些操作（请求）。接着适配器调用Adaptee的操作实现这个请求。

- 效果（类适配器和对象适配器有不同的权衡）
  - 类适配器
    - 用一个具体的Adapter类对Adaptee和Target进行匹配。结果是当我们想要匹配一个类以及所有它的子类时，类Adapter将不能胜任工作。
    - 使得Adapter可以重定义Adaptee的部分行为，因为Adapter是Adaptee的一个子类。
    - 仅仅引入了一个对象，并不需要额外的指针以间接得到adaptee。

Adapter模式

- 效果（类适配器和对象适配器有不同的权衡）
  - 对象适配器则
    - 允许一个Adapter与多个Adaptee—即Adaptee本身以及它的所有子类（如果有子类的话）同时工作。Adapter也可以一次给所有的Adaptee添加功能。
    - 使得重定义Adaptee的行为比较困难。这就需要生成Adaptee的子类并且使得Adapter引用这个子类而不是引用Adaptee本身。