适配器模式

1 适配器模式的定义

> 将一个类的接口转换成客户希望的而另外一个接口。适配器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。

2 适配器模式的结构和说明

2.1 适配器模式的结构如下图所示：

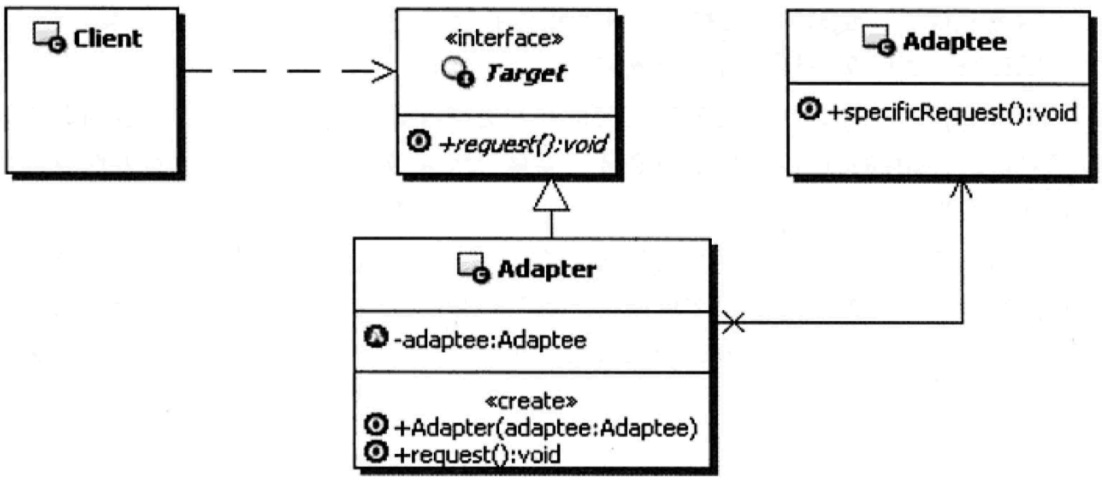


图 4.6　适配器模式的结构图

- Client：客户端，调用自己需要的领域接口 Target。
- Target：定义客户端需要的跟特定领域相关的接口。
- Adaptee：已经存在的接口，通常能满足客户端的功能要求，但是接口与客户端要求的特定领域接口不一致，需要被适配。
- Adapter：适配器，把 Adaptee 适配成为 Client 需要的 Target。

2.2 适配器模式示例代码：

- Target接口定义的示例代码如下：

```java
/**
 * 定义客户端使用的接口，与特定领域相关
 */
public interface Target{
  //示例犯法，客户端请求处理的方法
  public void request();
}
```

- 需要被适配的对象定义示例代码：

```java
/**
 * 已经存在的接口，这个接口需要被适配
 */
public class Adaptee{
  //原本已经被实现的方法
  public void specficRequest(){
   //具体的功能处理
  }
}
```

- 下面是适配器对象的基本实现：

```java
/**
 * 适配器
 */
public class Adapter implements Target{
  //持有需要被适配的接口对象
  private Adaptee adaptee;

  //构造方法传入需要被适配的对象
  public Adapter(Adaptee adaptee){
    this.adaptee = adaptee;
  }

  public void requet(){
    //转调已经实现了的方法，进行适配
    adaptee.specficRequest();
  }
}
```

- 客户端代码

```java
public class Client{
  public static void main(String[] args){
    Adaptee adaptee = new Adaptee();
    Target target = new Adapter(adaptee);
    target.request();
  }
}
```

3 模式讲解

3.1 认识适配器模式

3.1.1 模式的功能

适配器模式的主要功能是进行转换匹配，目的是复用已有的功能，而不是来实现新的接口。也就是说，客户端需要的功能应该是已经实现好了的，不需要适配器模式来实现，适配器模式主要负责把不兼容的接口转换成客户端期望的样子。

当然在适配器里面也可以实现功能，称这种适配器为智能适配器。

3.1.2 Adaptee和Target的关系

适配器模式中被适配的接口Adaptee和适配成为的接口Target是没有关联的，即Adaptee和Target中的方法既可以相同，也可以不同。

3.1.3 对象组合

适配器的实现方式其实是依靠对象组合的方法。通过给适配器对象组合被适配的对象，然后客户端调用Target的时候，适配器会把相应的功能委托给被适配的对象去完成。

3.1.4 适配器模式的调用顺序示意图

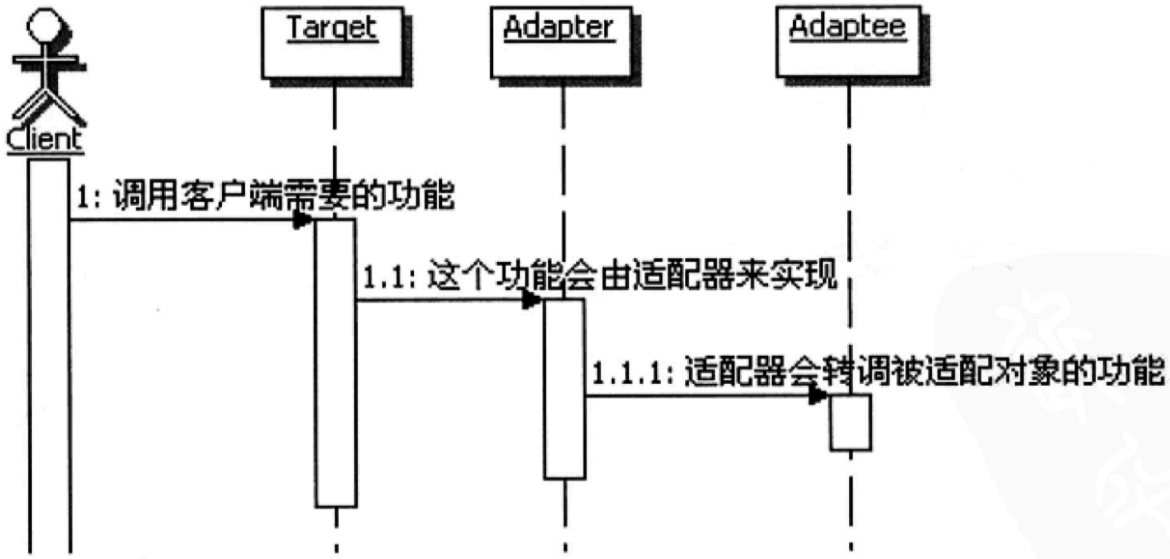


图 4.11　适配器模式的调用顺序示意图

3.2 适配器模式的实现

3.2.1 适配器的常见实现

在实现适配器的时候，适配器通常是一个类，一般会让适配器类去实现Target接口，然后在适配器的具体实现里面调用Adaptee，也就是说适配器通常是一个Target类型。

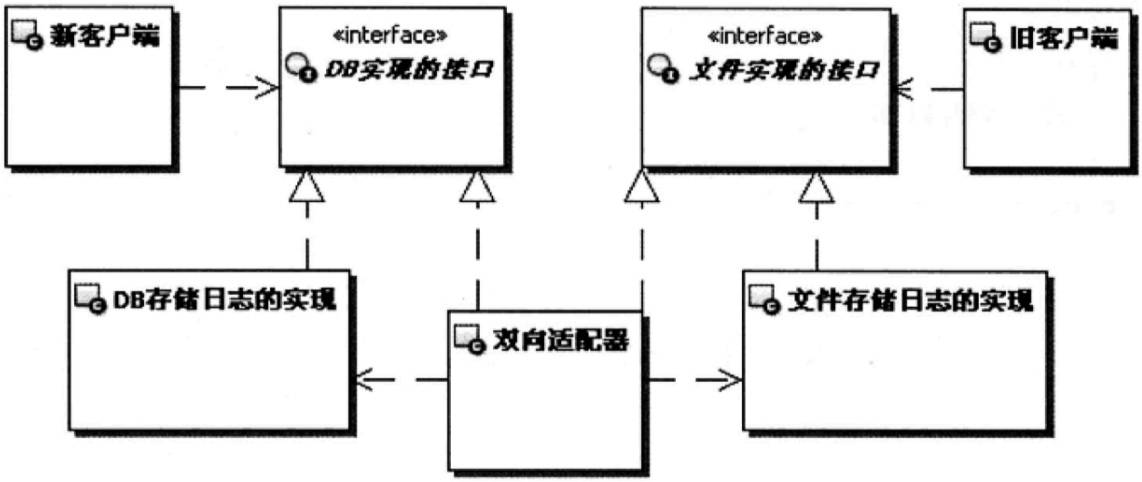
3.2.2 缺省适配

缺省适配的意思是，为一个接口提供缺省实现。有了它，就不用直接去实现接口，而是采用继承这个缺省适配对象，从而让子类可以有选择地去覆盖实现需要的方法，对于不需要的方法，使用缺省适配的方法就可以了。

3.3 双向适配器

适配器也可以实现双向的适配，这个适配器可以同时当作Target和Adaptee来使用。

这里以《研磨设计模式》中的例子示意：



双向适配器对象示意代码如下：

```
//需要同时实现需要适配的两个接口
public class TwoDirectAdapter implements
LogDbOperateApi,LogFileOperateApi{
  //持有双向适配的日志接口对象
  private LogFileOperateApi fileLog;
  private LogDbOperateApi dbLog;

  public TwoDirectAdapter(LogFileOperateApi fileLog,LogDbOperateApi
dbLog){
    this.fileLog = fileLog;
    this.dbLog = dbLog;
  }

  //下面就是实现客户端所需功能的方法
  //......
}
```

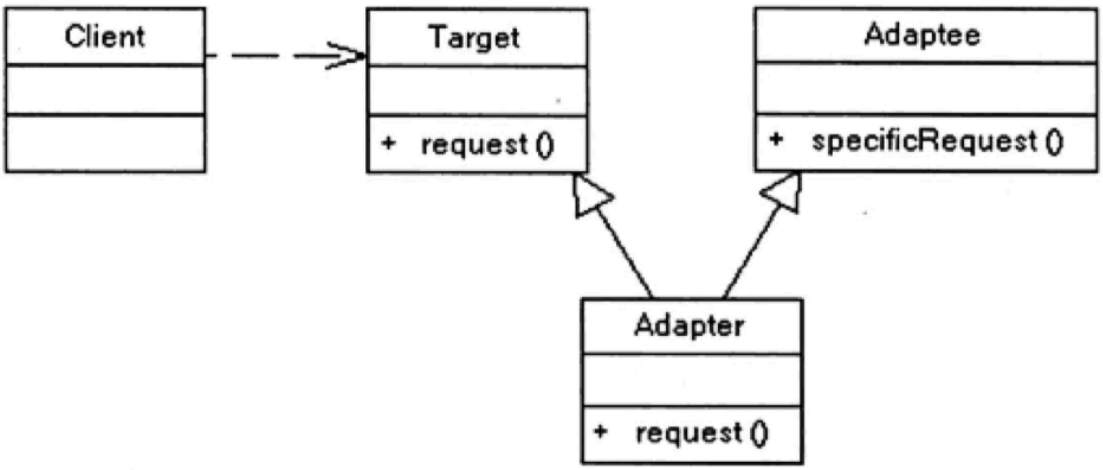
3.4 类适配器

在标准的适配器模式中，根据适配器的实现方式，把适配器分成了两种，一种是对象适配器，另一种是类适配器。

- 对象适配器的实现依赖于对象组合，如同前面的实现实例。
- 类适配器采用多重继承对一个接口与另一个接口进行匹配。

3.4.1 类适配器

类适配器的结构示意图：



从结构图上可以看出，类适配器是通过继承来实现接口适配的，标准的设计模式中，类适配器是同时继承Target和Adaptee的，也几居室一个多重继承，但这在java里是不被支持的，也就是说java中是不能实现标准的类适配器的。

但是Java中有一种变通的方式，也能够使用继承来实现接口的适配，即让适配器去实现Target的接口，然后继承Adaptee的实现。如下所示：

```
1 public class Adapter extends Adaptee implements Target{...}
```

3.4.2 类适配器和对象适配器的权衡

- 从实现上：类适配器使用对象继承的方式，是静态的定义方式；而对象适配器使用对象组合的方式，是动态组合的方式。
- 对于类适配器，由于适配器直接继承了Adaptee，使得适配器不能和Adaptee的子类一起工作，因为继承是静态的关系，当适配器继承了Adaptee后，就不可能再去处理Adaptee的子类了。对于对象适配器，允许一个Adapter和多个Adaptee，包括Adaptee和它所有的子类一起工作。因为对象适配器采用的是对象组合的关系，只要对象类型正确，是不是子类都无所谓。
- 对于类适配器，适配器可以重定义Adaptee的部分行为，相当于子类覆盖父类的部分实现方法。对于对象适配器，要重定义Adaptee的行为比较困难，这种情况下，需要定义Adaptee的子类来实现重定义，然后让适配器组合子类。
- 对于类适配器，仅仅引入了一个对象，并不需要额外的引用来间接得到Adaptee。对于对象适配器，需要额外的引用来间接得到Adaptee。

在java开发中，建议尽量使用对象适配器的实现方式。

3.5 适配器模式的优缺点

适配器模式有如下优点：

- 更好的复用性：如果功能是已经有了的，只是接口不兼容，那么通过适配器模式就

可以让这些功能得到更好地复用。
- 更好的可扩展性：在实现适配器功能的时候，可以调用自己开发的功能，从而自然地扩展系统的功能。

适配器模式有如下缺点：

- 过多的使用适配器，会让系统非常凌乱，不容易整体进行把握。

3.6 思考适配器模式

3.6.1 适配器模式的本质

> 转换匹配，复用功能

3.6.2 何时选用适配器模式

- 如果你想要使用一个已经存在的类，但是它的接口不符合你的需求，这种情况可以使用适配器模式，来把已有的实现转换成你需要的接口。
- 如果你想创建一个可以复用的类，这个类可能和一些不兼容的类一起工作，这种情况可以使用适配器模式，到时候需要什么就适配什么。
- 如果你想使用一些已经存在的子类，但是不可能对每一个子类都进行适配，这种情况可以选择对象适配器，直接适配这些子类的父类就尅一了。