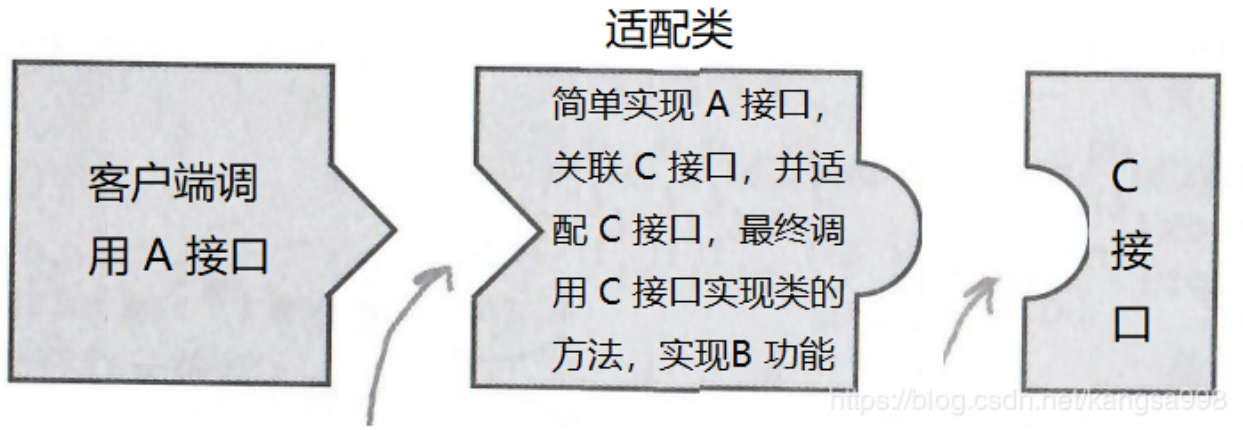
1 适配器模式的核心

- 客户端需要调用 A 接口，来实现 B 功能
- 而服务端并没有 A 接口现成的实现类
- 正好服务端有个 C 接口的实现类实现了 B 功能
- 这样开发人员就可以简单的实现 A 接口，并在实现类中引入 C 接口的实现类，做一些适配后，调用 C 接口实现类的相关方法，最终实现 B 功能
- 注：这里的接口也可以是类，也可以就是接口



2 适用场景

- 我们想使用一个已经存在的类，但是这个类的接口不符合要求，而我们又不希望直接修改这个类，所以我们定义另一个适配类，来适配这个类的接口
- 适配第三方包功能

3 示例代码

- 通过多继承实现适配器模式叫类适配器模式
- 通过组合(关联)实现适配器模式叫对象适配器模式
- 这两个没什么区别

3.1 客户端要调用的类(接口)

```
//外国教练调用 Player 接口来指导中国球员操作
//但是教练只会调用英文的指令，中国球员肯定听不懂
public abstract class ForeignPlayer {
    private String name;
    public ForeignPlayer(String name) {
        this.name = name;
    }
    public abstract void attack();
    public abstract void defense();
    //get/set 略
}
```

3.2 服务端只有中国球员听的懂的对象

```
public abstract class Player {
    private String 名字;
    public Player(String 名字) {
        this.名字 = 名字;
    }
    public abstract void 进攻();
    public abstract void 防守();
    //get/set 略
}
```

3.3 俱乐部通过一个适配类来适配

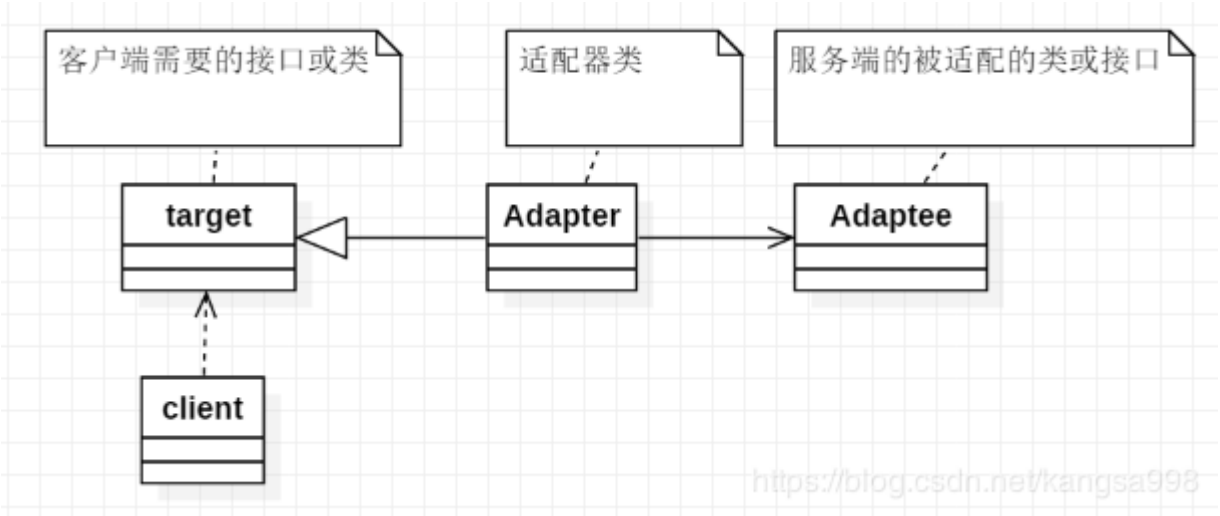
```
//通过继承 ForeignPlayer，来伪装成外国教练可以沟通的对象，并在这个对象中适配中国球员听的懂的对象
public class ForeignCenterAdapter extends ForeignPlayer {
    private Player player;
    public ForeignCenterAdapter(Player player) {
        super(player.get名字());
        this.player = player;
    }
    @Override
    public void attack() {
        player.进攻();//实际调用的是中国球员能听懂的
    }
    @Override
    public void defense() {
        player.防守();
    }
}
```

3.4 最终教练如何指挥中国球员

```
    @Test
    public void test() {
        //姚明上场
        Player player = new Center("姚明");
        //指定翻译和姚明合体，变成了一个会外国球员（英语的姚明）
        ForeignPlayer foreignPlayer = new ForeignCenterAdapter(player);
        //教练开始对这个会英语的姚明下命令了
        foreignPlayer.attack();
        //这样姚明和教练就可以顺利的沟通了，这样他们中的一方即节省了学语言的时间了
        //这样就可以把更多的时间用在训练上了
        foreignPlayer.defense();
    }
```

4 UML 类图

- 客户端只会调用（依赖）它需要的接口或类
- 服务端没有现成的实现类
- 新建个适配类，继承或实现客户端需要的类或接口，同时依赖于服务端的类或接口
- 这时因为多态特性，客户端时间调用的是适配类，但是对客户端是透明的，同样也实现了特定的功能
- 通过适配器类使得被适配的类或接口(Adaptee)可以和客户端接口协作(适配)！！



5 双向适配器

- 一个适配类可以实现两种适配嘛
- 注意：对于像 Java 一样的单继承语言，适配的就必须是接口而不能是类了！

```
public class TwoFacePalyerAdapter implements Player,ForeignPlayer {
    Player player;
    ForeignPlayer foreignPlayer;
    public TwoFacePalyerAdapter(Player player) {
        this.player = player;
    }
    public TwoFacePalyerAdapter(ForeignPlayer foreignPlayer) {
        this.foreignPlayer = foreignPlayer;
    }
    @Override
```

```
    public void 进攻() {
        foreignPlayer.attack();
    }
    @Override
    public void 防守() {
        foreignPlayer.defense();
    }
    @Override
    public void attack() {
        player.进攻();
    }
    @Override
    public void defense() {
        player.防守();
    }
}
@Test
public void test() {
    //中国教练指挥外国球员
    Player player = new TwoFacePalyerAdapter(new ForeignPlayerImpl("Jordan"));
    player.防守();
    player.进攻();
    //外国教练指挥中国球员
    ForeignPlayer fPlayer = new TwoFacePalyerAdapter(new PlayerImpl("姚明"));
    fPlayer.attack();
    fPlayer.defense();
}
```

6 我的困惑

- 设计模式一书中，说适配器模式改变了已有对象的接口
  - 并没有改变任一接口啊
  - 还是说我实现了一个接口，但是这个实现类最终调用的是另外一个接口？
- 设计模式一书中，说适配器模式中被适配的对象(被适配的对象是哪个啊？)不再兼容 Adaptee 接口，因此并不是所有 Adaptee 对象可以被使用的地方它都可以被使用。
  - 完全不明白这句话是什么意思，，
  - 这句话的前提是不是多重继承？

参考

大话设计模式 Head First 设计模式 设计模式 有趣的双向设计模式实例 github 源码地址