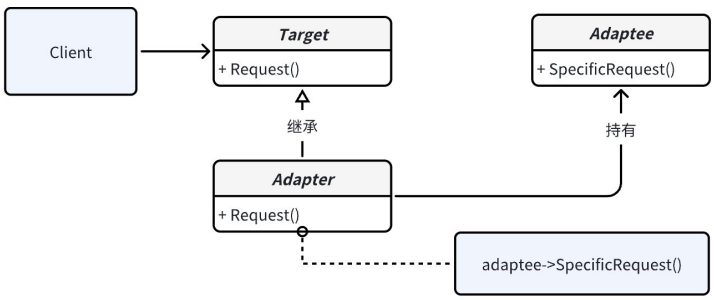
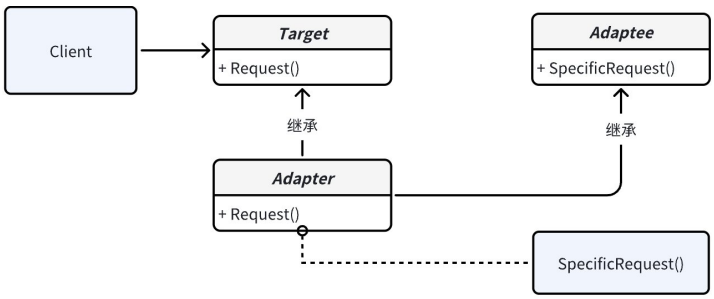
适配器模式

概念

- 适配器模式（Adapter Pattern）：是一种结构型设计模式，它的主要目的是将一个类的接口转换成客户端希望的另一个接口，使得原本由于接口不兼容而不能一起工作的类可以协同工作。适配器模式有两种实现方式：对象适配器和类适配器。
  - 对象适配器：通过继承的方式，将适配器类与被适配的对象关联起来。



  - 类适配器：通过组合的方式，将适配器类与被适配的类关联起来。



- 构成
  - 目标接口（Target）：定义客户端所需的接口。
  - 适配者（Adaptee）：定义一个已经存在的接口，这个接口需要适配。
  - 适配器（Adapter）：实现目标接口，并通过组合或继承的方式调用适配者的接口。
- 优点
  - 可以让任何两个没有关联的类一起运行。
  - 提高了类的复用性。
  - 增加了类的透明性和灵活性。

- 缺点
  - 过多地使用适配器会让系统变得复杂。
  - 由于引入了额外的适配器层，可能会影响性能。
- 在游戏开发中的应用
  - 数据库访问：不同的数据库（如MySQL、MongoDB）有不同的访问接口。适配器模式可以将这些不同的接口统一成一个标准接口，简化数据库操作。
  - 第三方库集成：游戏开发中经常需要使用第三方库（如物理引擎、图形库）。适配器模式可以将这些库的接口转换为游戏引擎所需的接口，简化集成过程。
  - 旧代码复用：在游戏开发中，可能需要复用旧项目中的代码。适配器模式可以将旧代码的接口适配到新项目中使用，避免重写代码。

实例

- 目标接口

```
1 public interface ITarget
2 {
3     void Request();
4 }
```

- 被适配者

```
1 public class Adaptee
2 {
3     public void SpecificRequest()
4     {
5         Console.WriteLine("Called SpecificRequest()");
6     }
7 }
```

对象适配器

- 适配器

```
1 public class Adapter : ITarget
2 {
3     // 维护
4     private readonly Adaptee _adaptee;
5
```

```
    public Adapter(Adaptee adaptee)
    {
        _adaptee = adaptee;
    }

    public void Request()
    {
        _adaptee.SpecificRequest();
    }
}
```

- 客户端

```
static void Main(string[] args)
{
    Adaptee adaptee = new Adaptee();
    ITarget target = new Adapter(adaptee);
    target.Request();
}
```

类适配器

- 适配器

```
public class Adapter : Adaptee, ITarget
{
    public void Request()
    {
        SpecificRequest();
    }
}
```

- 客户端

```
class Program
{
    static void Main(string[] args)
    {
        ITarget target = new Adapter();
        target.Request();
    }
```

```
8 }
```