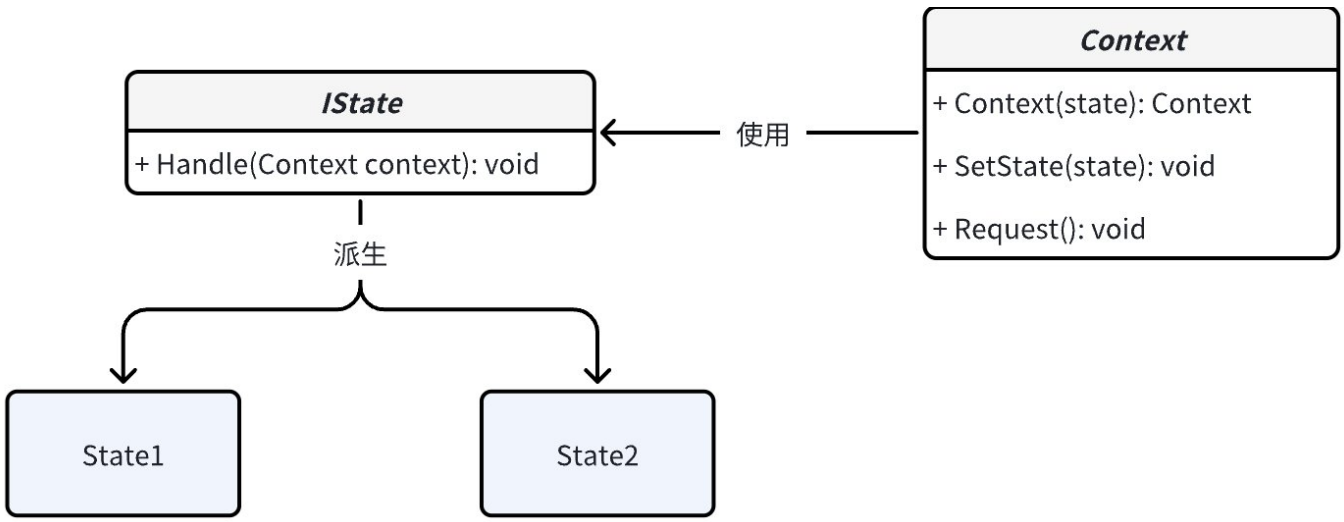
状态模式

概念

- 状态模式（State Pattern）：是一种行为设计模式，它允许对象在其内部状态改变时改变其行为。状态模式将与状态相关的行为抽取到独立的状态类中，使得原对象将工作委派给这些状态类的实例，而不是自行处理。这样可以避免大量的条件语句，提高代码的可维护性和扩展性。状态模式在游戏开发中有广泛的应用，比如处理角色状态、AI行为、游戏场景切换等。
- 组成：
  - State（状态）：定义一个接口，用以封装与上下文的一个特定状态相关的行为。
  - ConcreteState（具体状态）：实现状态接口的具体状态类。
  - Context（上下文）：维护一个状态实例，这个实例定义了当前的状态。



- 优点
  - 简化复杂状态转换逻辑：状态模式将与状态相关的行为分散到独立的状态类中，避免了在上下文类中使用大量的条件语句，从而简化了代码结构。
  - 提高可维护性和可扩展性：新的状态和状态转换可以通过添加新的状态类来实现，而不需要修改现有的代码，符合开闭原则。
  - 状态切换更清晰：状态模式使得状态切换更加明确和直观，每个状态类只负责处理其特定的行为和状态转换。
  - 封装状态：状态模式将状态和行为封装在独立的类中，使得状态的变化对其他类透明，减少了耦合。
- 缺点

- 类的数量增加：状态模式需要为每个具体状态创建一个类，可能会导致类的数量增加，增加系统的复杂性。
- 状态之间的依赖：如果状态之间存在复杂的依赖关系，可能会导致状态类之间的耦合增加。
- 状态切换的开销：状态切换可能会引入额外的开销，特别是在状态切换频繁的情况下。

实例

- 状态接口

```
public interface IOrderState
{
    void Handle(OrderContext context);
}
```

- 具体状态：继承状态接口并实现其中处理状态的方法

```
// 具体状态类：新订单
public class NewOrderState : IOrderState
{
    public void Handle(OrderContext context)
    {
        Console.WriteLine("订单状态：新订单");
        context.SetState(new ProcessingOrderState());
    }
}

// 具体状态类：处理中
public class ProcessingOrderState : IOrderState
{
    public void Handle(OrderContext context)
    {
        Console.WriteLine("订单状态：处理中");
        context.SetState(new ShippedOrderState());
    }
}

// 具体状态类：已发货
public class ShippedOrderState : IOrderState
{
    public void Handle(OrderContext context)
    {
        Console.WriteLine("订单状态：已发货");
        context.SetState(new DeliveredOrderState());
```

```
    }
}

// 具体状态类：已送达
public class DeliveredOrderState : IOrderState
{
    public void Handle(OrderContext context)
    {
        Console.WriteLine("订单状态：已送达");
    }
}
```

- 上下文类

```
// 上下文类
public class OrderContext
{
    private IOrderState _state;
    public OrderContext(IOrderState state)
    {
        _state = state;
    }

    public void SetState(IOrderState state)
    {
        _state = state;
    }

    public void Request()
    {
        _state.Handle(this);
    }
}
```

- 应用

```
class Program
{
    static void Main(string[] args)
    {
        // 初始化订单上下文，设置初始状态为新订单
        OrderContext order = new OrderContext(new NewOrderState());
        order.Request(); // 输出：订单状态：新订单
```

```
        order.Request(); // 输出：订单状态：处理中
        order.Request(); // 输出：订单状态：已发货
        order.Request(); // 输出：订单状态：已送达
    }
}
```