



第七章 状态模式

任课教师：武永亮

wuyongliang@edu2act.org

■ 上节回顾

- 装饰模式解决的问题是 “如何动态的给一个对象添加功能”
- 装饰模式的解决方案是利用子对象，委派

■ 课程内容

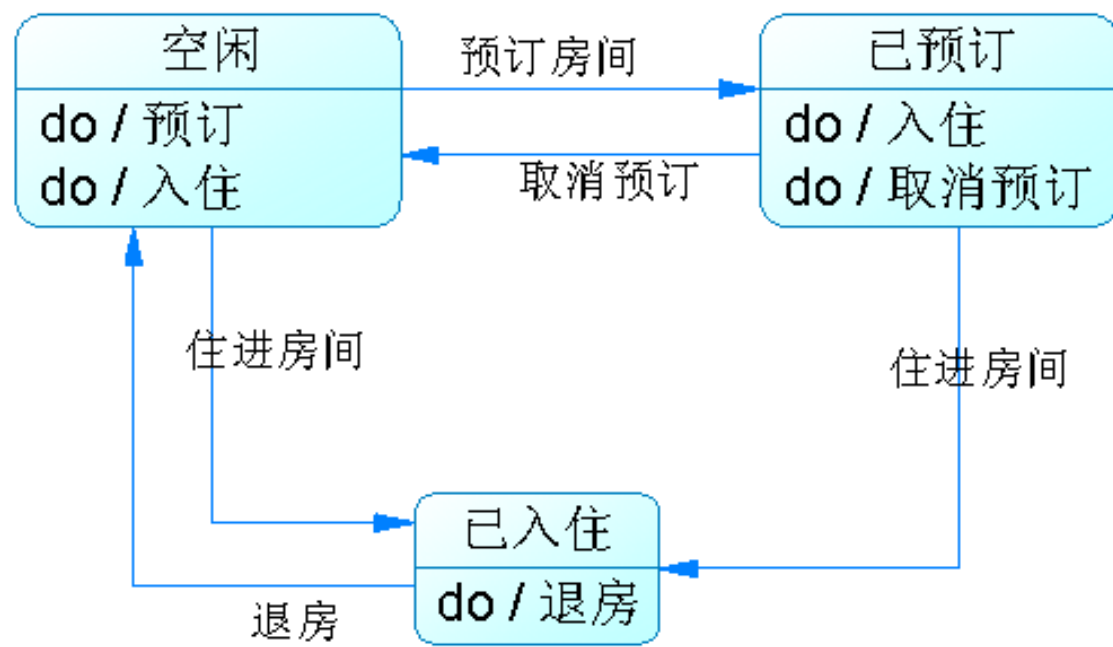
- 环境及问题
- 状态模式详解
- 状态模式实现
- 扩展练习

■ 课程内容

- 环境及问题
- 状态模式详解
- 状态模式实现
- 扩展练习

■ 环境

- 实现一个旅馆的住宿管理系统
- 房间的状态有三种：空闲，预定，入住



请利用15分钟时间对该系统进行设计编码

■ ■ 环境

```
.....  
if(state=="空闲")  
{  
    if(预订房间)  
    {  
        预订操作;  
        state="已预订";  
    }  
    else if(住进房间)  
    {  
        入住操作;  
        state="已入住";  
    }  
}  
else if(state=="已预订")  
{  
    if(住进房间)  
    {  
        入住操作;  
        state="已入住";  
    }  
    else if(取消预订)  
    {  
        取消操作;  
        state="空闲";  
    }  
}
```

❏ 问题

- **背景**：某对象发生**变化时**，其所能做的操作也随之变化。
- 应用程序的**可维护性和重用性差**。
- 代码的**逻辑较复杂**。

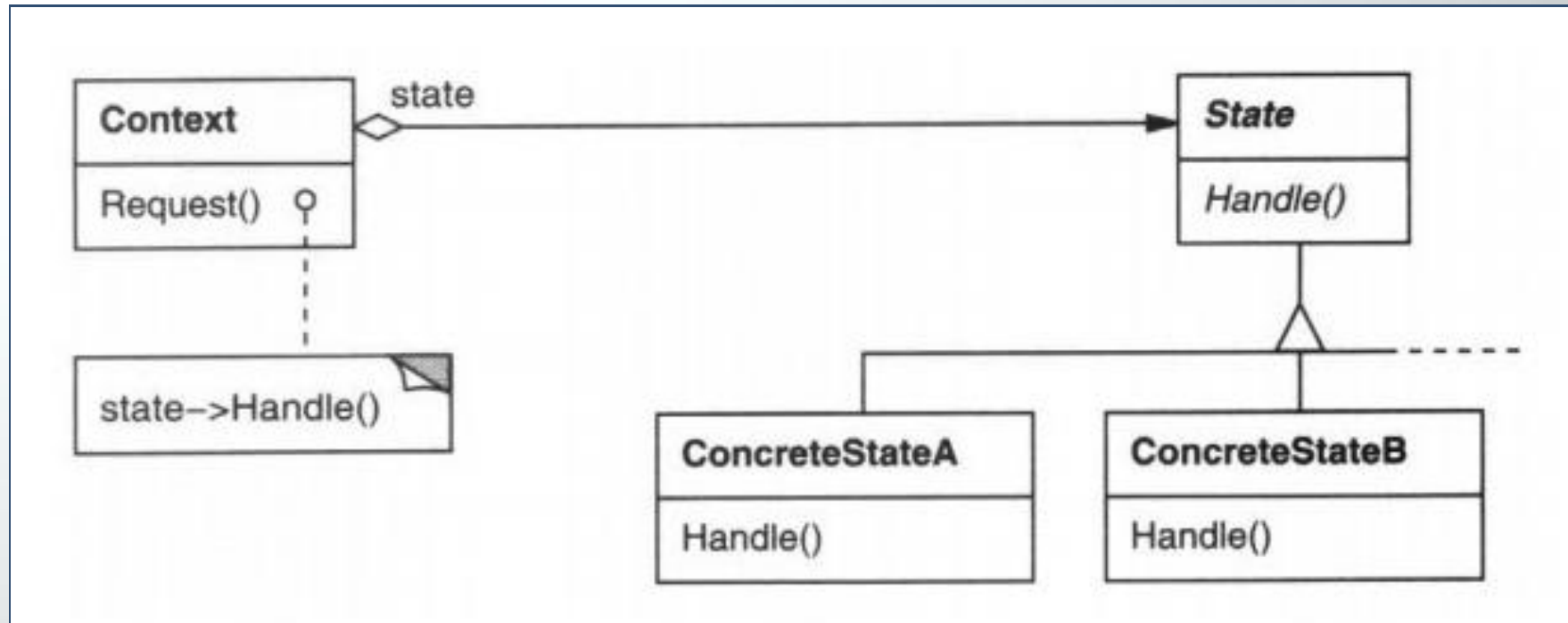
状态模式 (State)

■ 课程内容

- 环境及问题
- 状态模式详解
- 状态模式实现
- 扩展练习

■ 状态模式 (State模式)

- 允许对象在其内部状态改变的时候改变它的行为。
- 角色：
 - 环境类 (Context)：客户使用的对象类。维护一个State子类的实例，这个实例定义当前状态。
 - 抽象状态类 (State)：定义一个接口以封装与Context的一个特定状态相关的行为。
 - 具体状态类 (ConcreteState)：每一子类实现一个与Context的一个状态相关的行为。



■ ■ 状态模式有很多种实现形式，最常见的一种步骤如下：

■ ■ 定义状态类接口，实现当前系统的真实状态实现此接口

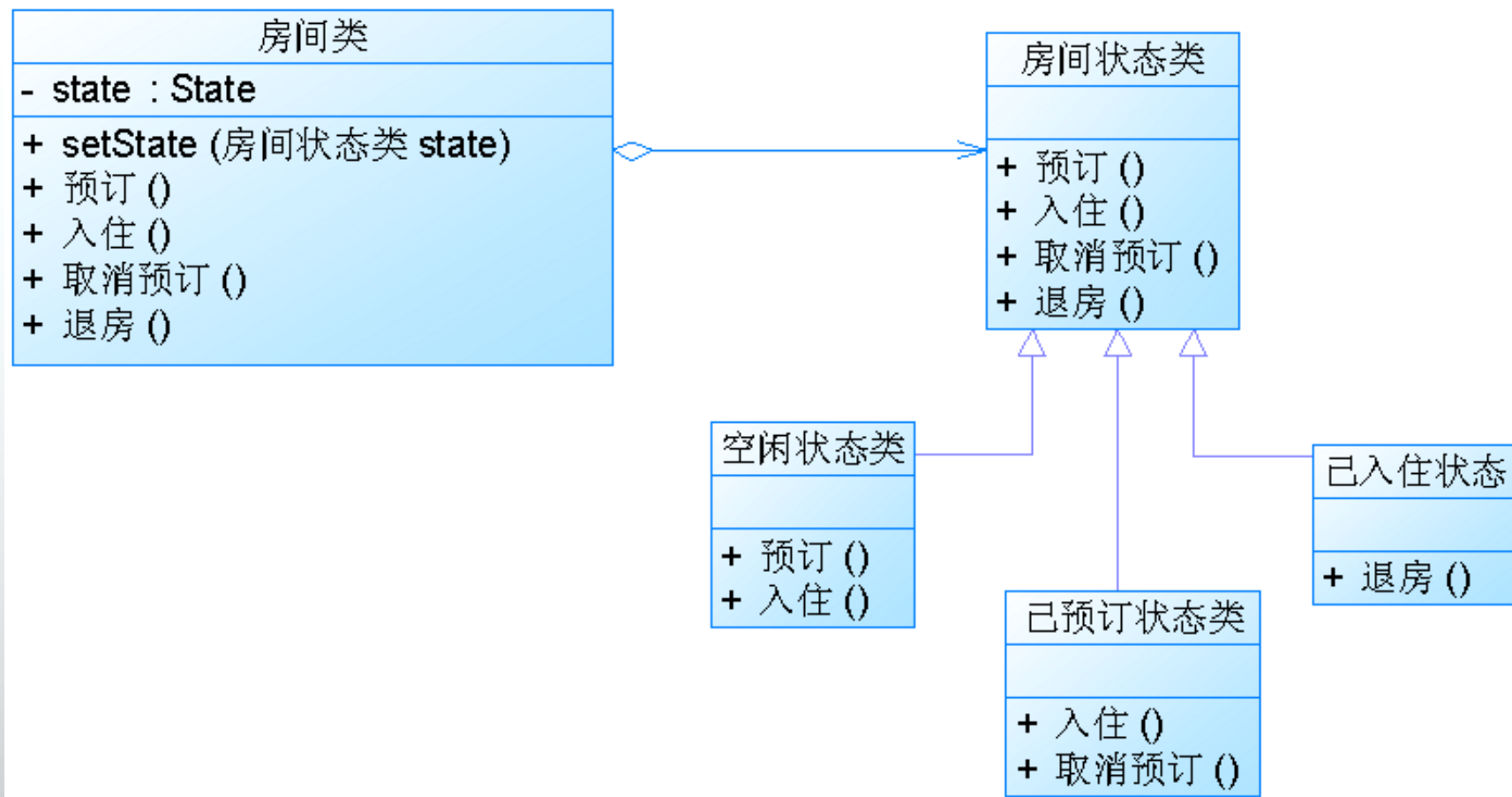
■ ■ 定义Context类，具有状态的类，其中包含状态类接口的对象

■ ■ 当Context类执行某个接口的方法时，去调用真实状态类的实现方法

■ ■ 当Context类修改状态时，修改Context类的真实状态对象

■ 课程内容

- 环境及问题
- 状态模式详解
- 状态模式实现
- 扩展练习



■ 状态模式重构之后的代码

```
abstract class State
{
    public abstract string getState();
    public abstract void book();
    public abstract void checkin();
    public abstract void unbook();
    public abstract void checkout();
}
```

```
class FreeState : State
{
    public override string getState()
    {
        return "当前为空闲状态";
    }

    public override void book()
    {
        System.Console.WriteLine("当前为空闲状态，进行预定操作");
    }

    public override void checkin()
    {
        System.Console.WriteLine("当前为空闲状态，进行入住操作");
    }

    public override void unbook()
    {
        System.Console.WriteLine("当前为空闲状态，无法进行取消预定操作");
    }

    public override void checkout()
    {
        System.Console.WriteLine("当前为空闲状态，无法进行退房操作");
    }
}
```

```
class BookState : State
{
    public override string getState()
    {
        return "当前为已预订状态";
    }

    public override void book()
    {
        System.Console.WriteLine("当前为已预订状态，无法进行预定操作");
    }

    public override void checkin()
    {
        System.Console.WriteLine("当前为已预订状态，进行入住操作");
    }

    public override void unbook()
    {
        System.Console.WriteLine("当前为已预订状态，进行取消预定操作");
    }

    public override void checkout()
    {
        System.Console.WriteLine("当前为已预订状态，无法进行退房操作");
    }
}
```

■ 状态模式重构之后的代码

```
class Room
{
    protected State s;

    public string getState()
    {
        return s.getState();
    }

    public void setState(State a)
    {
        s = a;
    }

    public void book()
    {
        s.book();
        s = new BookState();
    }

    public void checkin()
    {
        s.checkin();
        s = new CheckinState();
    }

    public void unbook()
    {
        s.unbook();
        s = new FreeState();
    }

    public void checkout()
    {
        s.checkout();
        s = new FreeState();
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Room r = new Room();
        State s = new FreeState();
        r.setState(s);

        System.Console.WriteLine("=====");
        System.Console.WriteLine(r.getState());

        System.Console.WriteLine("=====");
        r.checkin();
        System.Console.WriteLine(r.getState());

        System.Console.WriteLine("=====");
        r.checkout();
        System.Console.WriteLine(r.getState());

        System.Console.WriteLine("=====");
        r.checkout();
        System.Console.WriteLine(r.getState());

        System.Console.Read();
    }
}
```

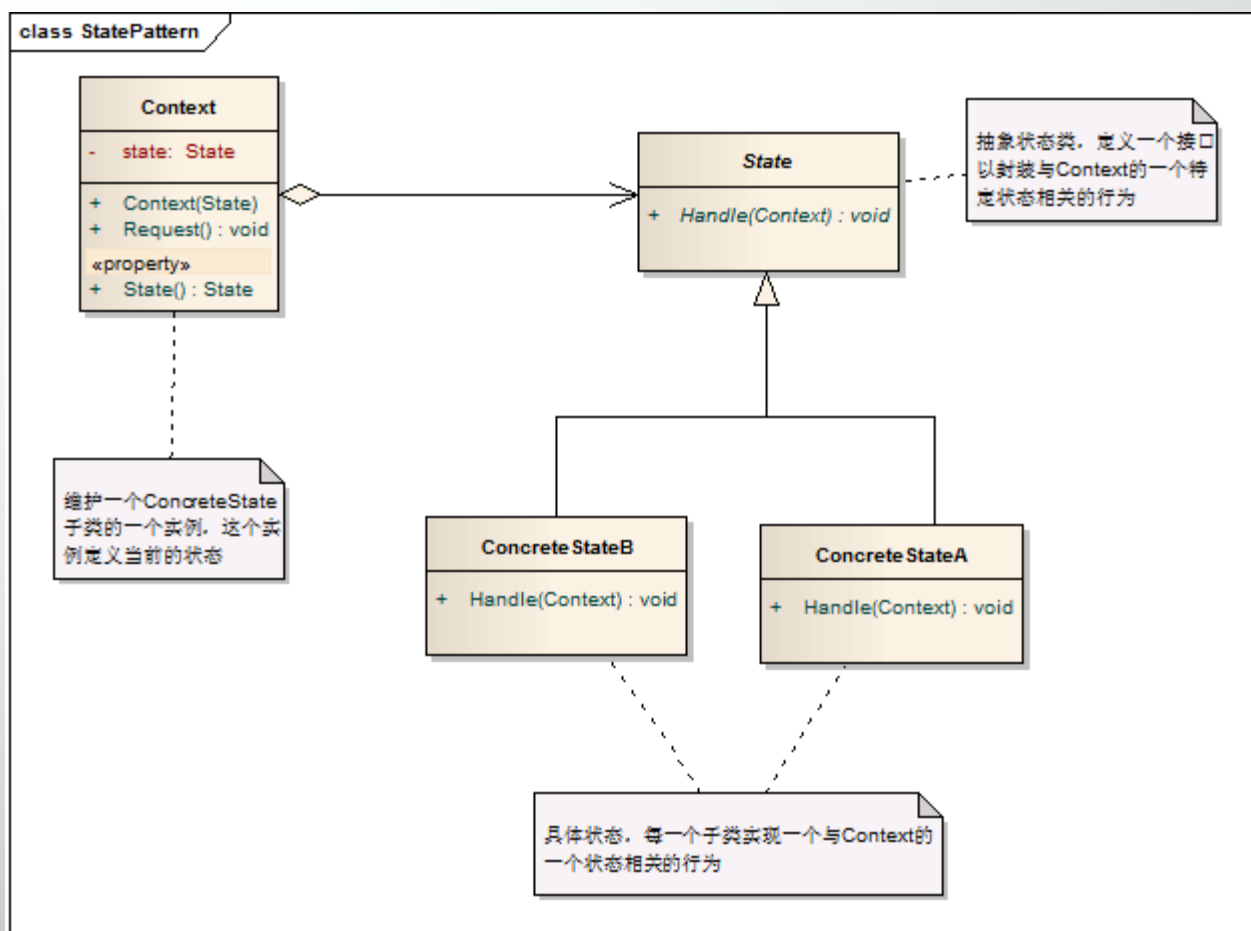
■ 课程内容

- 环境及问题
- 状态模式详解
- 状态模式实现
- 扩展练习

■ 扩展说明

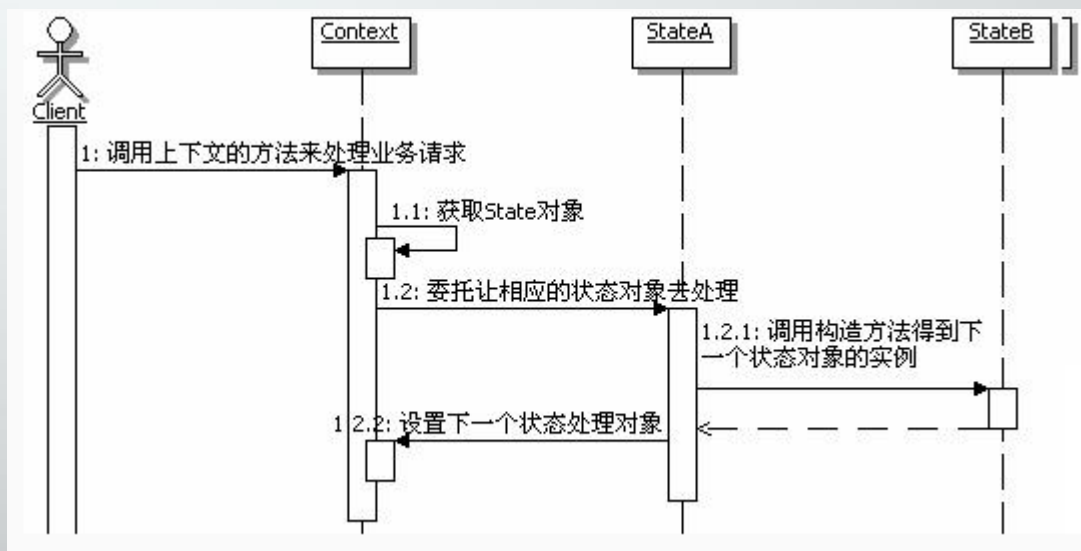
- 状态模式可封装状态转换规则
- 对于某个系统，可以方便的添加新的状态
- 允许状态转换逻辑于状态对象合成一体
- 状态模式必然会增加系统类和对象个数
- 结构和实现相对比较复杂
- 对“开闭原则”的支持并不是太好，增加状态需要修改复杂状态转换的代码

改进状态模式：在状态类内部维护状态的切换规则

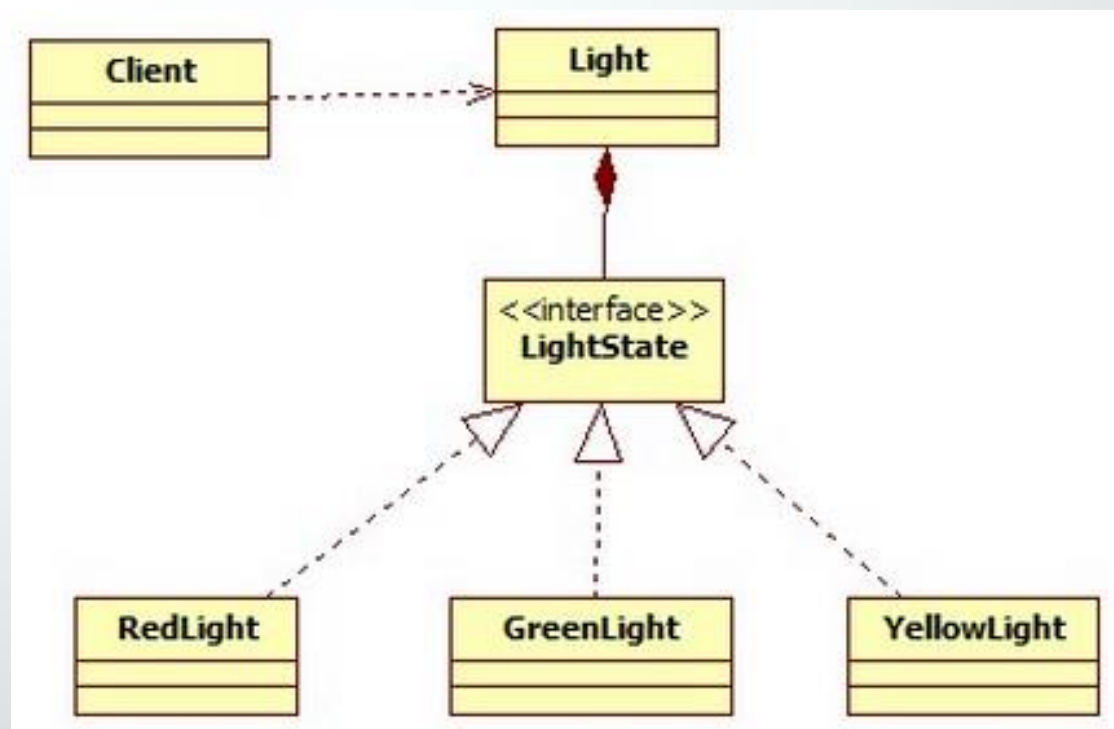


■ 状态切换规则：

- 1：一般情况下，如果状态转换的规则是一定的，一般不需要进行什么扩展规则，那么就适合在上下文中统一进行状态的维护。
- 2：如果状态的转换取决于前一个状态动态处理的结果，或者是依赖于外部数据，为了增强灵活性，这种情况下，一般是在状态处理类里面进行状态的维护。



红绿灯的切换



■ 小结

- 状态模式解决的问题是 “如何通过改变一个对象的状态，修改对象的行为”
- 状态模式的解决方案是利用包含状态类的子对象
- 如果状态之前具有前后关系，可在状态类内部进行状态的切换。

Thank You , 谢谢 !