状态模式，又称状态对象模式（Pattern of Objects for States），状态模式是对象的行为模式。

状态模式允许一个对象在其内部状态改变的时候改变其行为。这个对象看上去就像是改变了它的类一样。

为了避免过多的使用if/else判断，我们将每个状态对象封装到各自的类中，然后在动作发生前委托给当前对象。我们的步骤为：

1.我们创建一个state的接口，我们把每个状态共有的change方法放在这个接口中

2.我们让每一个状态都实现状态类，这些类将负责在对应的状态下进行相应行为

3.我们将动作委托到状态类中

当有新的状态产生时，我们只需要写一个新的状态类实现State接口，在对象类中提供一个变量即可，其他的都不需要动。对比原本的例子，我们明显的发现：

一：每个状态的行为局部化到它自己的类中

二、、使用多态代替了条件判断，将容易产生问题的if else结构去掉，使得代码的可维护性更强，不易出错，这样我们代码的扩展性更强，当要增加一些状态时，会非常的容易。

三、让每一个状态对修改关闭，对扩展开放

四、状态是可以被共享的，因为状态类一般是没有自己的内部状态的，所有它只是一个具有行为的对象，是可以被共享的。

六、状态的转换更加简单和安全，简单体现在状态的分割，每次转换的时候我们只需要关注一个固定的状态到其他状态的转换。安全体现在类型安全，我们设置状态时，必须是状态接口的实现类，而不是原本的一个整数，这可以杜绝数不正确的状态码。

状态模式允许对象在内部状态改变时改变它的行为，对象看起来好像改变了它的类

1.Context（上下文）是一个类，它可以拥有一些内部状态，相当于我们例子中的LuFei。

2.state接口定义了一个所有具体状态的共同接口；任何状态都实现这个相同的接口，这样一来状态之间可以互相替换。

3.ConcreteState（具体状态）处理来自Context的请求，每一个ConcreteState都提供了它自己对于请求的实现。所以，当Context改变状态时行为也跟着改变。

4.state.handle() :不管什么时候，只要有人调用Context的request方法，它就会被委托到状态来处理。

状态模式适用于某一个对象的行为取决于该对象的状态，并且该对象的状态会在运行时转换，又或者有很多的if else判断，而这些判断只是因为状态不同而不断的切换行为。

状态模式和策略模式的区别：

状态模式：将一群行为封装到状态对象中，context的行为随时可委托到那些状态对象中的一个。随着时间的流逝，当前状态在状态对象集合中游走改变，以反映出context内部的状态，因此，context的行为也会跟着改变。但是context的客户对于状态对象了解不多，甚至根本是浑然不觉。

策略模式：客户通常主动指定Context所要组合的策略对象是哪一个。现在，固然策略模式让我们具有弹性，能够在运行时改变策略，但对于某个context对象来说，通常都只有一个最适当的策略对象。一般来说，我们把策略模式想成是除了继承之外的一种弹性替代方案。如果你使用继承定义了一个类的行为，你将被这个行为困住，甚至要修改它都很难。有了策略模式，你可以通过组合不同的对象来改变行为。

下面我们总结一下状态模式的要点：

1.状态模式允许一个对象基于内部状态而拥有不同的行为。

2.状态模式用类代表状态

3.Context会将行为委托给当前状态对象

4.通过将每个状态封装进一个类，我们把以后需要做的任何改变局部化了

5.状态模式和策略模式有相同的类图，但是它们的意图不同

6.策略模式通常会用行为或算法来配置Context类

7.使用状态模式通常会导致设计中类的数目大量增加。

8.状态类可以被多个Context实例共享。

状态模式在项目当中也算是较经常会碰到的一个设计模式，但是通常情况下，我们还是在看到多重if/else的情况下，又或者你十分确定要做的项目会朝着状态模式发展，对项目进行重构时使用。一般情况下，不建议在项目的初期使用。