* 设计模式 – 状态模式（state）：

允许一个对象在其内部状态改变的时候改变行为。这个对象看上去象是改变了它的类一 样。

状态模式把所研究的对象的行为包装在不同的状态对象里，每一个状态对象都属于一个抽象状态类的一个子类。

状态模式的意图是让一个对象在其内部状态改变时候，其行为也随之改变。

**状态模式角色：**

1）**环境(Context)角色**：也叫上下文，定义客户端所感兴趣的接口，并且保留一个具体状态类的实例。这个具体状态类的实例给出此环境对象的现有状态。

2）**抽象状态(State)角色**：定义一个接口，用以封装环境（Context）对象的一个特定的状态所对应的行为。

3）**具体状态(ConcreteState)角色**：每一个具体状态类都实现了环境（Context）的一个状态所对应的行为。

* 设计模式 – 状态模式优劣：

**状态模式优点**：

状态模式将与特定状态相关的行为局部化，并且将不同状态的行为分割开来。

所有状态相关的代码都存在于某个ConcereteState中，所以通过定义新的子类很容易地增加新的状态和转换。

状态模式通过把各种状态转移逻辑分不到State的子类之间，来减少相互间的依赖。

**状态模式缺点：**

导致较多的ConcreteState子类。

* 设计模式 – 状态模式**适用场景：**

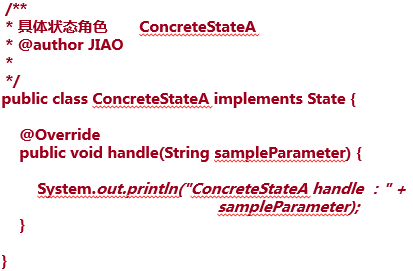
当一个对象的行为取决于它的状态，并且它必须在运行时刻根据状态改变它的行为时，就可以考虑使用状态模式来。

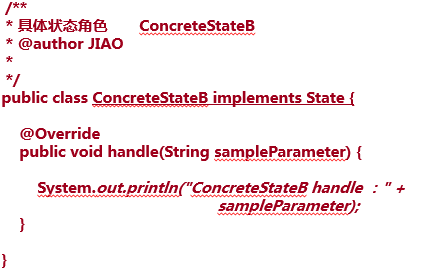
一个操作中含有庞大的分支结构，并且这些分支决定于对象的状态。

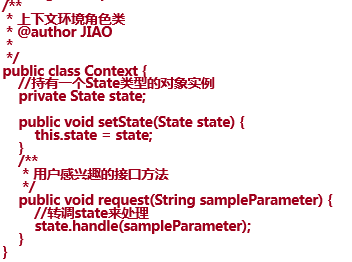
示例：电灯有两个状态，开（亮）与关（不亮），可用状态模式来实现对电灯的控制。

* 设计模式 – 状态模式Demo：











从上面可以看出：

环境类Context的行为request()是委派给某一个具体状态类的。通过使用多态性原则，可以动态改变环境类Context的属性State的内容，使其从指向一个具体状态类变换到指向另一个具体状态类，从而使环境类的行为request()由不同的具体状态类来执行。