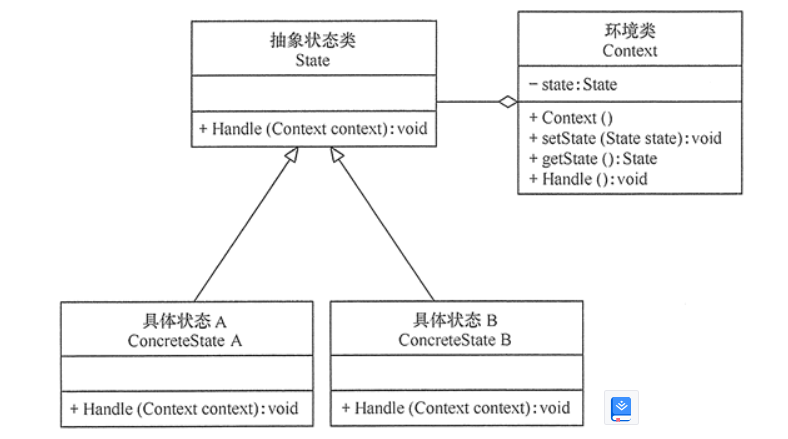
状态模式  
  
 在软件开发过程中，应用程序中的部分对象可能会根据不同的情况做出不同的行为，我们把这种对象称为有状态的对象，而把影响对象行为的一个或多个动态变化的属性称为状态。  
当有状态的对象与外部事件产生互动时，其内部状态就会发生改变，从而使其行为也发生改变。  
  
 对这种有状态的对象编程，传统的解决方案是：将这些所有可能发生的情况全都考虑到，然后使用 if-else 或 switch-case 语句来做状态判断，再进行不同情况的处理。  
但是显然这种做法对复杂的状态判断存在天然弊端，条件判断语句会过于臃肿，可读性差，且不具备扩展性，维护难度也大。且增加新的状态时要添加新的 if-else 语句，这违背了“开闭原则”，不利于程序的扩展。  
  
 状态模式的解决思想是：当控制一个对象状态转换的条件表达式过于复杂时，把相关“判断逻辑”提取出来，用各个不同的类进行表示，系统处于哪种情况，直接使用相应的状态类对象进行处理，这样能把原来复杂的逻辑判断简单化，消除了 if-else、switch-case 等冗余语句，代码更有层次性，并且具备良好的扩展力。  
============================================================  
状态模式包含以下主要角色。  
 **环境类（Context）角色：也称为上下文，它定义了客户端需要的接口，内部维护一个当前状态，并负责具体状态的切换。** 抽象状态（State）角色：定义一个接口，用以封装环境对象中的特定状态所对应的行为，可以有一个或多个行为。  
 具体状态（Concrete State）角色：实现抽象状态所对应的行为，并且在需要的情况下进行状态切换。  
   
   
 =========================================   
  
  
在状态模式（State Pattern）中，类的行为是基于它的状态改变的。这种类型的设计模式属于\*\*\*\*行为型模式。  
  
  
介绍  
  
意图：允许对象在内部状态发生改变时改变它的行为，对象看起来好像修改了它的类。  
主要解决：对象的行为依赖于它的状态（属性），并且可以根据它的状态改变而改变它的相关行为。  
  
何时使用：代码中包含大量与对象状态有关的条件语句。  
  
如何解决：将各种具体的状态类抽象出来。  
  
关键代码：通常命令模式的接口中只有一个方法。而状态模式的接口中有一个或者多个方法。而且，状态模式的实现类的方法，一般返回值，或者是改变实例变量的值。也就是说，状态模式一般和对象的状态有关。实现类的方法有不同的功能，覆盖接口中的方法。状态模式和命令模式一样，也可以用于消除 if...else 等条件选择语句。  
  
应用实例：  
  
  
优点：  
 1、封装了转换规则。  
 2、枚举可能的状态，在枚举状态之前需要确定状态种类。  
 3、将所有与某个状态有关的行为放到一个类中，并且可以方便地增加新的状态，只需要改变对象状态即可改变对象的行为。  
 4、允许状态转换逻辑与状态对象合成一体，而不是某一个巨大的条件语句块。  
 5、可以让多个环境对象共享一个状态对象，从而减少系统中对象的个数。  
  
缺点：  
 1、状态模式的使用必然会增加系统类和对象的个数。  
 2、状态模式的结构与实现都较为复杂，如果使用不当将导致程序结构和代码的混乱。  
 3、状态模式对"开闭原则"的支持并不太好，对于可以切换状态的状态模式，增加新的状态类需要修改那些负责状态转换的源代码，否则无法切换到新增状态，  
 而且修改某个状态类的行为也需修改对应类的源代码。  
  
使用场景：  
 1、行为随状态改变而改变的场景。  
 2、条件、分支语句的代替者。  
  
注意事项：在行为受状态约束的时候使用状态模式，而且状态不超过 5 个。