有限状态机（Finite State Machine或者Finite State Automata)是软件领域中一种重要的工具，很多东西的模型实际上就是有限状态机。

FSM的实现方式：  
1） switch/case或者if/else  
这无意是最直观的方式，使用一堆条件判断，会编程的人都可以做到，对简单小巧的状态机来说最合适，但是毫无疑问，这样的方式比较原始，对庞大的状态机难以维护。

1. 状态表  
   维护一个二维状态表，横坐标表示当前状态，纵坐标表示输入，表中一个元素存储下一个状态和对应的操作。这一招易于维护，但是运行时间和存储空间的代价较大。
2. 使用State Pattern  
   使用State Pattern使得代码的维护比switch/case方式稍好，性能上也不会有很多的影响，但是也不是100％完美。不过Robert C. Martin做了两个自动产生FSM代码的工具，for **[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/xgbing/article/details/_blank)**和for C++各一个，在http://www.objectmentor.com/resources/index上有免费下载，这个工具的输入是纯文本的状态 机描述，自动产生符合State Pattern的代码，这样developer的工作只需要维护状态机的文本描述，每必要冒引入bug的风险去维护code。
3. 使用宏定义描述状态机  
   一般来说，C++编程中应该避免使用#define，但是这主要是因为如果用宏来定义函数的话，很容易产生这样那样的问题，但是巧妙的使用,还是能够产生奇妙的效果。MFC就是使用宏定义来实现大的**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/xgbing/article/details/_blank)**的。  
   在实现FSM的时候，可以把一些繁琐无比的if/else还有花括号的组合放在宏中，这样，在代码中可以3）中状态机描述文本一样写，通过编译器的预编译处理产生1）一样的效果，我见过产生C代码的宏，如果要产生C++代码，己软MFC可以，那么理论上也是可行的。

有限状态机是一种用来进行对象行为建模的工具，主要作用是描述对象在它的生命周期内所经历的状态序列，以及如何响应来自外界的各种事件。在面向对象的软件系统当中，一个对象无论多么简单或复杂，都会经历一个从开始创建到最终消亡的过程，这通常被称为一个对象的生命周期。 一般来说，对象在其生命周期内不可能完全孤立，它必须要通过发送消息来影响其他的对象，或者是通过接受消息来改变自身。大多数情况下，这些消息都只不过是一些简单的、同步的方法调用。例如，在银行客户管理系统中，客户类（Customer）的实例在需要的时候，可能会调用帐户（Account）类中定义的getBalance()方法。在这种简单的情况下，类Customer并不需要一个有限状态机来描述自己的行为，主要原因在于它当前的行为并不依赖于过去的某个状态。

遗憾的是并不是所有情况都会如此简单，事实上许多实用的软件系统都必须维护一两个非常关键的对象，它们通常具有非常复杂的状态转换关系，而且需要对来自外部的各种异步事件进行响应。例如，在VoIP电话系统中，电话类（Telephone）的实例必须能够响应来自对方的随机呼叫，来自用户的按键事件，以及来自网络的信令等。在处理这些消息时，类Telephone所要采取的行为完全依赖于它当前所处的状态，因而此时使用状态机就将是一个不错的选择。

在描述有限状态机时，状态、事件、转换和动作是经常会碰到的几个基本概念。

**状态（State）**指的是对象在其生命周期中的一种状况，处于某个特定状态中的对象必然会满足某些条件、执行某些动作或者是等待某些事件。

**事件（Event）**指的是在时间和空间上占有一定位置，并且对状态机来讲是有意义的那些事情。事件通常会引起状态的变迁，促使状态机从一种状态切换到另一种状态。

**转换（Transition）**指的是两个状态之间的一种关系，表明对象将在第一个状态中执行一定的动作，并将在某个事件发生同时某个特定条件满足时进入第二个状态。

**动作（Action）**指的是状态机中可以执行的那些原子操作，所谓原子操作指的是它们在运行的过程中不能被其他消息所中断，必须一直执行下去。