|19.1 state模式

【用类表示状态】

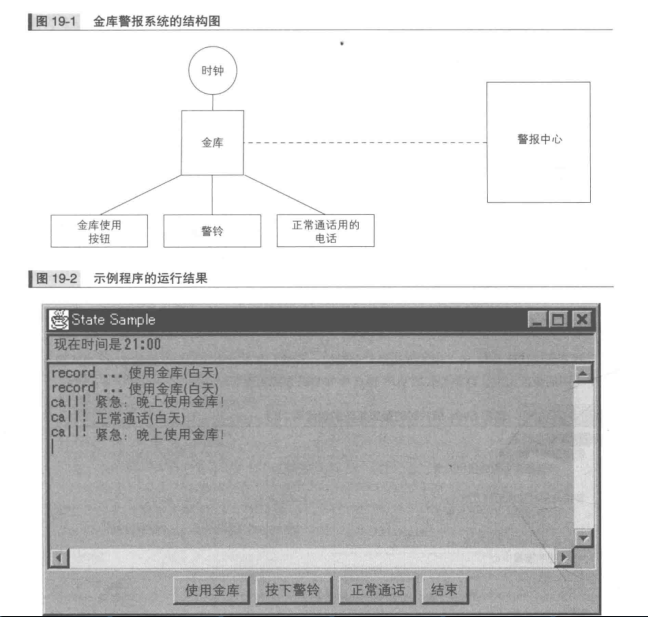
在面向对象编程中，是用类表示对象的。也就是说.程序的设计者需要考虑用类来表示什么东西。类对应的东西可能存在于真实世界中，也可能不存在于真实世界中。对于后者，可能有人看到代码后会感到吃惊：这些东西居然也可以是类啊。

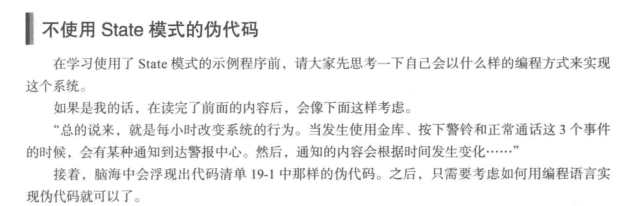
在本章中，我们将要学习State模式。

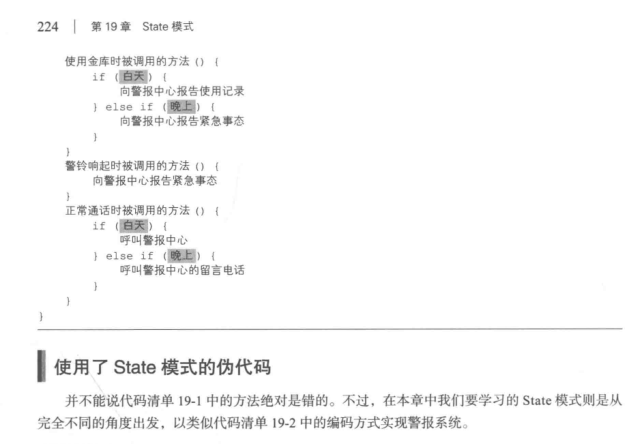
在State模式中，我们用类来表示状态。State的意思就是“状态”。在现实世界中，我们会考虑各种东西的“状态”，但是几乎不会将状态当作“东西”看待。因此，可能大家很难理解“用类来表示状态”的意思。

在本章中，我们将要学习用类来表示状态的方法。以类来表示状态后，我们就能通过切换类来方便地改变对象的状态。当需要增加新的状态时，如何修改代码这个问题也会很明确。

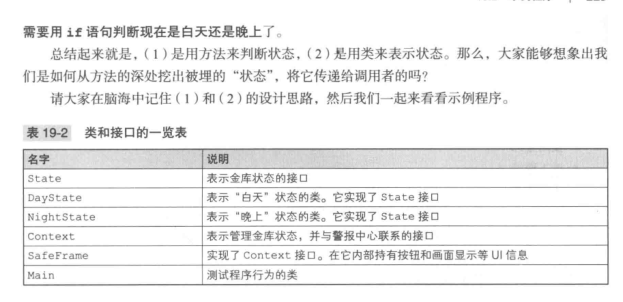


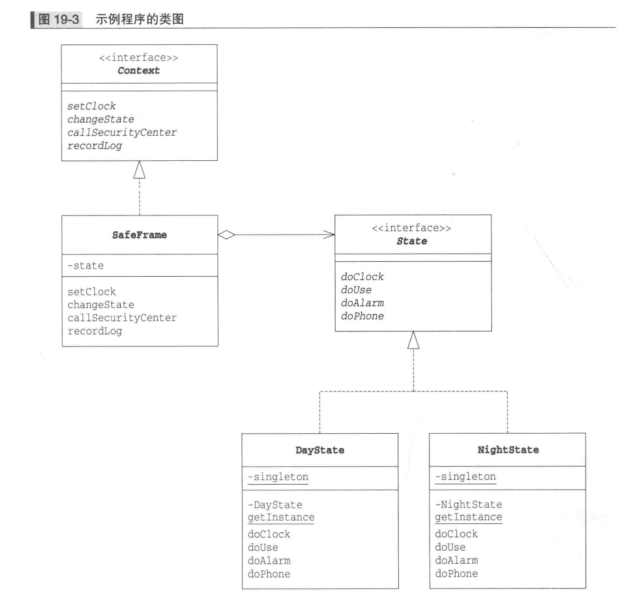












|19.3 state模式中的登场角色

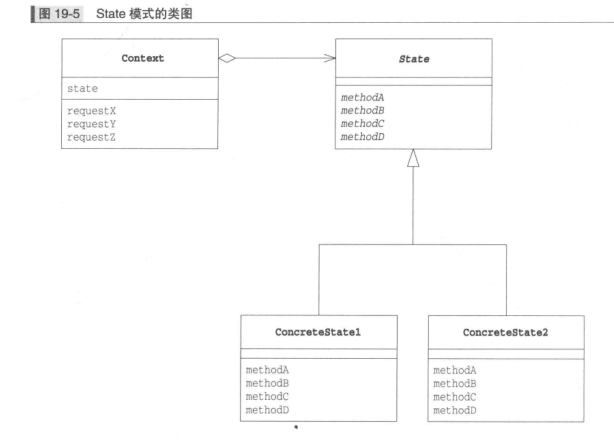
在State模式中有以下登场角色。

◆State（状态）State角色表示状态，定义了根据不同状态进行不同处理的接口（APl）。该接口（APl）是那些处理内容依赖于状态的方法的集合。在示例程序中，由state接口扮演此角色。

◆ConcreteState（具体状态）ConcreteState角色表示各个具体的状态，它实现了State接口。在示例程序中，由Daystate类和NightState类扮演此角色。

·Context（状况、前后关系、上下文）Context角色持有表示当前状态的ConcreteState角色。此外，它还定义了供外部调用者使用State模式的接口（API）。在示例程序中，由Context接口和SafeFrame类扮演此角色。

这里稍微做一下补充说明。在示例程序中，Context角色的作用被Context接口和SafeFrame类分担了。具体而言，Context接口定义了供外部调用者使用State模式的接口（API），而SafeFrame类则持有表示当前状态的ConcreteState角色。我们会在习题19-1中讨论一下为什么没有将Context实现为类。



|19.4拓展思路的要点

|分而治之

在编程时，我们经常会使用分而治之的方针。它非常适用于大规模的复杂处理。当遇到庞大且复杂的问题，不能用一般的方法解决时，我们会先将该问题分解为多个小问题。如果还是不能解决这些小问题，我们会将它们继续划分为更小的问题，直至可以解决它们为止。分而治之，简单而言就是将一个复杂的大问题分解为多个小问题然后逐个解决。

在State模式中，我们用类来表示状态，并为每一种具体的状态都定义一个相应的类。这样，问题就被分解了。开发人员可以在编写一个ConcreteState角色的代码的同时，在头脑中（一定程度上）考虑其他的类。在本章的金库警报系统的示例程序中，只有“白天”和“晚上”两个状态，可能大家对此感受不深，但是当状态非常多的时候，State模式的优势就会非常明显了。

请大家再回忆一下代码清单19-1中的伪代码（l）和（2）。在不使用State模式时，我们需要使用条件分支语句判断当前的状态，然后进行相应的处理。状态越多，条件分支就会越多。而且，我们必须在所有的事件处理方法中都编写这些条件分支语句。

State模式用类表示系统的“状态”，并以此将复杂的程序分解开来。

|依赖于状态的处理

我们来思考一下SafeFrame类的setclock方法（代码清单19-7）和State接口的doclock方法（代码清单19-3）之间的关系。

Main 类会调用SafeFrame类的 setclock方法，告诉 setclock方法“请设置时间”。在setclock方法中，会像下面这样将处理委托给state类。

state.doclock（this，hour）；也就是说，我们将设置时间的处理看作是“依赖于状态的处理”。

当然，不只是doClock方法。在State接口中声明的所有方法都是“依赖于状态的处理”，都是“状态不同处理也不同”。这虽然看似理所当然，不过却需要我们特别注意。

在State模式中，我们应该如何编程，以实现“依赖于状态的处理”呢？总结起来有如下两点。

·定义接口，声明抽象方法

·定义多个类，实现具体方法

这就是State模式中的“依赖于状态的处理”的实现方法。

这里故意将上面两点说得很笼统，但是，如果大家在读完这两点之后会点头表示赞同，那就意味着大家完全理解了State模式以及接口与类之间的关系。

应当是谁来管理状态迁移

用类来表示状态，将依赖于状态的处理分散在每个ConcreteState角色中，这是一种非常好的解决办法。

不过，在使用State模式时需要注意应当是谁来管理状态迁移。

在示例程序中，扮演Context角色的SafeFrame 类实现了实际进行状态迁移的changeState方法。但是，实际调用该方法的却是扮演ConcreteState角色的DayState类和NightState类。

也就是说，在示例程序中，我们将“状态迁移”看作是“依赖于状态的处理”。这种处理方式既有优点也有缺点。

优点是这种处理方式将“什么时候从一个状态迁移到其他状态”的信息集中在了一个类中。也就是说，当我们想知道“什么时候会从Daystate类变化为其他状态”时，只需要阅读Daystate类的代码就可以了。

缺点是“每个ConcreteState角色都需要知道其他ConcreteState角色”。例如，Daystate类的doclock方法就使用了NightState类。这样，如果以后发生需求变更，需要删除Nightstate类时，就必须要相应地修改Daystate类的代码。将状态迁移交给ConcreteState角色后，每个ConcreteState角色都需要或多或少地知道其他ConcreteState角色。也就是说，将状态迁移交给ConcreteState角色后，各个类之间的依赖关系就会加强。

我们也可以不使用示例程序中的做法，而是将所有的状态迁移交给扮演Context角色的SafeFrame类来负责。有时，使用这种解决方法可以提高ConcreteState角色的独立性，程序的整体结构也会更加清晰。不过这样做的话，Context角色就必须要知道“所有的ConcreteState角色”。

在这种情况下，我们可以使用Mediator模式（第16章）。

当然，还可以不用State模式，而是用状态迁移表来设计程序。所谓状态迁移表是可以根据

“输入和内部状态”得到“输出和下一个状态”的一览表（这超出了本书的范围，我们暂且不深入学习该方法）。当状态迁移遵循一定的规则时，使用状态迁移表非常有效。

此外，当状态数过多时，可以用程序来生成代码而不是手写代码。

|不会自相矛盾

如果不使用State模式，我们需要使用多个变量的值的集合来表示系统的状态。这时，必须十分小心，注意不要让变量的值之间互相矛盾。

而在State模式中，是用类来表示状态的。这样，我们就只需要一个表示系统状态的变量即可。

在示例程序中，safeFrame类的state字段就是这个变量，它决定了系统的状态。因此，不会存在自相矛盾的状态。易于增加新的状态

在State模式中增加新的状态是非常简单的。以示例程序来说，编写一个xxXstate类，让它实现State接口，然后实现一些所需的方法就可以了。当然，在修改状态迁移部分的代码时，还是需要仔细一点的。因为状态迁移的部分正是与其他ConcreteState角色相关联的部分。

但是，在State模式中增加其他“依赖于状态的处理”是很困难的。这是因为我们需要在State接口中增加新的方法，并在所有的ConcreteState角色中都实现这个方法。虽说很困难，但是好在我们绝对不会忘记实现这个方法。假设我们现在在State接口中增加了一个doyYY方法，而忘记了在Daystate类和Nightstate类中实现这个方法，那么编译器在编译代码时就会报错，告诉我们存在还没有实现的方法。

如果不使用State模式，那么增加新的状态时会怎样呢？这里，如果不使用State模式，就必须用if语句判断状态。这样就很难在编译代码时检测出“忘记实现方法”这种错误了（在运行时检测出问题并不难。我们只要事先在每个方法内部都加上一段“当检测到没有考虑到的状态时就报错”的代码即可）。

实例的多面性

请注意SafeFrame类中的以下两条语句（代码清单19-7）。

·SafeFrame类的构造函数中的

buttonUse.addActionListener（this）；

·actionPerformed方法中的

state.douse（this）；这两条语句中都有this。那么这个this到底是什么呢？当然，它们都是SafeFrame类的实例。由于在示例程序中只生成了一个safeFrame的实例，因此这两个this其实是同一个对象。

不过，在addActionListener方法中和douse方法中，对this的使用方式是不一样的。

向addActionListener方法传递this时，该实例会被当作“实现了ActionListener接口的类的实例”来使用。这是因为addActionListener方法的参数类型是ActionListener类型。在addActionListener方法中会用到的方法也都是在ActionListener接口中定义了的方法。至于这个参数是否是SafeFrame类的实例并不重要。

向doUse方法传递this时，该实例会被当作“实现了Context接口的类的实例”来使用。这是因为doUse方法的参数类型是Context类型。在doUse方法中会用到的方法也都是在Context接口中定义了的方法（大家只要再回顾一下Daystate类和NightState类的doUse方法就会明白了）。

请大家一定要透彻理解此处的实例的多面性。

|19.5相关的设计模式

◆Singleton模式（第5章）Singleton模式常常会出现在ConcreteState角色中。在示例程序中，我们就使用了Singleton模式。这是因为在表示状态的类中并没有定义任何实例字段（即表示实例的状态的字段）。

·Flyweight模式（第20章）在表示状态的类中并没有定义任何实例字段。因此，有时我们可以使用Flyweight模式在多个Context角色之间共享ConcreteState角色。