第22章

# 状态（State）模式

对象状态是指对象属性的当前值的组合。在调用对象的 set 方法，或给对象的成员变量赋值时，都是在改变对象的状态。通常在执行对象的方法时，其自身状态也会改变。

我们通常使用状态一词来代表对象中独立的、可改变的属性。例如，我们可以说机器的状态是打开还是关闭。此时，对象状态中的可变部分可能是它所拥有的行为中最突出的部分。因此，与对象状态相关的逻辑可能在许多类的方法中蔓延。多次重复出现相似或者相同的逻辑，会带来维护上的沉重负担。

有一种办法可以获知与状态相关的逻辑分布情况：引入一组新的类，每个类代表不同的状态，然后将与状态相关的行为分别写到每个类中。

**状态模式的意图是将表示对象状态的逻辑分散到代表状态的不同类中。**

## 对状态进行建模

对那些更为关注状态的对象进行建模时，需要一个依赖于状态的变量，来跟踪对象的行为。这个变量可能会出现在复杂的、层叠的 if 语句中，该语句可以对对象收到的事件进行处理。使用这种方式对状态建模，会让代码变得很复杂，尤其是当调整状态模型时，不得不调整很多方法的代码。状态模式利用分散的操作提供了一种整洁简单的操作方式。它将状态建模为对象，

并把状态相关的逻辑封装到独立的类中。为了了解状态模式的工作方式，我们在系统建模时先不使用状态模式。在下一小节中，我们对这些代码进行重构，以此验证状态模式是否能改进我们的设计。

假设 Oozinoz 公司要对传送带的入口门进行建模。传送带是一个大型的智能设备，它上面有一个入口门可以接收原料，并且通过原料上的条形码对原料进行存储。入口门只有一个操作按钮。如果门关闭，按键后门会自动打开。如果在门尚未完全打开前再次按键，门将再次关闭。如果门一直开着，2 秒后门会自动超时关闭。可以在门打开的时候通过再次按键来阻止门的自动关闭。图 22.1 展示了传送带入口门的状态和转换效果。

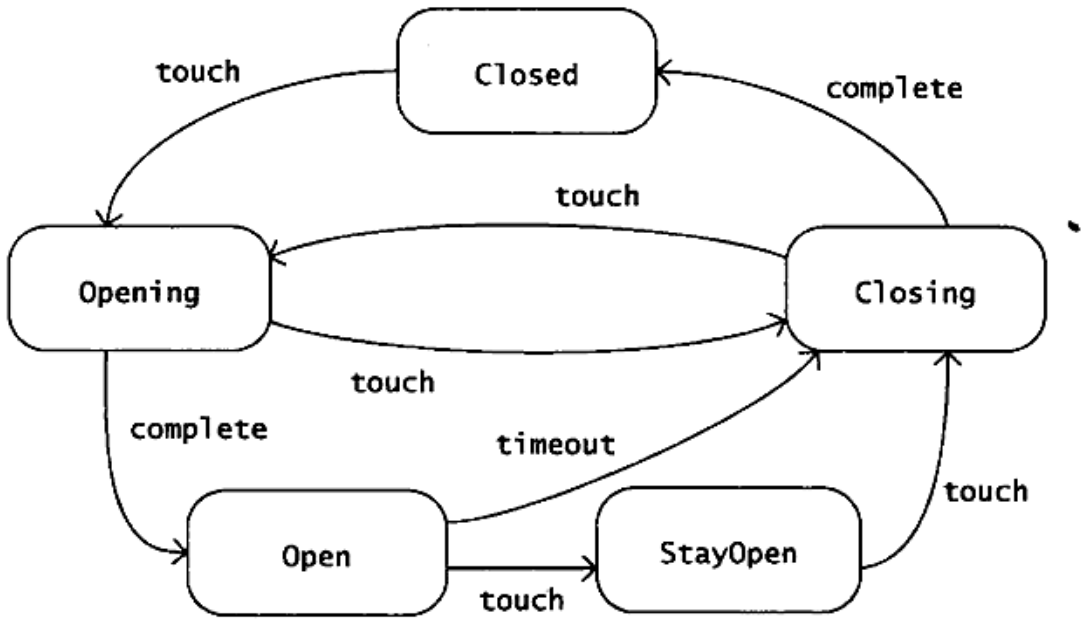


**图 22.1　传送带入口门通过触发一个单独的按钮来对门的状态进行控制**

图 22.1 是一个 UML 状态图。相对于文字表达来讲，这幅图描述得更加清楚。

> **挑战 22.1**
>
> 假设打开了传送带入口门，并将材料投进去。请问除了等待超时让门自动关闭外，是否还有其他方式可以关闭门？
>
> 答案参见第 349 页

可以为传送软件提供一个 Door 对象，随着状态的变化，传送软件可以更新对象的状态。

图 22.2 展示了 Door 类。

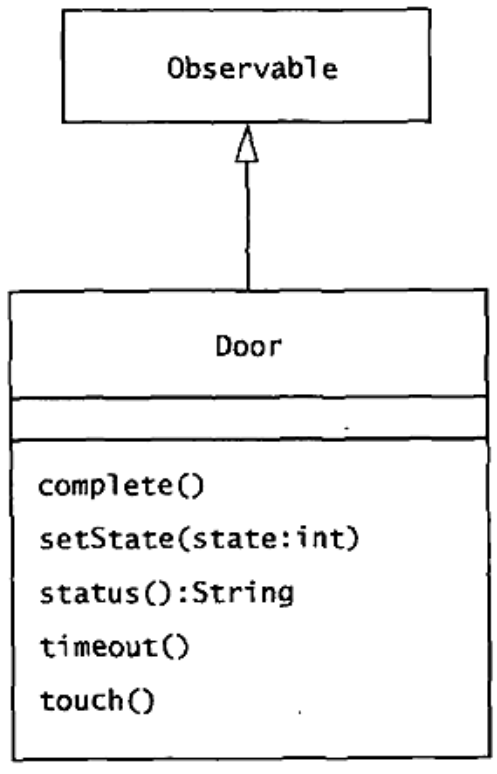


**图 22.2　Door 类依赖于传送带触发的状态改变事件，对传送带入口门进行建模**

Door 类是 Observable 的子类，诸如 GUI 等客户对象，可以显示门的状态。该类中定义了门的几种状态：

```
package com.oozinoz.carousel;
import java.util.Observable;

public class Door extends Observable {
    public final int CLOSED = -1;
    public final int OPENING = -2;
    public final int OPEN = -3;
    public final int CLOSING = -4;
    public final int STAYOPEN = -5;

    privateint state = CLOSED;
    // ...
}
```

（如果使用 Java 5，可以选择使用枚举类型。）

显而易见，状态的文本描述取决于门所处的状态。

```
public String status() {
    switch (state) {
```

```
            case OPENING:
                return "Opening";
            case OPEN:
                return "Open";
            case CLOSING:
                return "Closing";
            case STAYOPEN:
                return "StayOpen";
            default:
                return "Closed";
        }
    }
```

当用户触发传送带入口门的按键时，程序会调用 Door 对象的 touch()方法。图 22.1 模拟了 Door 类中的状态转移。

```
    public void touch() {
        switch (state) {
            case OPENING:
            case STAYOPEN:
                setState(CLOSING);
                break;
            case CLOSING:
            case CLOSED:
                setState(OPENING);
                break;
            case OPEN:
                setState(STAYOPEN);
                break;
            default:
                throw new Error("can't happen");
        }
    }
```

Door 类中的 setState()方法负责通知那些监听门状态改变的观察者。

```
    private void setState(int state) {
        this.state = state;
        setChanged();
        notifyObservers();
    }
```

**挑战 22.2**

请写出 Door 类中 complete()和 timeout()的代码。

答案参见第 349 页

# 重构为状态模式

Door 类的代码有些复杂，因为 state 变量是类的全局变量。此外，你可能发现很难比较状态转移的方法，特别是图 22.1 所示的状态机中的 touch()方法。此时，引入状态模式可以帮助你简化代码。要在本例中使用状态模式，需要将门的每种状态都写到一个独立的类中，如图 22.3 所示。

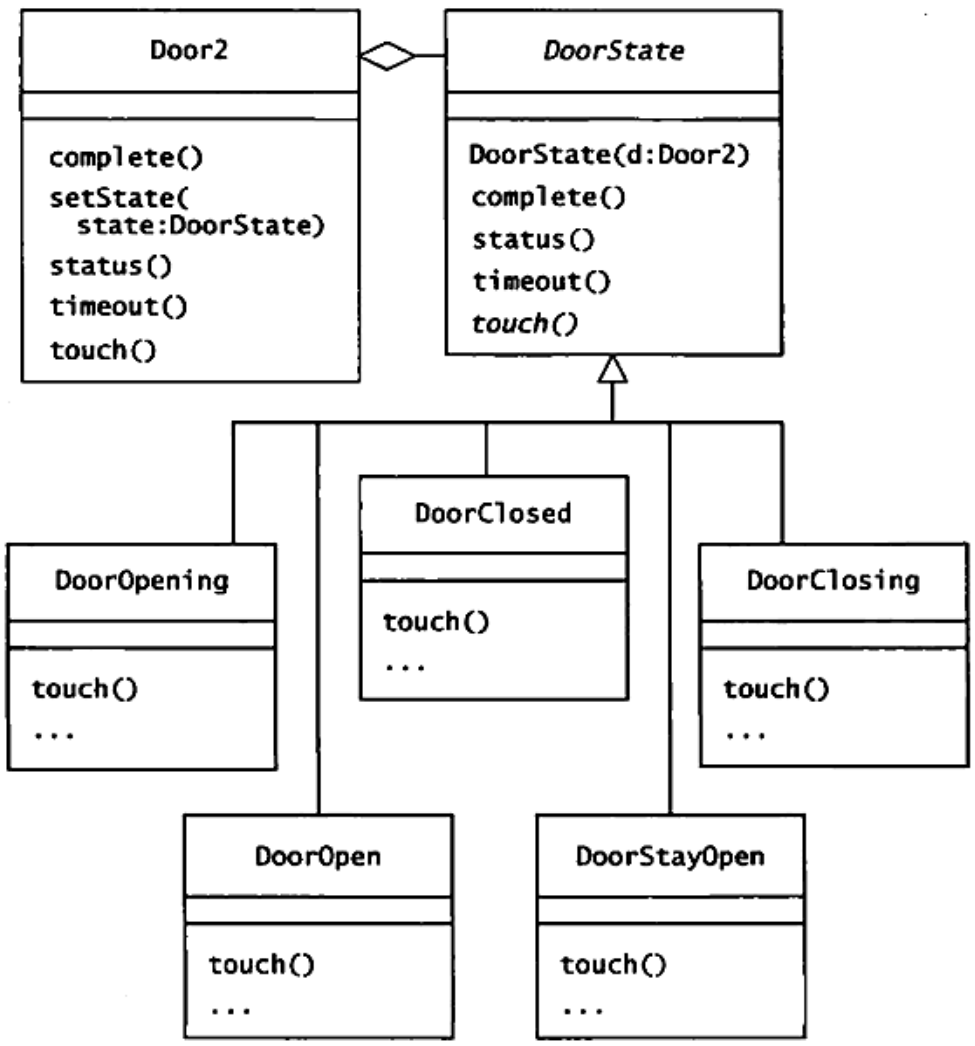


图 22.3　该图展示了将门的状态机映射成独立类后的样子

图 22.3 展示的重构将门的各种状态分别创建成相应的类。每个类都包含相应的逻辑来处理状态行为，即当门处于某个特定状态时，触动按钮后应有的行为。例如，DoorClosed.java 文件包含如下代码：

```
package com.oozinoz.carousel;
public class DoorClosed extends DoorState {
    public DoorClosed(Door2 door) {
        super(door);
    }
    public void touch() {
        door.setState(door.OPENING);
    }
}
```

DoorClosed 类中的 touch()方法通知 Door2 门对象的最新状态。Door2 对象通过 DoorClosed 类的构造函数传入。当前的设计要求每个状态对象都要保留对 Door2 对象的引用，确保状态对象可以通知门对象的状态变化。当前设计要求每个状态对象都引用某个特定的门对象，以便每个状态对象只能操作一个单独的门对象。下一节将指出如何修改该设计，以便一组单独的状态集可以满足任意数量的门对象。当前的设计则要求在创建 Door2 对象的同时，创建一组属于门对象的状态集。

```
package com.oozinoz.carousel;
import java.util.Observable;

public class Door2 extends Observable {
    public final DoorState CLOSED = new DoorClosed(this);
    public final DoorState CLOSING = new DoorClosing(this);
    public final DoorState OPEN = new DoorOpen(this);
    public final DoorState OPENING = new DoorOpening(this);
    public final DoorState STAYOPEN = new DoorStayOpen(this);

    private DoorState state = CLOSED;
    // ...
}
```

抽象类 DoorState 需要一些子类来实现 touch()方法。每个状态类都有一个 touch()转换函数，这和状态机是一致的。DoorState 类的其他转换方法都是无关方法，因此子类可以重写或者忽略不相干的消息。

```
package com.oozinoz.carousel;

public abstract class DoorState {
    protected Door2 door;

    public abstract void touch();

    public void complete() { }

    public void timeout() { }

    public String status() {
        String s = getClass().getName();
        returns.substring(s.lastIndexOf('.') + 1);
    }

    public DoorState(Door2 door) {
        this.door = door;
    }
}
```

注意，所有的状态对象都包含了 status()方法，它比重构之前要简单许多。

从接收状态的改变来看，新的设计并没有改变 Door2 对象的角色。现在，Door2 对象只是简单地将这些改变传递给当前的 state 对象：

```
package com.oozinoz.carousel;
import java.util.Observable;

public class Door2 extends Observable {
    // 变量与构造函数 ...

    public void touch() {
        state.touch();
    }

    public void complete() {
        state.complete();
    }
```

```
    public void timeout() {
        state.timeout();
    }

    public String status() {
        return state.status();
    }

    protected void setState(DoorState state) {
        this.state = state;
        setChanged();
        notifyObservers();
    }
}
```

在当前设计中，touch()、complete()、timeout()和 status()方法展示了多态性的应用。每个方法都表示一种状态的迁移。操作是固定的，但是接收的类——状态类，却是不一样的。多态的原则是方法的执行依赖于接收的类以及该类中的方法签名。当调用 touch()方法时会发生什么呢？答案取决于门对象的状态。代码依然可以有效地进行转换，由于引入了多态，代码比之以前更为简洁。

Door2 类中的 setState()方法现在被用于 DoorState 的子类。这些子类类似于图 22.1 状态机中对应的状态。例如，DoorOpen 类处理有关 touch()和 timeout()的调用，对应状态机中 Open 状态的两个转换：

```
package com.oozinoz.carousel;
public class DoorOpen extends DoorState {
    public DoorOpen(Door2 door) {
        super(door);
    }

    public void touch() {
        door.setState(door.STAYOPEN);
    }

    public void timeout() {
        door.setState(door.CLOSING);
    }
}
```

**挑战 22.3**

写出 DoorClosing.java 的代码。

答案参见第 349 页

新的设计使代码变得更加简单，但或许你对现在的设计仍不满意，因为 Door 类使用的“常量”实际上是一个局部变量。

## 使状态成为常量

状态模式将状态相关的逻辑转移到独立的类中，用以表示对象的状态。然而该模式并没有指出如何管理状态和主对象之间的通信和依赖关系。在之前的设计中，每个状态类都在构造函数中接收一个 Door 对象。状态对象保存这个对象的引用，并利用该对象更新门的状态。这样的设计并不算坏，但它确实造成了在实例化 Door 对象时，伴随着对一组 DoorState 对象的实例化。若能创建一组独立的静态 DoorState 对象，然后让 Door 状态管理因为状态改变产生的所有更新，会是更好的一种设计。

使状态对象变成常量的一种做法是，让当前的状态类标识下一个状态，让 Door 类去更新 state 变量。根据这样的设计，Door 类的 touch()方法就会采用如下方式更新 state 变量：

```
public void touch() {
    state = state.touch();
}
```

注意，Door 类的 touch()方法返回了 void，DoorState 的子类也会实现 touch()方法，但是，这些方法会返回 DoorState 值。例如，DoorOpen 类的 touch()方法：

```
public DoorState touch() {
    return DoorState.STAYOPEN;
}
```

在这个设计中，DoorState 对象不包含任何 Door 对象的引用。因此，应用程序的所有 DoorState 对象只需要引用一个 Door 对象的实例。

使 DoorState 对象变成常量的另一种方式是在状态迁移时传递核心的 Door 对象。你可以为 complete()、timeout()、touch()这些与状态改变有关的方法都加上一个 Door 参数。这些方法接收一个 Door 对象来更新其状态，而不需要在它的内部保留 Door 对象的引用。

**挑战 22.4**

完成图 22.4，将 DoorState 对象设计为常量，并在状态迁移时传递 Door 对象。

答案参见第 350 页

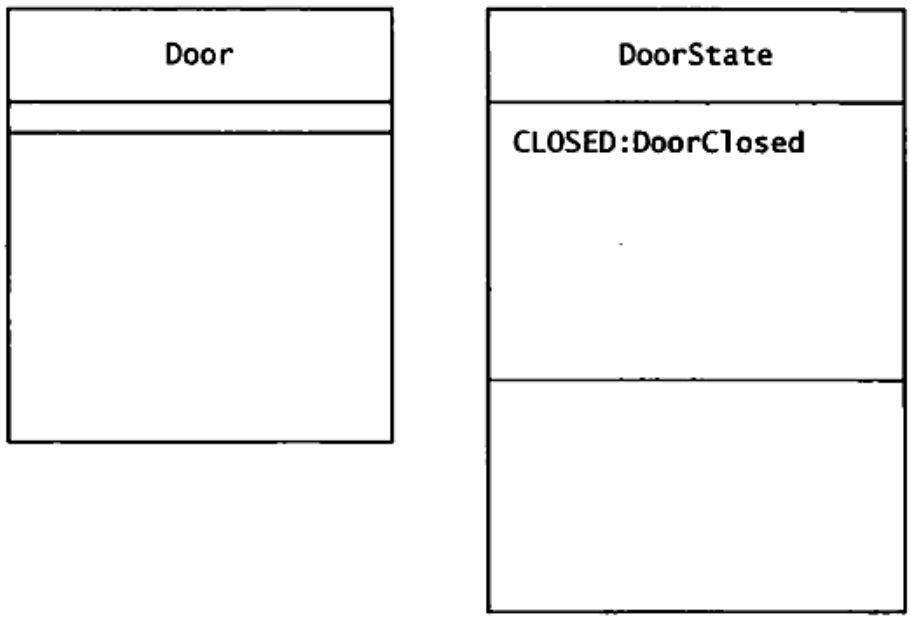


图 22.4　完成这个设计图，使之可以表示连续的门状态

在应用状态模式时，可以针对状态改变时的通信机制进行完全自由的设计。状态类维护了一个主对象的引用，该对象的状态已经建模。或者，可以在状态迁移时传递这个对象，也可以让这些状态子类决定下一个状态，但不更新主对象。使用哪种方式取决于你的应用场景或者个人喜好。

如果这些状态类在多线程环境下使用，要确保这些状态迁移方法是同步的，并保证在同一时间，两个线程在更改状态时不会有冲突。

状态模式的作用是让任何一个状态的逻辑都集中放在一个单独的类中。

## 小结

总体而言，对象的状态取决于对象实例中变量值的集合。在一些场景下，对象中的绝大多数属性一旦被赋值后都不会改变，但其中某个属性值经常变化，并在类逻辑中扮演重要角色。该属性可能会代表整个对象的状态，甚至被命名为 state。

在对现实世界中拥有重要状态的实体进行建模时，可能会发生一些显著的状态迁移，例如事务或机器的状态迁移。在这些场景下，依赖于对象状态的逻辑可能会分布在多个方法中。可以通过将状态相关的行为转移到状态对象类中，以此来简化这些代码。这会让每个状态类都包含领域中一个状态的行为，同时也让状态类对应状态机中的相应状态。

为了处理状态间的迁移，可以让主对象包含一组状态的引用。或者在状态迁移的调用中，将主对象传递给状态改变的类。你也可以让状态类的信息提供者仅仅给出下一个状态，而不去更新主对象。无论怎样管理状态迁移，状态模式都会通过将对象的不同状态操作，分散到一组类的集合中，从而简化代码。