8.19 实现状态对象或者状态机¶

问题¶

你想实现一个状态机或者是在不同状态下执行操作的对象，但是又不想在代码中出现太多的条件判断语句。

解决方案¶

在很多程序中，有些对象会根据状态的不同来执行不同的操作。比如考虑如下的一个连接对象：

```
class Connection:
    """普通方案，好多个判断语句，效率低下~~"""

    def __init__(self):
        self.state = 'CLOSED'

    def read(self):
        if self.state != 'OPEN':
            raise RuntimeError('Not open')
        print('reading')

    def write(self, data):
        if self.state != 'OPEN':
            raise RuntimeError('Not open')
        print('writing')

    def open(self):
        if self.state == 'OPEN':
            raise RuntimeError('Already open')
        self.state = 'OPEN'

    def close(self):
        if self.state == 'CLOSED':
            raise RuntimeError('Already closed')
        self.state = 'CLOSED'
```

这样写有很多缺点，首先是代码太复杂了，好多的条件判断。其次是执行效率变低， 因为一些常见的操作比如read()、write()每次执行前都需要执行检查。

一个更好的办法是为每个状态定义一个对象：

```
class Connection1:
    """新方案——对每个状态定义一个类"""

    def __init__(self):
        self.new_state(ClosedConnectionState)

    def new_state(self, newstate):
        self._state = newstate
        # Delegate to the state class

    def read(self):
        return self._state.read(self)
```

```
    def write(self, data):
        return self._state.write(self, data)

    def open(self):
        return self._state.open(self)

    def close(self):
        return self._state.close(self)


# Connection state base class
class ConnectionState:
    @staticmethod
    def read(conn):
        raise NotImplementedError()

    @staticmethod
    def write(conn, data):
        raise NotImplementedError()

    @staticmethod
    def open(conn):
        raise NotImplementedError()

    @staticmethod
    def close(conn):
        raise NotImplementedError()


# Implementation of different states
class ClosedConnectionState(ConnectionState):
    @staticmethod
    def read(conn):
        raise RuntimeError('Not open')

    @staticmethod
    def write(conn, data):
        raise RuntimeError('Not open')

    @staticmethod
    def open(conn):
        conn.new_state(OpenConnectionState)

    @staticmethod
    def close(conn):
        raise RuntimeError('Already closed')


class OpenConnectionState(ConnectionState):
    @staticmethod
    def read(conn):
        print('reading')

    @staticmethod
    def write(conn, data):
        print('writing')

    @staticmethod
    def open(conn):
```

```
    raise RuntimeError('Already open')

  @staticmethod
  def close(conn):
    conn.new_state(ClosedConnectionState)
```

下面是使用演示：

```
>>> c = Connection()
>>> c._state
<class '__main__.ClosedConnectionState'>
>>> c.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 10, in read
    return self._state.read(self)
  File "example.py", line 43, in read
    raise RuntimeError('Not open')
RuntimeError: Not open
>>> c.open()
>>> c._state
<class '__main__.OpenConnectionState'>
>>> c.read()
reading
>>> c.write('hello')
writing
>>> c.close()
>>> c._state
<class '__main__.ClosedConnectionState'>
>>>
```

讨论¶

如果代码中出现太多的条件判断语句的话，代码就会变得难以维护和阅读。 这里的解决方案是将每个状态抽取出来定义成一个类。

这里看上去有点奇怪，每个状态对象都只有静态方法，并没有存储任何的实例属性数据。 实际上，所有状态信息都只存储在 `Connection` 实例中。 在基类中定义的 `NotImplementedError` 是为了确保子类实现了相应的方法。 这里你或许还想使用8.12小节讲解的抽象基类方式。

设计模式中有一种模式叫状态模式，这一小节算是一个初步入门！