

状态模式 - 文斯莫克·山治

📅 发表于 2022-09-20 | 🔄 更新于 2023-04-06 | 📁 设计模式

| 📄 字数总计: 2.2k | ⌚ 阅读时长: 8分钟 | 👁 阅读量: 670 | 💬 评论数: 1



配套视频课程已更新完毕，大家可通过以下两种方式观看视频讲解：



关注公众号：👉 爱编程的大丙 ，或者进入 👉 大丙课堂 学习。



苏丙楦

合抱之木，生于毫末；九层之台，起于垒土；千里之行，始于足下。

1. 厨师山治


山治是文斯莫克家族的第三子，基因改造人，由于小时候未能觉醒能力而被父亲文斯莫克·伽治放逐到东海。遇见恩师哲普后在海上餐厅巴拉蒂担任厨师。为了寻找传说之海ALL BLUE而随草帽一伙踏入伟大航路。

文章	标签	分类
134	37	12

 大丙课堂



公告

微信公众号 爱编程的大丙 和
大丙课堂 上线了, 可
点击上方  图标关注 ~~~

≡ 目录

1. 厨师山治
2. 山治的一天
3. 结构图

最新文章



CMake 保姆级教程
(下)

2023-03-15



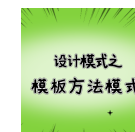
CMake 保姆级教程
(上)

2023-03-06



访问者模式 - 再见,
香波地群岛

2022-09-22



模板方法模式 - 和平
主义者

2022-09-21



状态模式 - 文斯莫克·
山治

2022-09-20

在海上航行的每一天里，最忙的应该就是山治了。他需要准备食材、做早饭、做午饭、做下午茶、做晚饭，一整天都在不同的状态之间忙碌。

在设计模式中有一种和山治工作状态类似的模式叫做状态模式。**状态模式就是在一个类的内部会有多种状态的变化，因为状态变化从而导致其行为的改变，在类的外部看上去这个类就像是自身发生了改变一样。**

在日常生活中由于内部属性的变化导致外在样貌或者行为发生改变的例子比比皆是，比如：

- 人在幼年、童年、少年、中年、老年各个使其的形态都是不一样的
- 工作期间，上午、中午、下午、傍晚、深夜的工作状态也不一样
- 人的心情不同时，会有喜、怒、哀、乐
- 手机在待机、通话、充电、游戏时的状态也不一样
- 文章的发表会有草稿、审阅、发布状态

状态模式和策略模式比较类似，策略模式中的各个策略是独立的不关联的，但是状态模式下的对象的各种状态可以是独立的也可以是相互依赖的，比如上面关于文章的发布的例子：

- 普通用户的文章草稿发表之后被审阅，审阅失败重新变成草稿
- 管理用户的文章操作发布成功变成已发表状态，发布失败重新变成草稿

🌀 2. 山治的一天

假设这一天草帽一伙一直在海上航行，没有遇到海军也没有遇到极端天气，对于船上的成员来说这又是可以大饱口福的一天。

🔗2.1 开始工作

山治作为厨师一整天都在忙碌，我们在这里简单的把他的工作状态划分一下：**上午的状态、中午的状态、下午的状态、晚上的状态**。不论哪种状态他都是在认真的在完成自己的本职工作，只不过在不同的时间点工作的内容是不一样的。所以，我们可以给这些状态定义一个基类：

```
1 // State.h
2 // 抽象状态
3 class Sanji;
4 class AbstractState
5 {
6 public:
7     virtual void working(Sanji* sanji) = 0;
8     virtual ~AbstractState() {}
9 };
```

由于这个状态是属于山治的，所以在这个抽象的状态类中通过提供的工作函数 `working()` 的参数指定了这个状态的所有者，在这里只是对 **山治类 Sanji** 做了一个声明 **第3行**，尽量不要在这个头文件中包含山治的头文件，否则会造成头文件重复包含（**因为山治类和状态类需要相互引用对方**）。

有了上面的抽象的状态类，就可以基于这个基类把山治全天对应的状态类的子类依次定义出来了：

头文件 State.h

```
1 // 上午状态
2 class ForenoonState : public AbstractState
3 {
4 public:
```

```
5     void working(Sanji* sanji) override;
6 };
7
8 // 中午状态
9 class NoonState : public AbstractState
10 {
11 public:
12     void working(Sanji* sanji) override;
13 };
14
15 // 下午状态
16 class AfternoonState : public AbstractState
17 {
18 public:
19     void working(Sanji* sanji) override;
20 };
21
22 // 晚上状态
23 class EveningState : public AbstractState
24 {
25 public:
26     void working(Sanji* sanji) override;
27 };
```

源文件 State.cpp

```
1 #include <iostream>
2 #include "State.h"
3 #include "Sanji.h"
4 using namespace std;
5
```

```
6 void ForenoonState::working(Sanji* sanji)
7 {
8     int time = sanji->getClock();
9     if (time < 8)
10    {
11        cout << "当前时间<" << time << ">点, 准备早餐, 布鲁克得多喝点牛奶..." <<
12    }
13    else if (time > 8 && time < 11)
14    {
15        cout << "当前时间<" << time << ">点, 去船头钓鱼, 储备食材..." << endl;
16    }
17    else
18    {
19        sanji->setState(new NoonState);
20        sanji->working();
21    }
22 }
23
24 void NoonState::working(Sanji* sanji)
25 {
26     int time = sanji->getClock();
27     if (time < 13)
28     {
29        cout << "当前时间<" << time << ">点, 去厨房做午饭, 给路飞多做点肉..." <<
30    }
31    else
32    {
33        sanji->setState(new AfternoonState);
34        sanji->working();
35    }
36 }
```

```
37
38 void AfternoonState::working(Sanji* sanji)
39 {
40     int time = sanji->getClock();
41     if (time < 15)
42     {
43         cout << "当前时间<" << time << ">点, 准备下午茶, 给罗宾和娜美制作爱心甜点."
44     }
45     else if (time > 15 && time < 18)
46     {
47         cout << "当前时间<" << time << ">点, 和乔巴去船尾钓鱼, 储备食材..." << e
48     }
49     else
50     {
51         sanji->setState(new EveningState);
52         sanji->working();
53     }
54 }
55
56 void EveningState::working(Sanji* sanji)
57 {
58     int time = sanji->getClock();
59     if (time < 19)
60     {
61         cout << "当前时间<" << time << ">点, 去厨房做晚饭, 让索隆多喝点汤..." <<
62     }
63     else
64     {
65         cout << "当前时间<" << time << ">点, 今天过得很高兴, 累了睡觉了..." << e
66     }
67 }
```


在状态类的源文件中包含了山治类的头文件，因为在这些状态类的子类中重写 `working()` 函数的时候，需要通过山治类对象调用他的成员函数。通过上面的代码可以看到山治在不同的时间状态下所做的事情是不一样的。

另外我们可以看到状态模式下各个模式之间是可以有依赖关系的，这一点和策略模式是有区别的，策略模式下各个策略都是独立的，当前策略不知道有其它策略的存在。

🔗 2.2 山治

上面定义的一系列的状态都是属于山治这个对象的，只不过通过状态模式来处理山治全天的工作状态变化的时候，把他们分离出去了，成了独立的个体，从逻辑上讲他们之间是包含和被包含的关系，从UML类图的角度来讲他们之间是组合（整体和部分）关系。关于山治这个类我们可以这样定义：

```
1 // Sanji.h
2 #pragma once
3 #include "State.h"
4
5 class Sanji
6 {
7 public:
8     Sanji()
9     {
10         m_state = new ForenoonState;
11     }
12     void working()
13     {
14         m_state->working(this);
```

```
15     }
16     void setState(AbstractState* state)
17     {
18         if (m_state != nullptr)
19         {
20             delete m_state;
21         }
22         m_state = state;
23     }
24     void setClock(int time)
25     {
26         m_clock = time;
27     }
28     int getClock()
29     {
30         return m_clock;
31     }
32     ~Sanji()
33     {
34         delete m_state;
35     }
36 private:
37     int m_clock = 0;    // 时钟
38     AbstractState* m_state = nullptr;
39 };
```

在山治类中有两个私有成员变量:

1. **m_clock** : 通过这整形的时钟变量来描述一天中当前这个时刻的时间点
2. **m_state** : 通过这个状态指针来保存当前描述山治状态的对象

关于山治类的成员函数有以下这么几个：

1. `working()`：工作函数，在不同的时间状态下，工作的内容也不同
2. `setClock()`：设置当前的时间
3. `getClock()`：得到当前的时间
4. `setState()`：设置山治当前的状态

🔗2.3 工作日志

我们可以修改山治类内部的时钟的值来模拟时间的流逝，这样山治的状态就会不停地发生变化了：

```
1  int main()
2  {
3      Sanji* sanji = new Sanji;
4      // 时间点
5      vector<int> data{7, 10, 12, 14, 16, 18, 22};
6      for (const auto& item : data)
7      {
8          sanji->setClock(item);
9          sanji->working();
10     }
11     delete sanji;
12
13     return 0;
14 }
```

最后我们来看一下山治这一整天都干了些什么吧：

- 1 当前时间<7>点, 准备早餐, 布鲁克得多喝点牛奶...
- 2 当前时间<10>点, 去船头钓鱼, 储备食材...
- 3 当前时间<12>点, 去厨房做午饭, 给路飞多做点肉...
- 4 当前时间<14>点, 准备下午茶, 给罗宾和娜美制作爱心甜点...
- 5 当前时间<16>点, 和乔巴去船尾钓鱼, 储备食材...
- 6 当前时间<18>点, 去厨房做晚饭, 让索隆多喝点汤...
- 7 当前时间<22>点, 今天过得很高兴, 累了睡觉了...

🔗3. 结构图

最后画一下状态模式对应的UML类图（学会状态模式之后，要先画UML类图再写程序。）

如果对象需要根据当前自身状态进行不同的行为，同时状态的数量非常多且与状态相关的代码会频繁变更或者类对象在改变自身行为时需要使用大量的条件语句时，可使用状态模式。

文章作者: 苏丙楹



文章链接: <https://subingwen.cn/design-patterns/state/>

版权声明: 本博客所有文章除特别声明外，均采用 [CC BY-NC-SA 4.0](#) 许可协议。转载请注明来自 爱编程的大丙！

设计模式

 打赏

上一篇

设计模式之
模板方法模式设计模式之
策略模式

下一篇

策略模式 - 蒙奇·D·路飞

👍 相关推荐

2022-08-27
UML 类图
设计模式

2022-08-31
设计模式之
抽象工厂模式 - 弗三奇
抽象工厂模式

2022-09-06
设计模式之
桥接模式 - 大海贼时代
桥接模式



评论

昵称

邮箱

网址(http://)

来都来了, 说点什么吧...



提交

1 评论

**Anonymous**

Wechat 8.0.33.2320

Android 12

2023-03-29

[回复](#)

1

Powered By [Valine](#)
v1.5.1

©2021 - 2023 By 苏丙楹

冀ICP备2021000342号-1



冀公网安备 13019902000353号