

装饰模式 - 黑胡子

📅 发表于 2022-09-05 | ⌚ 更新于 2023-04-06 | 📁 设计模式

| 📄 字数总计: 2.7k | ⌚ 阅读时长: 8 分钟 | 👁 阅读量: 475 | 💬 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. 马歇尔·D·蒂奇

在海贼世界中，马歇尔·D·蒂奇 绰号 黑胡子，他是 黑胡子海贼团的提督、新世界四皇之一，自然系 暗暗果实 和超人系 震震果实 能力者，据说他还可以吃下第三个恶魔果实（就目前剧情而言尚未盖棺定论）。

对于黑胡子来说，它拥有的恶魔果实能力并不是与生俱来的，也就是后天获得，恶魔果实能力是对黑胡子原有实力的加成。黑胡子获得的这种恶魔果实能力和设计模式中的装饰模式差不多，都是动态的给一个对象绑定额外的属性（比如：能力、职责、功能等）。

关于装饰模式也可以称之为封装模式，所谓的封装就是在原有行为之上进行拓展，并不会改变该行为，看下面的例子：

1. 在进行网络通信的时候，数据是基于 IOS 七层或四层网络模型（某些层合并之后就是四层模型）进行传输，通过下图可得知从应用层到物理层，数据每向下走一层就会被封装一层，最后将封装好的数据以比特流的方式发送给接收端。封装之后数据只是变得更复杂了，并没有改变它是数据的本质。

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

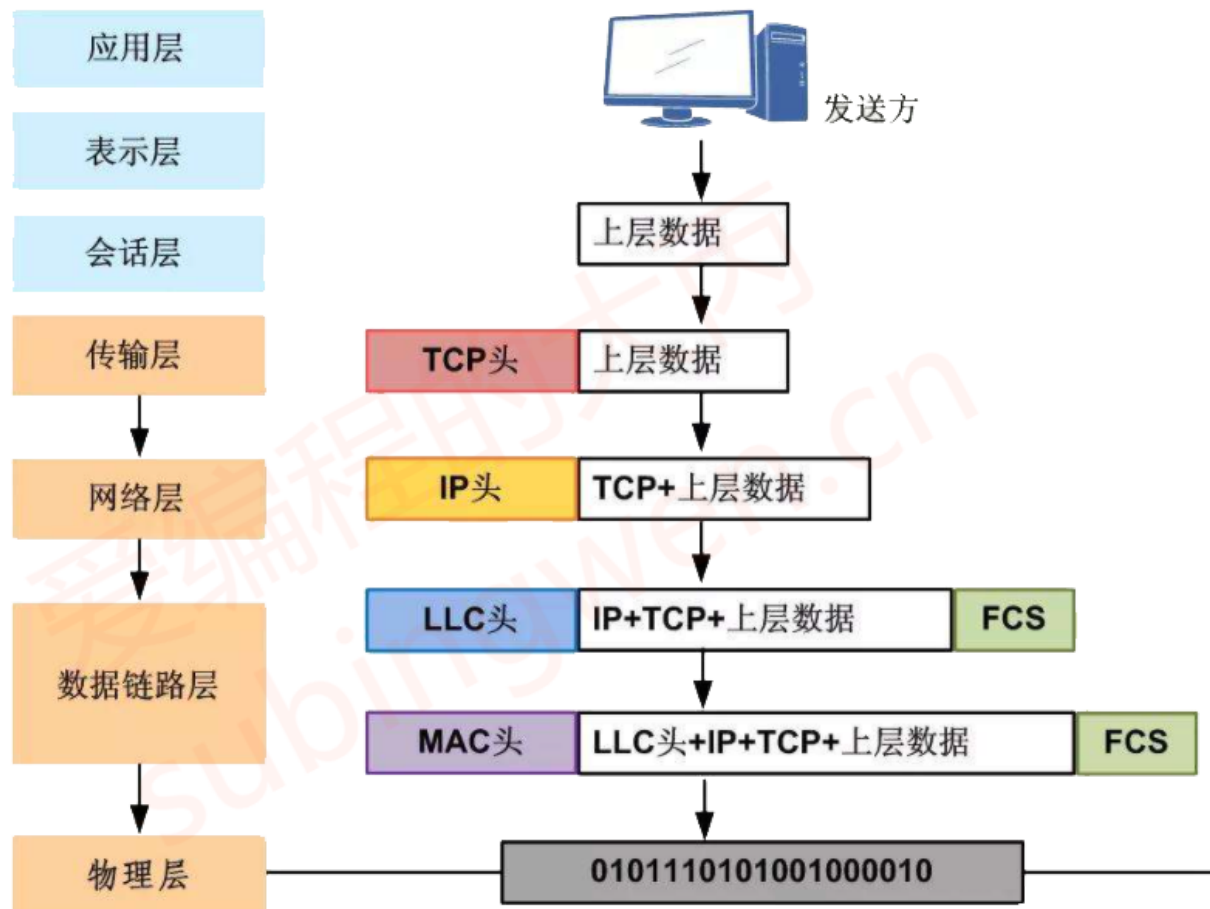
大丙课堂 上线了，可

点击上方 图标关注 ~ ~ ~

三 目录

1. 马歇尔·D·蒂奇
2. 解构黑胡子
3. 结构图

🕒 最新文章



2. A 端和 B 端进行网络通信，默认数据是在网络环境中裸奔的，如果要对数据进行装饰（也就是封装）就可以在发送数据之前对其加密，接收端收到数据之后对其解密。加解密是对数据的装饰，但是没有改变数据的本质。
3. 平时都是穿长衣长裤，疫情来袭之后穿上了防护服。防护服是对人的装饰，没有改变本体是人的本质。



CMake 保姆级教程
(下)

2023-03-15



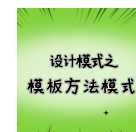
CMake 保姆级教程
(上)

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

上述例子中都是对原有行为进行了拓展，但是并没有改变原有行为，就好比饿了去煮面条，为了使其更美味最终会对其进行装饰，做成 打卤面、炸酱面、热干面、油泼面等，不论怎么处理最终得到的还是面条，最终不可能得到一锅酱牛肉，大方向是不会变的。

🌀2. 解构黑胡子

从概念上对装饰模式有了一定的了解之后，继续分析黑胡子的个人战斗力，根据文章开头对他的介绍，可以知道他的能力来自三个不同的方向：

1. 与生俱来加上后天努力练就的本领
2. 来自自然系·暗暗果实的能力
3. 来自超人系·震震果实的能力

所以这两个恶魔果实的能力就是对黑胡子个人战力的装饰（加成）。另外，还需要明确一点，对于恶魔果实来说不是只有黑胡子能吃，谁吃了都会拥有对应的果实能力，这样这个人的战斗力也就提升了。

🌀2.1 战魂

人自身是拥有战斗力的，而恶魔果实又可以给人附加战斗力，所以我们可以定义一个战士的抽象类（这个抽象类不能被实例化）。



C++



```
1 // 战士的抽象类
2 class Soldier
3 {
4 public:
5     Soldier() {}
6     Soldier(string name) : m_name(name) {}
7     string getName()
8     {
9         return m_name;
10    };
11    virtual void fight() = 0;
12    virtual ~Soldier() {}
13 protected:
14     string m_name = string();
15 };
```

有了这个抽象类就可以对某个人，或者某个恶魔果实的战力进行具体的实现，也就是说它们需要继承这个抽象类。

- 所有的战士都可以战斗 – `fight()`
- 所有的战士都有自己的名字
 - 设置名字 – 构造函数 `Soldier(string name)`
 - 获取名字 – `string getName()`

🔗 2.2 黑胡子

上面的战士是一个抽象类，如果想要对它进行实例化就需要写一个子类继承这个抽象类，下面我们定义一个黑胡子类：

▼ C++

```
1  // 黑胡子(Marshall·D·Teach)
2  class Teach : public Soldier
3  {
4  public:
5      using Soldier::Soldier;
6      void fight() override
7      {
8          cout << m_name << "依靠惊人的力量和高超的体术战斗..." << endl;
9      }
10 };
```

在黑胡子类中主要是重写了从父类继承的纯虚函数，这样黑胡子这个类就可以被实例化，得到对应的黑胡子对象了。

🌀 2.3 附魔

如果黑胡子想要让自己的实力再提升一个层次，就需要得到外部力量的辅助，可行的方案就是吃恶魔果实，这样就相当于给自己附魔了。恶魔果实的作用是对战士的战力进行装饰，使其战力得到大大的提升，所以恶魔果实类也可以先继承战士这个类。

▼ C++

```
1  // 抽象的恶魔果实
2  class DevilFruit : public Soldier
```

```
3  {
4  public:
5      // 指定要给哪个人吃恶魔果实 -- 附魔
6      void enchantment(Soldier* soldier)
7      {
8          m_human = soldier;
9          m_name = m_human->getName();
10     }
11     virtual ~DevilFruit() {}
12 protected:
13     Soldier* m_human = nullptr;
14 };
```

上面的恶魔果实类 `DevilFruit` 继承了战士类 `Soldier` 之后还是一个抽象类，关于这个类有以下几点需要说明：

1. 在 `DevilFruit` 类中没有重写父类 `Soldier` 的纯虚函数 `fight()`，所以它还是抽象类
2. 恶魔果实有很多种类，每种恶魔果实能力不同，所以战斗方式也不同，因此需要在恶魔果实的子类中根据每种果实能力去重写作战函数 `fight()` 的行为。
3. 恶魔果实 `DevilFruit` 类的作用是给某个 `Soldier` 的子类对象附魔，所以在类内部提供了一个附魔函数 `enchantment(Soldier* soldier)`，参数就是即将要得到恶魔果实能力的那个战士。

🌀 2.4 群魔乱舞

黑胡子目前一共吃下了两颗恶魔果实：自然系 `暗暗果实` 和超人系 `震震果实`，所以需先定义两个恶魔果实的子类：

C++



```
1 // 暗暗果实
2 class DarkFruit : public DevilFruit
3 {
4 public:
5     void fight() override
6     {
7         m_human->fight();
8         // 使用当前恶魔果实的能力
9         cout << m_human->getName()
10             << "吃了暗暗果实，可以拥有黑洞一样的无限吸引力..." << endl;
11         warning();
12     }
13 private:
14     void warning()
15     {
16         cout << m_human->getName()
17             << "你要注意：吃了暗暗果实，身体元素化之后不能躲避攻击，会吸收所有伤害!"
18     }
19 };
20
21 // 震震果实
22 class QuakeFruit : public DevilFruit
23 {
24 public:
25     void fight() override
26     {
27         m_human->fight();
28         cout << m_human->getName()
29             << "吃了震震果实，可以在任意空间引发震动，摧毁目标..." << endl;
```



```
30     }  
31 };
```

关于这两个恶魔果实子类需要说明以下几点:

1. 在重写父类的 `fight()` 函数的时候, 用当前恶魔果实能力和战士的自身能力进行了加成, 调用了战士对象的作战函数 `m_human→fight()`, 在原有基础上提升了其战斗力。
2. 在两个恶魔果实子类中, 可以根据实际需要定义类独有的方法, 比如: `DarkFruit` 类中有 `warning()` 方法, `QuakeFruit` 类中却没有。
3. 再次强调, 这两个子类都继承了父类的附魔函数 `enchantment(Soldier* soldier)`, 这样就可以完成对战士战力的加成(装饰)了。

假设黑胡子确实是可以吃下第三个恶魔果实, 并且发现了一颗神奇的超人系恶魔果实 **大饼果实**, 可以将身边的一切物体变成大饼, 帮助自己和队友快速回血。

```
▼ C++  
  
1 // 大饼果实  
2 class PieFruit : public DevilFruit  
3 {  
4 public:  
5     void fight() override  
6     {  
7         m_human→fight();  
8         cout << m_human→getName()  
9             << "吃了大饼果实, 获得大饼铠甲...!" << endl;  
10        ability();  
11    }
```

```
12
13     void ability()
14     {
15         cout << "最强辅助 -- 大饼果实可以将身边事物变成大饼，帮助自己和队友回血..."
16     }
17 };
```

使用装饰模式，可以非常方便地给任意一个战士增加战斗技能，而无需修改原有代码，完全符合开放 – 封闭原则。

🌀2.5 六边形战士


最后展示一下无敌的四皇之一的黑胡子的战斗力：

▼ C++

```
1  int main()
2  {
3      Teach* teach = new Teach("马歇尔·D·蒂奇");
4      DarkFruit* dark = new DarkFruit;
5      QuakeFruit* quake = new QuakeFruit;
6      PieFruit* pie = new PieFruit;
7      // 黑胡子吃了暗暗果实
8      dark->enchantment(teach);
9      // 黑胡子又吃了震震果实
10     quake->enchantment(dark);
11     // 黑胡子又吃了大饼果实
12     pie->enchantment(quake);
13     // 战斗
14     pie->fight();
```


```
15     delete teach;
16     delete dark;
17     delete quake;
18     delete pie;
19     return 0;
20 }
```

输出的结果如下:

▼ C++ 

- 1 马歇尔·D·蒂奇依靠惊人的力量和高超的体术战斗...
- 2 马歇尔·D·蒂奇吃了暗暗果实，可以拥有黑洞一样的无限吸引力...
- 3 马歇尔·D·蒂奇你要注意：吃了暗暗果实，身体元素化之后不能躲避攻击，会吸收所有伤害！
- 4 马歇尔·D·蒂奇吃了震震果实，可以在任意空间引发震动，摧毁目标...！
- 5 马歇尔·D·蒂奇吃了大饼果实，获得大饼铠甲...！
- 6 最强辅助 -- 大饼果实可以将身边事物变成大饼，帮助自己和队友回血...

关于装饰模式就是在原有基础上一层一层进行包装，对于黑胡子的能力也是如此，不论是 **Teach** 类 还是恶魔果实类的子类 **DarkFruit**、**QuakeFruit**、**PieFruit** 它们都是 **Soldier** 类的子类，所以新的恶魔果实对象是可以为旧的恶魔果实附魔的，因为在恶魔果实内部都绑定了一个实体，他就是黑胡子的对象，最终所有恶魔果实的能力都集中在了这个黑胡子对象身上。

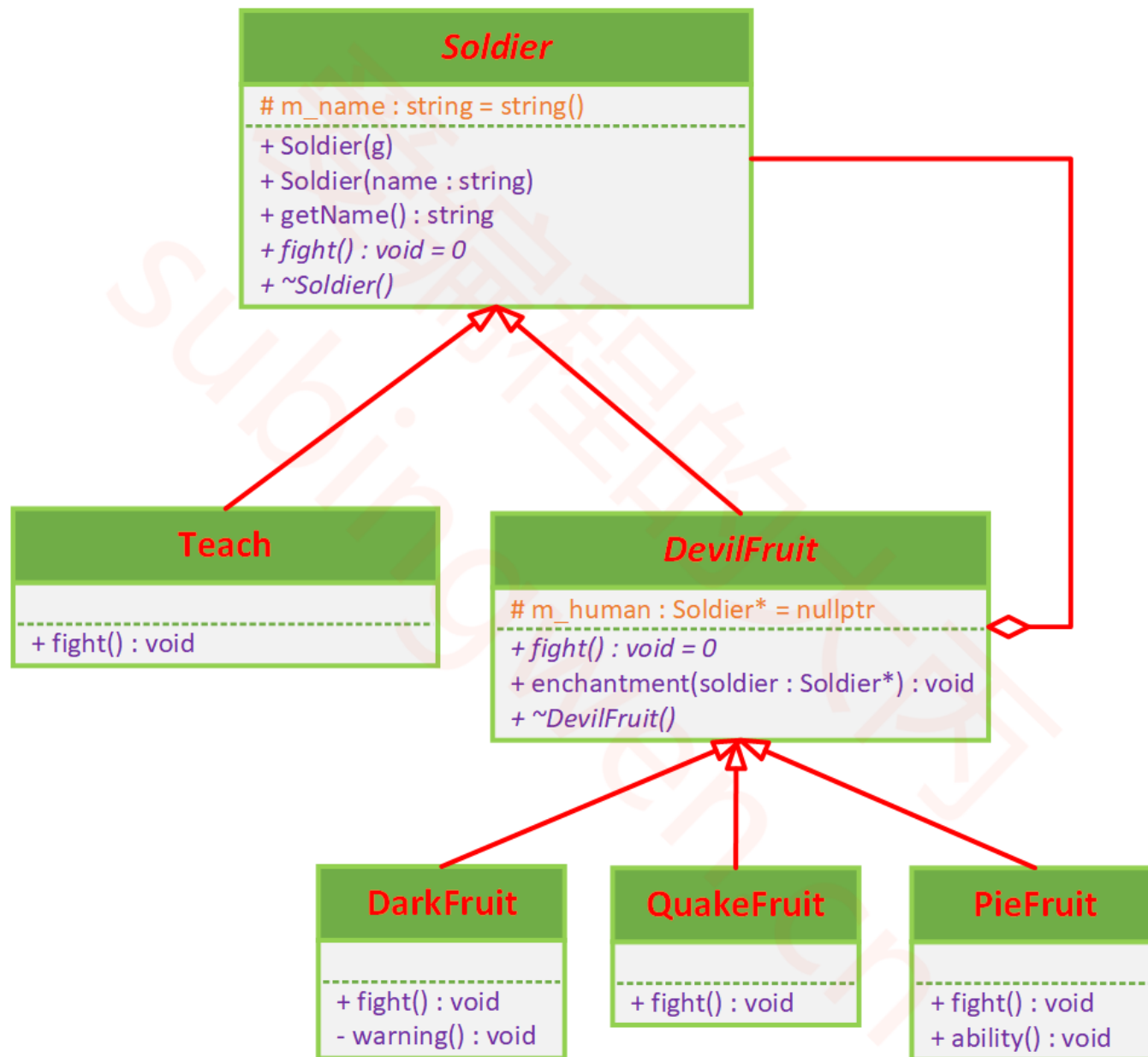
▼ C++ 

```
1 // 给黑胡子附魔
2 dark→enchantment(teach);
3 // 继续附魔
4 quake→enchantment(dark);
```

```
5 // 继续附魔  
6 pie→enchantment(quake);
```

🔗 3. 结构图

最后根据黑胡子吃恶魔果实的这个例子把装饰模式对应的 UML 类图画一下（**等学会了装饰模式之后，需要先画 UML 类图再写程序**）。



恶魔果实类 `DevilFruit` 就是装饰模式中的装饰类的基类，并且恶魔果实类和 父类 `Soldier` 之间还是聚合关系，通过它的派生类 `DarkFruit`、`QuakeFruit`、`PieFruit` 最终实现了对 `Teach` 类的

装饰，使黑胡子这个主体有了三种恶魔果实能力，最终是战力 `fight()` 得到了加成效果。



文章作者: 苏丙楦



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

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
桥接模式 - 大闹天宫

下一篇

设计模式之
组合模式 - 草帽大船团

👍 相关推荐



评论

昵称

邮箱

网址(http://)

来都来了, 说点什么吧...



提交

1 评论



Anonymous

Chrome 94.0.4606.61

Windows 11

2022-10-05

回复

这是先打个桩，后面再写吗

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