

抽象工厂模式 - 弗兰奇一家

📅 发表于 2022-08-31 | ⌚ 更新于 2023-04-06 | 📁 设计模式

| 📄 字数总计: 2.7k | ⌚ 阅读时长: 10 分钟 | 👁 阅读量: 1211 | 💬 评论数: 3



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. 奔向大海

在海贼世界中，位于 **水之都的弗兰奇一家** 是由铁人弗兰奇所领导的以拆船为职业的家族，当然了他们的逆向工程做的也很好，会拆船必然会造船。船是海贼们出海所必备的海上交通工具，它由很多的零件组成，从宏观上看它有这么几个组成部分：**船体**、**动力系统**、**武器**。

有一天我攒够了钱要出海，找到了 **弗兰奇一家**，发现他们的老大跟着草帽路飞出海了，但是我还是选择相信他们的技术。下面是他们给我制定的造船方案，根据我的购买力提供了不同型号的海贼船，一共是三个级别，如下表：

	基础型	标准型	旗舰型
船体	木头	钢铁	合成金属
动力	手动	内燃机	核能
武器	枪	速射炮	激光

根据这个表，在造船的时候需要根据不同的型号选择相应的零部件，在设计程序的时候还需要保证遵循 **开放-封闭原则**，即添加了新型号之后不需要修改原有代码，而是添加新的代码。

1.1 船体

文章134

标签37

分类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. 奔向大海
- 2. 准备生产



最新文章

因为要建造的这艘船是由多个部件组成的并且每个部件还有不同的品级可供选择，先说船体，关于船体材料的这个属性是可变的，所以还需要给它提供一个抽象类，这样在这个抽象类的子类中就可以更换不同的船体材料了：

```
1 // 船体
2 class ShipBody
3 {
4 public:
5     virtual string getShipBody() = 0;
6     virtual ~ShipBody() {}
7 };
8
9 class WoodBody : public ShipBody
10 {
11 public:
12     string getShipBody() override
13     {
14         return string("用<木材>制作轮船船体...");
15     }
16 };
17
18 class IronBody : public ShipBody
19 {
20 public:
21     string getShipBody() override
22     {
23         return string("用<钢铁>制作轮船船体...");
24     }
25 };
```



CMake 保姆级教程
(下)

2023-03-15



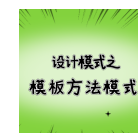
CMake 保姆级教程
(上)

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

```
26
27 class MetalBody : public ShipBody
28 {
29 public:
30     string getShipBody() override
31     {
32         return string("用<合金>制作轮船船体...");
33     }
34 };
```

这样，只要添加了新的造船材料，就给它添加一个对应的子类（父类是 `ShipBody`），在这个子类重写父类的虚函数 `getShipBody()`，用这种材料把船体造出来就行了。

🔗 1.2 动力和武器

知道了如何处理 船体 部分，那么 动力 和 武器 部分的处理思路也是一样的：

- 可以给船提供不同的动力系统，因此这个属性是可变的，所以需要提供一个抽象类
- 可以给船提供不同的武器系统，因此这个属性也是可变的，所以也需要提供一个抽象类

照葫芦画瓢把代码写一下：

▼ C++

```
1 // 动力
2 class Engine
3 {
4 public:
5     virtual string getEngine() = 0;
```



```
6     virtual ~Engine() {}
7 };
8
9 class Human : public Engine
10 {
11 public:
12     string getEngine() override
13     {
14         return string("使用<人力驱动> ...");
15     }
16 };
17
18 class Diesel : public Engine
19 {
20 public:
21     string getEngine() override
22     {
23         return string("使用<内燃机驱动> ...");
24     }
25 };
26
27 class Nuclear : public Engine
28 {
29 public:
30     string getEngine() override
31     {
32         return string("使用<核能驱动> ...");
33     }
34 };
35
36 // 武器
```

```

37  class Weapon
38  {
39  public:
40      virtual string getWeapon() = 0;
41      virtual ~Weapon() {}

```



不论是 **动力** 还是 **武器** 系统都是需要提供一个 **抽象类**，这样它们的子类就可以基于这个 **抽象基类** 进行专门定制，如果要对它们进行拓展也只需添加新的类，不需要修改原有代码。

🔗 1.3 一艘船

如果有了以上的零件，只需要在工厂中将它们装配到一起，这样就得到了一艘船，这是一艘什么型号的船取决于使用的是什麼零件，所以只需要让这艘船对应一个类就可以了，这个类的定义如下：

▼

C++

📄

```

1  // 轮船类
2  class Ship
3  {
4  public:
5      Ship(ShipBody* body, Weapon* weapon, Engine* engine) :
6          m_body(body), m_weapon(weapon), m_engine(engine)
7      {
8      }
9      string getProperty()
10     {
11         string info = m_body->getShipBody() + m_weapon->getWeapon() + m_eng
12         return info;
13     }

```

```
14     ~Ship()  
15     {  
16         delete m_body;  
17         delete m_engine;  
18         delete m_weapon;  
19     }  
20 private:  
21     ShipBody* m_body = nullptr;  
22     Weapon* m_weapon = nullptr;  
23     Engine* m_engine = nullptr;  
24 };  
25
```

这艘船使用的零件是通过构造函数参数传递进来的，并在类的内部对这些零件对象进行了保存，这样在释放船这个对象的时候就可以将相应的零件对象一并析构了。

另外，在 `Ship` 这个类中保存零件对象的时候使用的是它们的父类指针，这样就可以实现多态了。

🔗 2. 准备生产

万事俱备，只剩建厂了。造船厂要生产三种型号的船，那么也就是至少需要三条生产线，所以对应的工厂类也就不止一个，处理思路还是一样的，提供一个抽象的基类，然后在它的子类中完成各种型号的船的组装，每个子类对应的就是一条生产线。

🔗 2.1 设计图纸

现在，关于抽象工厂模式的逻辑应该还是比较清晰了，下面来看一下这个模式对应的 UML 类图：

在这个图中有四个抽象类，分别是：

1. **ShipBody** 类：船体的抽象类

- 有三个子类，在子类中通过不同的材料来建造船体

2. **Weapon** 类：武器的抽象类

- 有三个子类，在子类中给战船提供不同种类的武器

3. **Engine** 类：动力系统抽象类

- 有三个子类，在子类中给战船提供不同动力系统

4. **AbstractFactory** 类：抽象工厂类

- 在子工厂类中生产不同型号的战船
- 和 **ShipBody** 、 **Weapon** 、 **Engine** 有依赖关系，在工厂函数中创建了它们的实例对象
- 和 **Ship** 类有依赖关系，在工厂函数中创建了它的实例对象

关于 **Ship**类 它可以和 **ShipBody** 、 **Weapon** 、 **Engine** 可以是聚合关系，也可以是组合关系：

- 组合关系：析构 **Ship**类 对象的时候，也释放了 **ShipBody** 、 **Weapon** 、 **Engine** 对象
- 聚合关系：析构 **Ship**类 对象的时候，没有释放 **ShipBody** 、 **Weapon** 、 **Engine** 对象

在上面的 **Ship**类的析构函数中做了释放操作，因此在 UML 中将它们之间描述为了组合关系。



在使用抽象工厂模式来处理实际问题的时候，由于实际需求不一样，我们画出的 UML 类图中有些类和类之间的关系可能也会有所不同，所以上图只适用于当前的业务场景，在处

理其他需求的时候还需要具体问题具体分析。

🔗2.2 开工

给上面的程序再添加相应的工厂类，就可以生产出我们需要的型号的船只了，示例代码如下：

```
✓ C++  
  
1  #include <iostream>  
2  #include <string>  
3  using namespace std;  
4  
5  // 船体  
6  class ShipBody  
7  {  
8  public:  
9      virtual string getShipBody() = 0;  
10     virtual ~ShipBody() {}  
11 };  
12  
13 class WoodBody : public ShipBody  
14 {  
15 public:  
16     string getShipBody() override  
17     {  
18         return string("用<木材>制作轮船船体...");  
19     }  
20 };  
21  
22 class IronBody : public ShipBody
```

```
23 {
24 public:
25     string getShipBody() override
26     {
27         return string("用<钢铁>制作轮船船体...");
28     }
29 };
30
31 class MetalBody : public ShipBody
32 {
33 public:
34     string getShipBody() override
35     {
36         return string("用<合金>制作轮船船体...");
37     }
38 };
39
40 // 武器
41 class Weapon
42 {
43 public:
```



在 `main()` 函数中，要通过工厂类的工厂函数生产什么型号的战船，和用户的需求息息相关，所以这个选择也是用户通过客户端的操作界面做出的，在这个例子中，关于客户端的界面操作就直接忽略了。



抽象工厂模式适用于比较复杂的多变的业务场景，总体上就是给一系列功能相同但是属性会发生变化的组件（如：船体材料、武器系统、动力系统）添加一个抽象类，这样就可以

非常方便地进行后续的拓展，再搭配工厂类就可以创建出我们需要的对象了。

关于简单工厂模式、工厂模式和抽象工厂模式的区别可以做如下总结：

1. 简单工厂模式不能遵守 开放-封闭 原则，工厂和抽象工厂模式可以
2. 简单工厂模式只有一个工厂类，工厂和抽象工厂有多个工厂类
3. 工厂模式创建的产品对象相对简单，抽象工厂模式创建的产品对象相对复杂
 - 工厂模式创建的对象对应的类不需要提供抽象类【这产品类组件中没有可变因素】
 - 抽象工厂模式创建的对象对应的类有抽象的基类【这个产品类组件中有可变因素】

文章作者: 苏丙楦



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

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

单例模式 - 白里公

设计模式

单例模式

工厂模式

工厂模式 - 设计模式 - 人造恶魔果实工厂 2

下一篇

相关推荐



评论

昵称

邮箱

网址(http://)

来都来了, 说点什么吧...



提交

3 评论



Anonymous

Chrome 110.0.0.0 Windows 11

2023-04-11

回复

上一篇工厂模式中举的例子, 创建的恶魔果实对象对应的类也提供抽象类呀~ 意思是提不提供抽象类都可以是吗?



Anonymous

Chrome 81.0.4044.145 Android 12

2023-02-13

回复

受益匪浅, 谢谢




Anonymous

Firefox 109.0 Windows 11

2023-01-31

回复

简单易懂，赞！ 

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