

- **简单工厂模式**

简单工厂模式：通过参数控制可以生产任何产品

优点：

1. 简单粗暴，直观易懂。使用一个工厂生产同一结构下的任意产品

缺点：

1. 所有东西在一起生产，产品多代码冗余
2. 不符合开闭原则，新增不是拓展而是修改

```
1 //抽象对象
2 class Fruit
3 {
4 public:
5     virtual void show() = 0;
6 };
7
8 //具体对象Apple
9 class Apple : public Fruit
10 {
11 public:
12     void show() override
13     {
14         std::cout << "我是一个苹果" << std::endl;
15     }
16 };
17
18 //具体对象Banana
19 class Banana : public Fruit
20 {
21 public:
22     void show() override
23     {
24         std::cout << "我是一个香蕉" << std::endl;
25     }
26 };
27
28 //一个水果工厂 负责生产水果
29 class FruitFactory
30 {
31 public:
32     static std::shared_ptr<Fruit> create(const std::string &name)
33     {
34         if (name == "苹果")
35             return std::make_shared<Apple>();
36         else
37             return std::make_shared<Banana>();
38     }
```

```
39  };  
40  
41  int main()  
42  {  
43      std::shared_ptr<Fruit> fruit = FruitFactory::create("苹果");  
44      fruit->show();  
45      fruit = FruitFactory::create("香蕉");  
46      fruit->show();  
47      return 0;  
48  }
```

- **工厂方法模式：**

定义一个创建对象的接口，但是由子类来决定创建哪种对象，使用多个工厂分别生产指定的固定产品

优点：

1. 减轻了工厂类的负担，将某类产品的生产交给指定的工厂来进行
2. 遵循了开闭原则，只拓展不修改

缺点：

1. 对于某种可以形成一组产品族的情况处理较为复杂,需要创建大量的工厂类

```
1 //抽象产品
2 class Fruit
3 {
4 public:
5     virtual void show() = 0;
6 };
7
8 //具体产品 Apple
9 class Apple : public Fruit
10 {
11 public:
12     void show() override
13     {
14         std::cout << "我是一个苹果" << std::endl;
15     }
16 };
17
18 //具体产品 Banana
19 class Banana : public Fruit
20 {
21 public:
22     void show() override
23     {
24         std::cout << "我是一个香蕉" << std::endl;
25     }
26 };
27
28 //抽象工厂
29 class FruitFactory
30 {
31 public:
32     virtual std::shared_ptr<Fruit> create() = 0;
33 };
34
35 //具体Apple工厂
36 class AppleFactory : public FruitFactory
37 {
38     std::shared_ptr<Fruit> create() override
```

```

39     {
40         return std::make_shared<Apple>();
41     }
42 };
43
44 //具体Banana工厂
45 class BananaFactory : public FruitFactory
46 {
47     std::shared_ptr<Fruit> create() override
48     {
49         return std::make_shared<Banana>();
50     }
51 };
52
53 int main()
54 {
55     std::shared_ptr<FruitFactory> factory(new AppleFactory());
56     std::shared_ptr<Fruit> fruit = factory->create();
57     fruit->show();
58     factory.reset(new BananaFactory());
59     fruit = factory->create();
60     fruit->show();
61     return 0;
62 }
63

```

抽象工厂模式：

围绕一个超级工厂创建其他工厂。每个生成的工厂按照工厂模式提供对象

思想：将工厂抽象成两层，抽象工厂 & 具体工厂子类，在工厂子类种生产不同类型的子产品

```
1 // 抽象产品
2 class Fruit
3 {
4 public:
5     virtual void show() = 0;
6 };
7
8 // 具体产品
9 class Apple : public Fruit
10 {
11 public:
12     void show() override
13     {
14         std::cout << "我是一个苹果" << std::endl;
15     }
16 };
17
18 class Banana : public Fruit
19 {
20 public:
21     void show() override
22     {
23         std::cout << "我是一个香蕉" << std::endl;
24     }
25 };
26
27 // 抽象产品
28 class Animal
29 {
30 public:
31     virtual void voice() = 0;
32 };
33
34 // 具体产品
35 class Lamp : public Animal
36 {
37 public:
38     void voice() { std::cout << "咩咩咩\n"; }
```

```
39 };
40 class Dog : public Animal
41 {
42 public:
43     void voice() { std::cout << "汪汪汪\n"; }
44 };
45
46 // 抽象工厂
47 class Factory
48 {
49 public:
50     virtual std::shared_ptr<Fruit> getFruit(const std::string &name) = 0;
51     virtual std::shared_ptr<Animal> getAnimal(const std::string &name) = 0;
52 };
53
54 // 具体工厂
55 class FruitFactory : public Factory
56 {
57 public:
58     virtual std::shared_ptr<Animal> getAnimal(const std::string &name)
59     {
60         return std::shared_ptr<Animal>();
61     }
62     virtual std::shared_ptr<Fruit> getFruit(const std::string &name)
63     {
64         if (name == "苹果")
65         {
66             return std::make_shared<Apple>();
67         }
68         else if (name == "香蕉")
69         {
70             return std::make_shared<Banana>();
71         }
72         return std::shared_ptr<Fruit>();
73     }
74 };
75
76 // 具体工厂
77 class AnimalFactory : public Factory
```

```
78 {
79 public:
80     virtual std::shared_ptr<Fruit> getFruit(const std::string &name)
81     {
82         return std::shared_ptr<Fruit>();
83     }
84
85     virtual std::shared_ptr<Animal> getAnimal(const std::string &name)
86     {
87         if (name == "小羊")
88         {
89             return std::make_shared<Lamp>();
90         }
91         else if (name == "小狗")
92         {
93             return std::make_shared<Dog>();
94         }
95         return std::shared_ptr<Animal>();
96     }
97 };
98
99 //制造工厂
100 class FactoryProducer
101 {
102 public:
103     static std::shared_ptr<Factory> getFactory(const std::string &name)
104     {
105         if (name == "动物")
106         {
107             return std::make_shared<AnimalFactory>();
108         }
109         else
110         {
111             return std::make_shared<FruitFactory>();
112         }
113     }
114 };
115 int main()
116 {
```



```
117     std::shared_ptr<Factory> fruit_factory = FactoryProducer::getFactory("水果");
118     std::shared_ptr<Fruit> fruit = fruit_factory->getFruit("苹果");
119     fruit->show();
120     fruit = fruit_factory->getFruit("香蕉");
121     fruit->show();
122     std::shared_ptr<Factory> animal_factory = FactoryProducer::getFactory("动物");
123     std::shared_ptr<Animal> animal = animal_factory->getAnimal("小羊");
124     animal->voice();
125     animal = animal_factory->getAnimal("小狗");
126     animal->voice();
127     return 0;
128 }
```