- **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

```cpp
//抽象对象
class Fruit
{
public:
    virtual void show() = 0;
};

//具体对象Apple
class Apple : public Fruit
{
public:
    void show() override
    {
        std::cout << "我是一个苹果" << std::endl;
    }
};

//具体对象Banana
class Banana : public Fruit
{
public:
    void show() override
    {
        std::cout << "我是一个香蕉" << std::endl;
    }
};

//一个水果工厂 负责生产水果
class FruitFactory
{
public:
    static std::shared_ptr<Fruit> create(const std::string &name)
    {
        if (name == "苹果")
            return std::make_shared<Apple>();
        else
            return std::make_shared<Banana>();
    }
};

int main()
{
    std::shared_ptr<Fruit> fruit = FruitFactory::create("苹果");
    fruit->show();
    fruit = FruitFactory::create("香蕉");
    fruit->show();
    return 0;
}
```

- **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

```cpp
//抽象产品
class Fruit
{
public:
    virtual void show() = 0;
};

//具体产品 Apple
class Apple : public Fruit
{
public:
    void show() override
    {
        std::cout << "我是一个苹果" << std::endl;
    }
};

//具体产品 Banana
class Banana : public Fruit
{
public:
    void show() override
    {
        std::cout << "我是一个香蕉" << std::endl;
    }
};

//抽象工厂
class FruitFactory
{
public:
    virtual std::shared_ptr<Fruit> create() = 0;
};

//具体Apple工厂
class AppleFactory : public FruitFactory
{
    std::shared_ptr<Fruit> create() override
    {
        return std::make_shared<Apple>();
    }
};

//具体Banana工厂
class BananaFactory : public FruitFactory
{
    std::shared_ptr<Fruit> create() override
    {
        return std::make_shared<Banana>();
    }
};

int main()
{
    std::shared_ptr<FruitFactory> factory(new AppleFactory());
    std::shared_ptr<Fruit>  fruit = factory->create();
    fruit->show();
    factory.reset(new BananaFactory());
    fruit = factory->create();
    fruit->show();
    return 0;
}

```

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

```cpp
// 抽象产品
class Fruit
{
public:
    virtual void show() = 0;
};

// 具体产品
class Apple : public Fruit
{
public:
    void show() override
    {
        std::cout << "我是一个苹果" << std::endl;
    }
};

class Banana : public Fruit
{
public:
    void show() override
    {
        std::cout << "我是一个香蕉" << std::endl;
    }
};

// 抽象产品
class Animal
{
public:
    virtual void voice() = 0;
};

// 具体产品
class Lamp : public Animal
{
public:
    void voice() { std::cout << "咩咩咩\n"; }
};
class Dog : public Animal
{
public:
    void voice() { std::cout << "汪汪汪\n"; }
};

// 抽象工厂
class Factory
{
public:
    virtual std::shared_ptr<Fruit> getFruit(const std::string &name) = 0;
    virtual std::shared_ptr<Animal> getAnimal(const std::string &name) = 0;
};

// 具体工厂
class FruitFactory : public Factory
{
public:
    virtual std::shared_ptr<Animal> getAnimal(const std::string &name)
    {
        return std::shared_ptr<Animal>();
    }
    virtual std::shared_ptr<Fruit> getFruit(const std::string &name)
    {
        if (name == "苹果")
        {
            return std::make_shared<Apple>();
        }
        else if (name == "香蕉")
        {
            return std::make_shared<Banana>();
        }
        return std::shared_ptr<Fruit>();
    }
};

// 具体工厂
class AnimalFactory : public Factory
```

```cpp
{
public:
    virtual std::shared_ptr<Fruit> getFruit(const std::string &name)
    {
        return std::shared_ptr<Fruit>();
    }

    virtual std::shared_ptr<Animal> getAnimal(const std::string &name)
    {
        if (name == "小羊")
        {
            return std::make_shared<Lamp>();
        }
        else if (name == "小狗")
        {
            return std::make_shared<Dog>();
        }
        return std::shared_ptr<Animal>();
    }
};

//制造工厂
class FactoryProducer
{
public:
    static std::shared_ptr<Factory> getFactory(const std::string &name)
    {
        if (name == "动物")
        {
            return std::make_shared<AnimalFactory>();
        }
        else
        {
            return std::make_shared<FruitFactory>();
        }
    }
};
int main()
{
    std::shared_ptr<Factory> fruit_factory = FactoryProducer::getFactory("水果");
    std::shared_ptr<Fruit> fruit = fruit_factory->getFruit("苹果");
    fruit->show();
    fruit = fruit_factory->getFruit("香蕉");
    fruit->show();
    std::shared_ptr<Factory> animal_factory = FactoryProducer::getFactory("动物");
    std::shared_ptr<Animal> animal = animal_factory->getAnimal("小羊");
    animal->voice();
    animal = animal_factory->getAnimal("小狗");
    animal->voice();
    return 0;
}
```