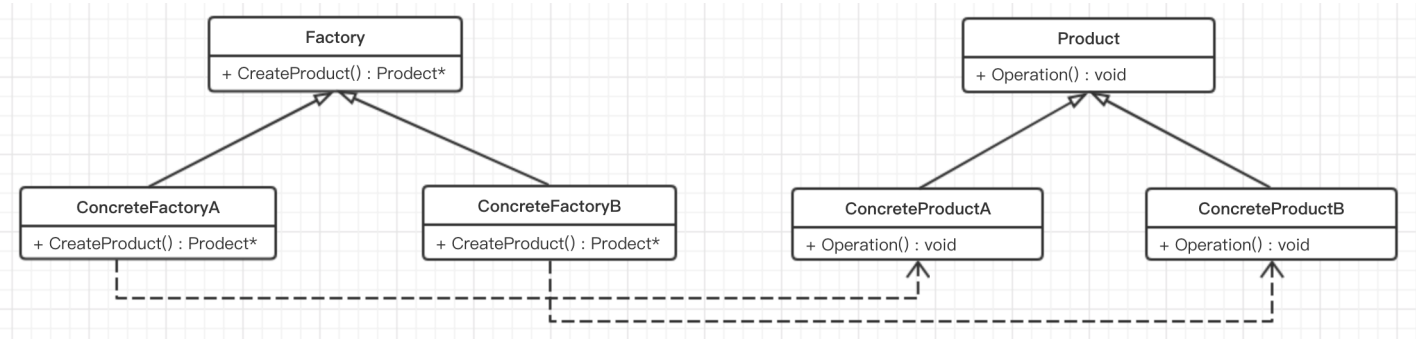
工厂方法模式

资料 C++工厂方法模式

01.简述

工厂方法模式 是一种常用的对象创建型设计模式，此模式的核心思想是封装类中不变的部分，提取其中个性化善变的部分为独立类，通过依赖注入以达到解耦、复用以及方便后期维护扩展的目的。

02.UML结构图



- Factory（抽象工厂）：是工厂方法的核心，定义了工厂类共有的接口
- ConcreteFactory（具体工厂）：实现抽象工厂接口的具体工厂，负责创建具体的产品
- Product（抽象产品）：所创建产品的基类，定义了产品类共有的接口
- ConcreteProduct（具体产品）：实现抽象产品类定义的接口，具体工厂负责创建具体产品

03.优缺点

优点：

- 克服了简单工厂模式违背开放-封闭原则的缺点（每添加一个产品的时候就需要修改工厂类的创建接口），又保留了封装对象创建过程的优点，降低客户端和工厂的耦合性。“工厂方法模式”是“简单工厂模式”的进一步抽象

缺点：

- 每增加一个产品，相应的就要增加一个子工厂类，加大了额外的开发量

04.使用场景

- 对于某个产品，调用者清楚知道应该使用哪个具体工厂服务，然后实例化该具体工厂，生产具体产品
- 只是需要一种产品，但是具体由哪个工厂生产的由调用者决定

05.实现代码

```
//
//  T3_20190123.h
//  DesignPattern
//
//  Created by shadot on 2019/1/23.
//  Copyright © 2019 shadot. All rights reserved.
//

#ifndef T3_20190123_h
#define T3_20190123_h

//C++工厂方法模式

#include <iostream>

using namespace std;

class ICar{
public:
    virtual string Name() = 0;
};

class ConcreteCarA : public ICar{
public:
    string Name(){
        return "CarA";
    }
};

class ConcreteCarB : public ICar{
public:
    string Name(){
        return "CarB";
    }
};

class IFactory{
public:
    virtual ICar* CreateCar() = 0;
};
```

```
41
42 class ConcreteFactoryA : public IFactory{
43 public:
44     ICar* CreateCar(){
45         return new ConcreteCarA();
46     }
47 };
48
49 class ConcreteFactoryB : public IFactory{
50 public:
51     ICar* CreateCar(){
52         return new ConcreteCarB();
53     }
54 };
55
56 #endif /* T3_20190123_h */
57
58     //20190123工厂方法模式
59     {
60         IFactory* pFactoryA = new ConcreteFactoryA();
61         ICar* pCarA = pFactoryA->CreateCar();
62         cout << pCarA->Name() << endl;
63
64         IFactory* pFactoryB = new ConcreteFactoryB();
65         ICar* pCarB = pFactoryB->CreateCar();
66         cout << pCarB->Name() << endl;
67     }
68
69
```