設計模式C++實現（1）——工廠模式

星期六, 2013 12月 14, 12:57 上午

軟件領域中的設計模式為開發人員提供了一種使用專家設計經驗的有效途徑。設計模式中運用了面向對象編程語言的重要特性：封裝、繼承、多態，真正領悟設計模式的精髓是可能一個漫長的過程，需要大量實踐經驗的積累。最近看設計模式的書，對於每個模式，用C++寫了個小例子，加深一下理解。主要參考《大話設計模式》和《設計模式:可復用面向對象軟件的基礎》兩本書。本文介紹工廠模式的實現。

工廠模式屬於創建型模式，大致可以分為三類，簡單工廠模式、工廠方法模式、抽象工廠模式。聽上去差不多，都是工廠模式。下面一個個介紹，首先介紹簡單工廠模式，它的主要特點是需要在工廠類中做判斷，從而創造相應的產品。當增加新的產品時，就需要修改工廠類。有點抽象，舉個例子就明白了。有一家生產處理器核的廠家，它只有一個工廠，能夠生產兩種型號的處理器核。客戶需要什麼樣的處理器核，一定要顯示地告訴生產工廠。下面給出一種實現方案。

```
enum CTYPE {COREA, COREB};
class SingleCore
{
public:
    virtual void Show() = 0;
};
//單核A
class SingleCoreA: public SingleCore
{
public:
    void Show() { cout<<"SingleCore A"<<endl; }
};
//單核B
class SingleCoreB: public SingleCore
{
public:
    void Show() { cout<<"SingleCore B"<<endl; }
};
//唯一的工廠，可以生產兩種型號的處理器核，在內部判斷
class Factory
{
public:
    SingleCore* CreateSingleCore(enum CTYPE ctype)
    {
        if(ctype == COREA) //工廠內部判斷
            return new SingleCoreA(); //生產核A
        else if(ctype == COREB)
            return new SingleCoreB(); //生產核B
        else
            return NULL;
    }
};
```

這樣設計的主要缺點之前也提到過，就是要增加新的核類型時，就需要修改工廠類。這就違反了開放封閉原則：軟件實體（類、模塊、函數）可以擴展，但是不可修改。於是，工廠方法模式出現了。所謂工廠方法模式，是指定義一個用於創建對象的接口，讓子類決定實例化哪

一個類。Factory Method使一個類的實例化延遲到其子類。

聽起來很抽象，還是以剛才的例子解釋。這家生產處理器核的產家賺了不少錢，於是決定再開設一個工廠專門用來生產B型號的單核，而原來的工廠專門用來生產A型號的單核。這時，客戶要做的是找好工廠，比如要A型號的核，就找A工廠要；否則找B工廠要，不再需要告訴工廠具體要什麼型號的處理器核了。下面給出一個實現方案。

```
1. class SingleCore
2. {
3. public:
4.     virtual void Show() = 0;
5. };
6. //單核A
7. class SingleCoreA: public SingleCore
8. {
9. public:
10.     void Show() { cout<<"SingleCore A"<<endl; }
11. };
12. //單核B
13. class SingleCoreB: public SingleCore
14. {
15. public:
16.     void Show() { cout<<"SingleCore B"<<endl; }
17. };
18. class Factory
19. {
20. public:
21.     virtual SingleCore* CreateSingleCore() = 0;
22. };
23. //生產A核的工廠
24. class FactoryA: public Factory
25. {
26. public:
27.     SingleCoreA* CreateSingleCore() { return new SingleCoreA; }
28. };
29. //生產B核的工廠
30. class FactoryB: public Factory
31. {
32. public:
33.     SingleCoreB* CreateSingleCore() { return new SingleCoreB; }
34. };
```

工廠方法模式也有缺點，每增加一種產品，就需要增加一個對象的工廠。如果這家公司發展迅速，推出了很多新的處理器核，那麼就要開設相應的新工廠。在C++實現中，就是要定義一個個的工廠類。顯然，相比簡單工廠模式，工廠方法模式需要更多的類定義。

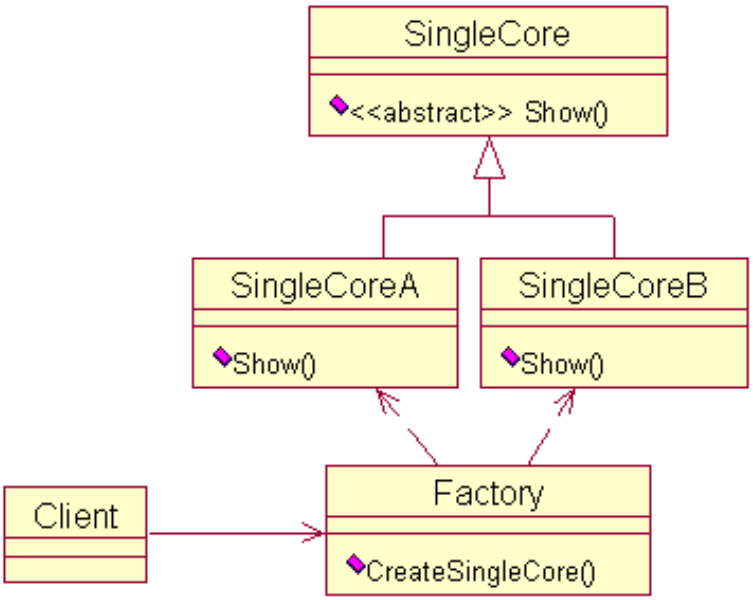
既然有了簡單工廠模式和工廠方法模式，為什麼還要有抽象工廠模式呢？它到底有什麼作用呢？還是舉這個例子，這家公司的技術不斷進步，不僅可以生產單核處理器，也能生產多核處理器。現在簡單工廠模式和工廠方法模式都鞭長莫及。抽象工廠模式登場了。它的定義為提供一個創建一系列相關或相互依賴對象的接口，而無需指定它們具體的類。具體這樣應用，這家公司還是開設兩個工廠，一個專門用來生產A型號的單核多核處理器，而另一個工廠專門用來生產B型號的單核多核處理器，下面給出實現的代碼。

```
1. //單核
2. class SingleCore
```

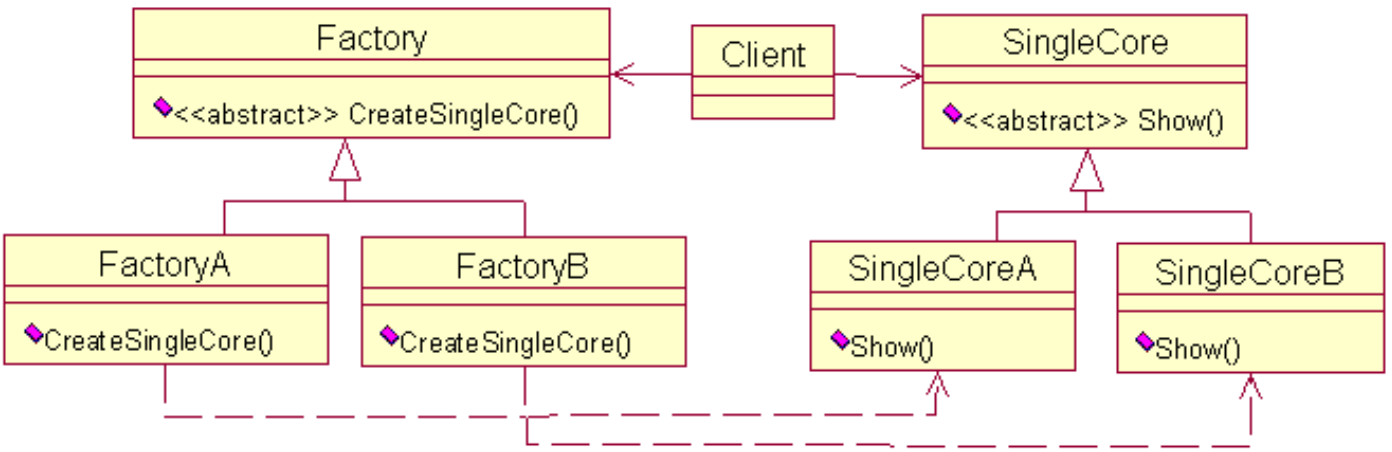
```
{
public:
    virtual void Show() = 0;
};
class SingleCoreA: public SingleCore
{
public:
    void Show() { cout<<"Single Core A"<<endl; }
};
class SingleCoreB :public SingleCore
{
public:
    void Show() { cout<<"Single Core B"<<endl; }
};
//多核
class MultiCore
{
public:
    virtual void Show() = 0;
};
class MultiCoreA : public MultiCore
{
public:
    void Show() { cout<<"Multi Core A"<<endl; }

};
class MultiCoreB : public MultiCore
{
public:
    void Show() { cout<<"Multi Core B"<<endl; }
};
//工廠
class CoreFactory
{
public:
    virtual SingleCore* CreateSingleCore() = 0;
    virtual MultiCore* CreateMultiCore() = 0;
};
//工廠A，專門用來生產A型號的處理器
class FactoryA :public CoreFactory
{
public:
    SingleCore* CreateSingleCore() { return new SingleCoreA(); }
    MultiCore* CreateMultiCore() { return new MultiCoreA(); }
};
//工廠B，專門用來生產B型號的處理器
class FactoryB : public CoreFactory
{
public:
    SingleCore* CreateSingleCore() { return new SingleCoreB(); }
    MultiCore* CreateMultiCore() { return new MultiCoreB(); }
};
```

至此，工廠模式介紹完了。利用Rational Rose 2003軟件，給出三種工廠模式的UML圖，加深印象。

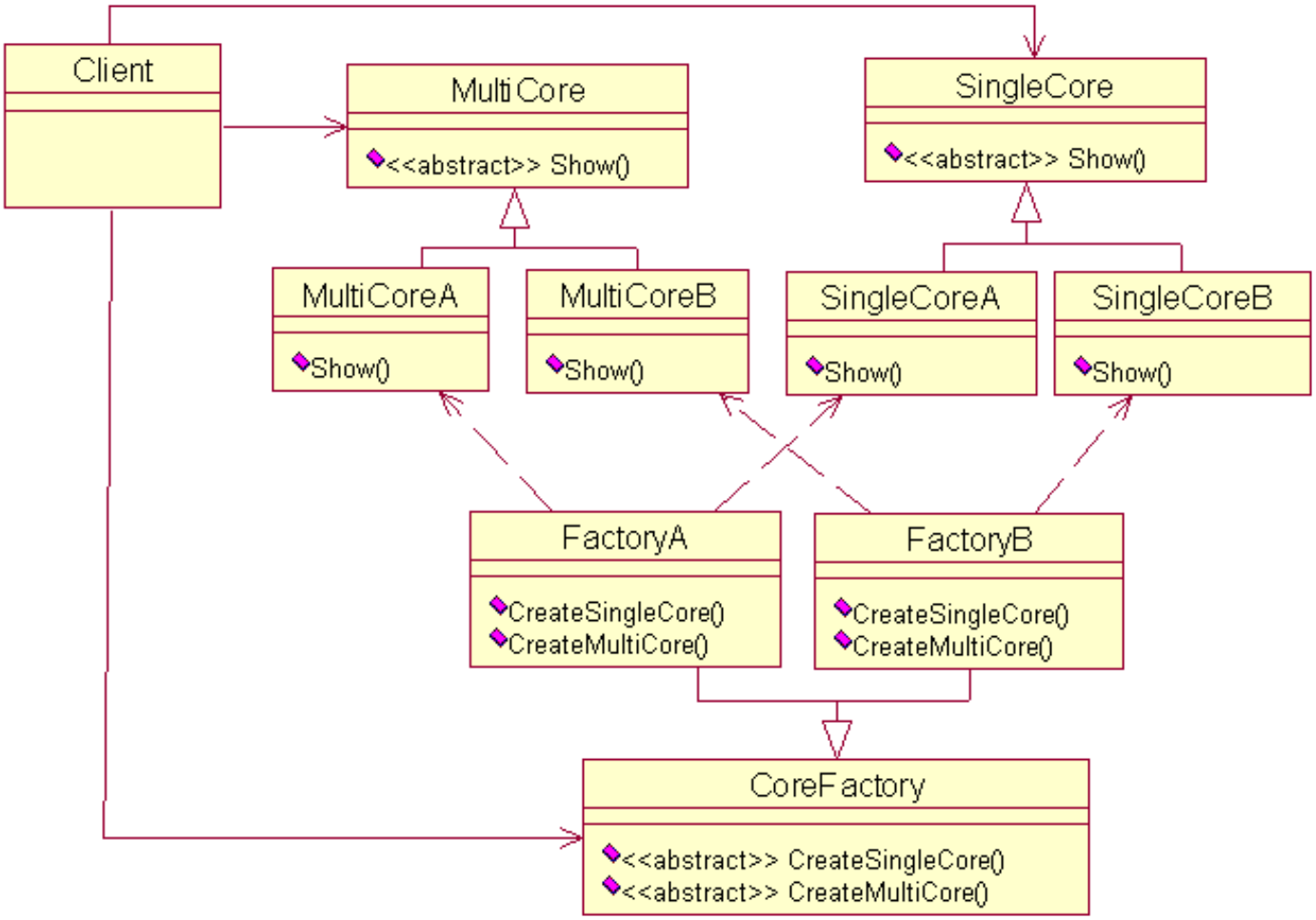
簡單工廠模式的UML圖：



工廠方法的UML圖：



抽象工廠模式的UML圖：


Client
MultiCore
<<abstract>> Show()
SingleCore
<<abstract>> Show()
MultiCoreA
Show()
MultiCoreB
Show()
SingleCoreA
Show()
SingleCoreB
Show()
FactoryA
CreateSingleCore()
CreateMultiCore()
FactoryB
CreateSingleCore()
CreateMultiCore()
CoreFactory
<<abstract>> CreateSingleCore()
<<abstract>> CreateMultiCore()