

3. 抽象工厂模式(Abstract Factory)

目录

- 抽象工厂模式(Abstract Factory)
 - 模式动机
 - 模式定义
 - 模式结构
 - 时序图
 - 代码分析
 - 模式分析
 - 实例
 - 优点
 - 缺点
 - 适用环境
 - 模式应用
 - 模式扩展
 - “开闭原则”的倾斜性
 - 工厂模式的退化
 - 总结

3.1. 模式动机

- 在工厂方法模式中具体工厂负责生产具体的产品，每一个具体工厂对应一种具体产品，工厂方法也具有唯一性，一般情况下，一个具体工厂中只有一个工厂方法或者一组重载的工厂方法。但是有时候我们需要一个工厂可以提供多个产品对象，而不是单一的产品对象。

为了更清晰地理解工厂方法模式，需要先引入两个概念：

- 产品等级结构：产品等级结构即产品的继承结构，如一个抽象类是电视机，其子类有海尔电视机、海信电视机、TCL电视机，则抽象电视机与具体品牌的电视机之间构成了一个产品等级结构，抽象电视机是父类，而具体品牌的电视机是其子类。
 - 产品族：在抽象工厂模式中，产品族是指由同一个工厂生产的，位于不同产品等级结构中的一组产品，如海尔电器工厂生产的海尔电视机、海尔电冰箱，海尔电视机位于电视机产品等级结构中，海尔电冰箱位于电冰箱产品等级结构中。
- 当系统所提供的工厂所需生产的具体产品并不是一个简单的对象，而是多个位于不同产品等级结构中属于不同类型的产品时需要使用抽象工厂模式。
 - 抽象工厂模式是所有形式的工厂模式中最为抽象和最具一般性的一种形态。

- 抽象工厂模式与工厂方法模式最大的区别在于，工厂方法模式针对的是一个产品等级结构，而抽象工厂模式则需要面对多个产品等级结构，一个工厂等级结构可以负责多个不同产品等级结构中的产品对象的创建。当一个工厂等级结构可以创建出分属于不同产品等级结构的一个产品族中的所有对象时，抽象工厂模式比工厂方法模式更为简单、有效率。

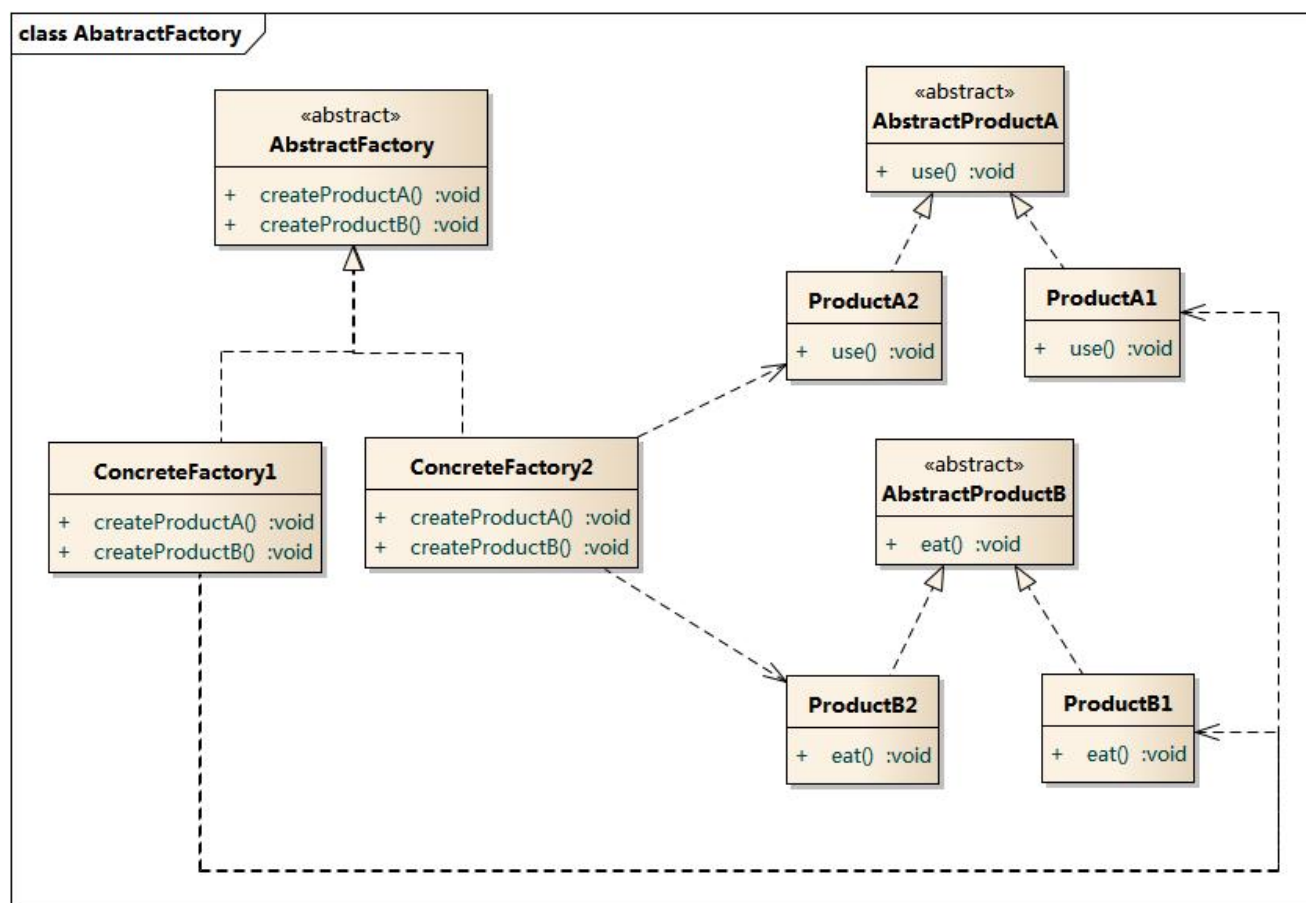
3.2. 模式定义

抽象工厂模式(Abstract Factory Pattern): 提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。抽象工厂模式又称为Kit模式，属于对象创建型模式。

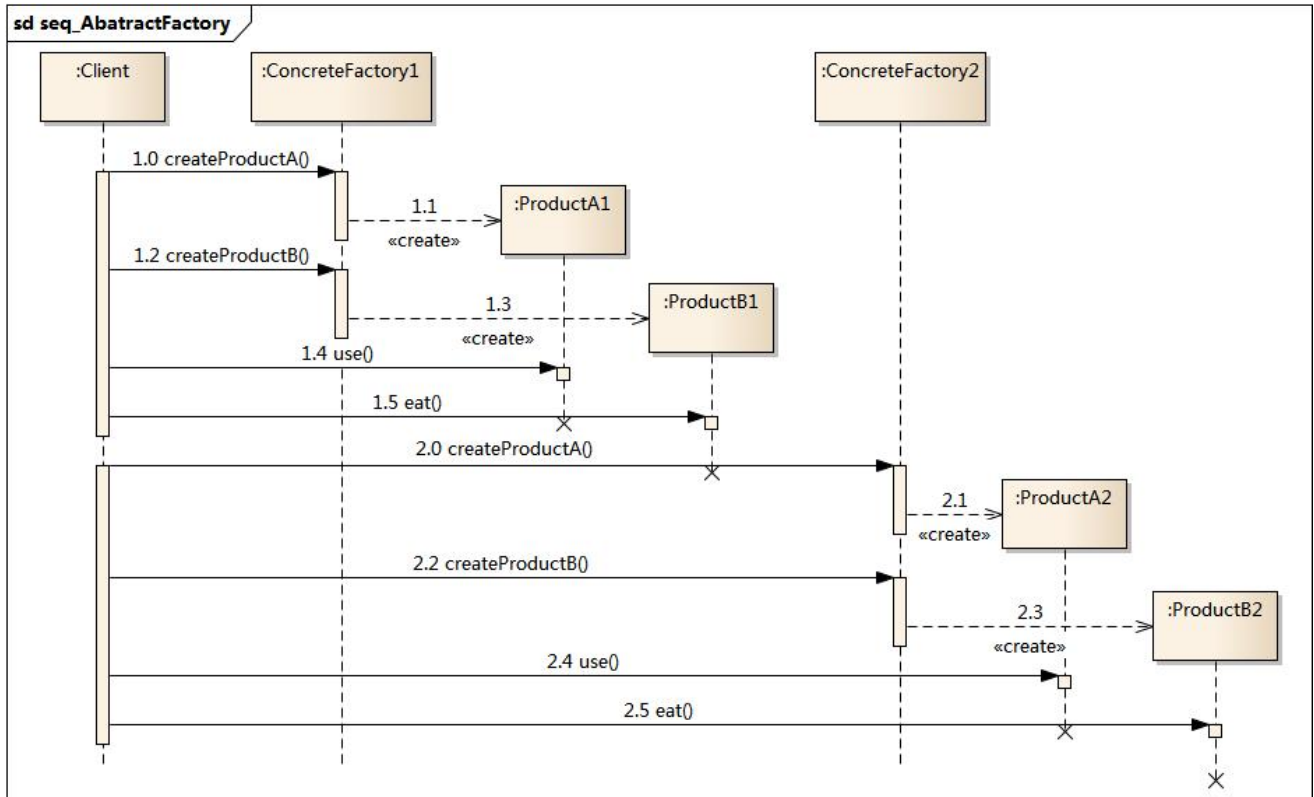
3.3. 模式结构

抽象工厂模式包含如下角色：

- AbstractFactory：抽象工厂
- ConcreteFactory：具体工厂
- AbstractProduct：抽象产品
- Product：具体产品



3.4. 时序图



3.5. 代码分析

```

1  #include <iostream>
2  #include "AbstractFactory.h"
3  #include "AbstractProductA.h"
4  #include "AbstractProductB.h"
5  #include "ConcreteFactory1.h"
6  #include "ConcreteFactory2.h"
7  using namespace std;
8
9  int main(int argc, char *argv[])
10 {
11     AbstractFactory * fc = new ConcreteFactory1();
12     AbstractProductA * pa = fc->createProductA();
13     AbstractProductB * pb = fc->createProductB();
14     pa->use();
15     pb->eat();
16
17     AbstractFactory * fc2 = new ConcreteFactory2();
18     AbstractProductA * pa2 = fc2->createProductA();
19     AbstractProductB * pb2 = fc2->createProductB();
20     pa2->use();
21     pb2->eat();
22 }

```

```

1  //////////////////////////////////////
2  //  ConcreteFactory1.cpp
3  //  Implementation of the Class ConcreteFactory1
4  //  Created on:      02-十月-2014 15:04:11
5  //  Original author: colin
6  //////////////////////////////////////
7  #include "ConcreteFactory1.h"
8  #include "ProductA1.h"
9  #include "ProductB1.h"
10 AbstractProductA * ConcreteFactory1::createProductA(){
11     return new ProductA1();
12 }
13
14 AbstractProductB * ConcreteFactory1::createProductB(){
15     return new ProductB1();
16 }
17
18

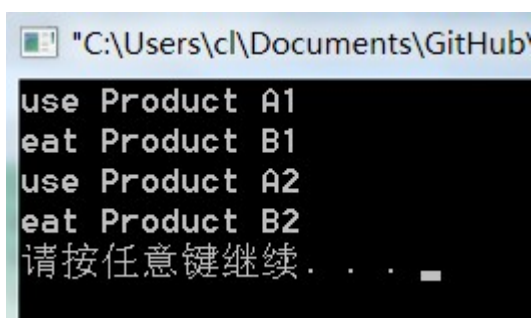
```

```

1  //////////////////////////////////////
2  //  ProductA1.cpp
3  //  Implementation of the Class ProductA1
4  //  Created on:      02-十月-2014 15:04:17
5  //  Original author: colin
6  //////////////////////////////////////
7  #include "ProductA1.h"
8  #include <iostream>
9  using namespace std;
10 void ProductA1::use(){
11     cout << "use Product A1" << endl;
12 }
13

```

运行结果：



```

"C:\Users\cl\Documents\GitHub"
use Product A1
eat Product B1
use Product A2
eat Product B2
请按任意键继续. . .

```

3.6. 模式分析

3.7. 实例

3.8. 优点

- 抽象工厂模式隔离了具体类的生成，使得客户并不需要知道什么被创建。由于这种隔离，更换一个具体工厂就变得相对容易。所有的具体工厂都实现了抽象工厂中定义的那些公共接口，因此只需改变具体工厂的实例，就可以在某种程度上改变整个软件系统的行为。另外，应用抽象工厂模式可以实现高内聚低耦合的设计目的，因此抽象工厂模式得到了广泛的应用。
- 当一个产品族中的多个对象被设计成一起工作时，它能够保证客户端始终只使用同一个产品族中的对象。这对一些需要根据当前环境来决定其行为的软件系统来说，是一种非常实用的设计模式。
- 增加新的具体工厂和产品族很方便，无须修改已有系统，符合“开闭原则”。

3.9. 缺点

- 在添加新的产品对象时，难以扩展抽象工厂来生产新种类的产品，这是因为在抽象工厂角色中规定了所有可能被创建的产品集合，要支持新种类的产品就意味着要对该接口进行扩展，而这将涉及到对抽象工厂角色及其所有子类的修改，显然会带来较大的不便。
- 开闭原则的倾斜性（增加新的工厂和产品族容易，增加新的产品等级结构麻烦）。

3.10. 适用环境

在以下情况下可以使用抽象工厂模式：

- 一个系统不应当依赖于产品类实例如何被创建、组合和表达的细节，这对于所有类型的工厂模式都是重要的。
- 系统中有多于一个的产品族，而每次只使用其中某一产品族。
- 属于同一个产品族的产品将在一起使用，这一约束必须在系统的设计中体现出来。
- 系统提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于具体实现。

3.11. 模式应用

在很多软件系统中需要更换界面主题，要求界面中的按钮、文本框、背景色等一起发生改变时，可以使用抽象工厂模式进行设计。

3.12. 模式扩展

“开闭原则”的倾斜性

- “开闭原则”要求系统对扩展开放，对修改封闭，通过扩展达到增强其功能的目的。对于涉及到多个产品族与多个产品等级结构的系统，其功能增强包括两方面：
 1. 增加产品族：对于增加新的产品族，工厂方法模式很好的支持了“开闭原则”，对于新增加的产品族，只需要对应增加一个新的具体工厂即可，对已有代码无须做任何修改。

2. 增加新的产品等级结构：对于增加新的产品等级结构，需要修改所有的工厂角色，包括抽象工厂类，在所有的工厂类中都需要增加生产新产品的方法，不能很好地支持“开闭原则”。

- 抽象工厂模式的这种性质称为“开闭原则”的倾斜性，抽象工厂模式以一种倾斜的方式支持增加新的产品，它为新产品族的增加提供方便，但不能为新的产品等级结构的增加提供这样的方便。

工厂模式的退化

- 当抽象工厂模式中每一个具体工厂类只创建一个产品对象，也就是只存在一个产品等级结构时，抽象工厂模式退化成工厂方法模式；当工厂方法模式中抽象工厂与具体工厂合并，提供一个统一的工厂来创建产品对象，并将创建对象的工厂方法设计为静态方法时，工厂方法模式退化成简单工厂模式。

3.13. 总结

- 抽象工厂模式提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。抽象工厂模式又称为Kit模式，属于对象创建型模式。
- 抽象工厂模式包含四个角色：抽象工厂用于声明生成抽象产品的方法；具体工厂实现了抽象工厂声明的生成抽象产品的方法，生成一组具体产品，这些产品构成了一个产品族，每一个产品都位于某个产品等级结构中；抽象产品为每种产品声明接口，在抽象产品中定义了产品的抽象业务方法；具体产品定义具体工厂生产的具体产品对象，实现抽象产品接口中定义的业务方法。
- 抽象工厂模式是所有形式的工厂模式中最为抽象和最具一般性的一种形态。抽象工厂模式与工厂方法模式最大的区别在于，工厂方法模式针对的是一个产品等级结构，而抽象工厂模式则需要面对多个产品等级结构。
- 抽象工厂模式的主要优点是隔离了具体类的生成，使得客户并不需要知道什么被创建，而且每次可以通过具体工厂类创建一个产品族中的多个对象，增加或者替换产品族比较方便，增加新的具体工厂和产品族很方便；主要缺点在于增加新的产品等级结构很复杂，需要修改抽象工厂和所有的具体工厂类，对“开闭原则”的支持呈现倾斜性。
- 抽象工厂模式适用情况包括：一个系统不应当依赖于产品类实例如何被创建、组合和表达的细节；系统中有多于一个的产品族，而每次只使用其中某一产品族；属于同一个产品族的产品将在一起使用；系统提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于具体实现。