文章编号：2096-1472(2017)-05-18-05

C++11实现可变参数泛型抽象工厂

闵 军，罗 泓
（宜宾学院，四川 宜宾 644000）

摘 要：由于抽象工厂模式有利于达到高内聚低耦合的设计目的，因此在软件设计中得到广泛应用。但抽象工厂模式的传统实现方式在处理具体产品构造函数参数不同、异类组合、具体产品数量繁多的情况时，都显得很繁琐、很困难、复用性很低。针对这一问题，本文以抽象工厂模式为例，应用C++11新标准和泛型编程技术，提出一种C++11可变参数泛型抽象工厂的实现方式。实验结果表明，该方式比传统实现方式更为简洁高效、复用性更强，优雅地实现了对产品类型可变、参数可变、异类组合的支持。该实现方式及代码具有实用性，可应用到软件项目中。

关键词：C++11；泛型；可变参数；抽象工厂；设计模式
中图分类号：TP311.1 **文献标识码**：A

The Implementation of the Variable Parameter Generic Abstract Factory in C++11

MIN Jun,LUO Hong
(*Yibin University*,*Yibin* 644000,*China*)

Abstract: As the abstract factory pattern is conducive to the achievement of high cohesion and low coupling design,it has been widely applied in software design.But the traditional implementation models of the abstract factory design pattern are cumbersome and difficult with poor reusability,when dealing with the concrete product constructor of different parameters,heterogeneous combination and miscellaneous concrete products.Focusing on the problem and taking the abstract factory as an example,the paper adopts the new C++11 standards and generic programming technology to present an implementation model of the C++11 variable parameter generic abstract factory.Experimental results show that this model is more concise and more efficient with better reusability than the traditional implementation models.This model gracefully implements the support for variable product types,variable parameters and heterogeneous combination.With good practicability,the implementation model and its code can be applied in software projects.

Keywords:C++11;generic;variable parameter;abstract factory;design pattern

1 引言(Introduction)

一种设计模式就是解决某一类软件设计问题的可复用的解决方案。设计模式的实现技术主要是多态，包括动多态和静多态。起先使用动多态，动多态是通过继承和虚函数实现，在运行期间，通过虚函数调用不同子类的虚成员函数来实现不同的功能。动多态的绑定是入侵性、插入式的，实现代价较高。静多态是通过模板编程实现的，通过在编译期间接口绑定不同的功能代码来实现多态。传统的静多态可以在一定程度上提供非入侵性、非插入式的实现，降低实现代价。但是，当具体产品构造函数参数不同、需要异类组合、具体产品数量繁多时，传统的动多态和静多态实现起来都很繁琐、很困难、复用性很低。

随着技术的发展，在模板编程的基础上进一步发展出泛型编程技术，并在C++标准库(STL)和实际工程中得到广泛应用[1]。针对设计模式传统实现方式存在的问题，本文以抽象工厂模式为例，通过C++11新标准泛型编程技术，给出一种可变参数泛型抽象工厂的实现方式。

2 抽象工厂模式(Abstract factory pattern)

抽象工厂模式属于创建型模式，其设计动机是：提供一个接口，让该接口负责创建一系列“相关或者相互依赖的对象”，用户无需指定工厂和产品的具体的类型、无需了解它们的具体实现，从而绕过常规的对象创建方式。

抽象工厂模式结构如图1所示，抽象工厂将工厂和产品全部抽象化，一个抽象工厂可以生成一组抽象产品，而一个具体工厂则生成一个系列具体产品的特定组合[2]。AbstractFactory定义抽象工厂生成的一组抽象产品的接

口，AbstractProductA、AbstractProductB定义某一种抽象产品的接口；具体工厂ConcreteFactory1负责创建1系列具体产品类型ProductA1、ProductB1的组合，具体工厂ConcreteFactory2负责创建2系列具体产品类型ProductA2、ProductB2的组合，依此类推。

对象创建时，用户先创建具体工厂ConcreteFactory1或ConcreteFactory2类的对象，再用ConcreteFactory1创建1系列具体产品类型的组合，或用ConcreteFactory2创建2系列具体产品类型的组合。

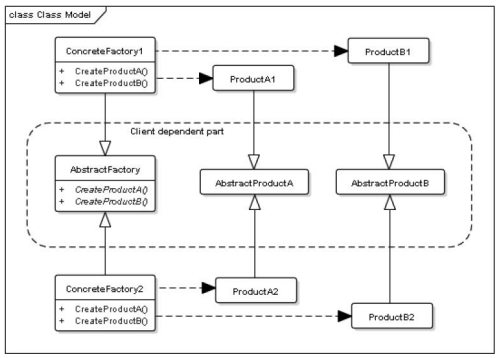

图1 抽象工厂模式结构图

Fig.1 Structure chart of abstract factory pattern

3 泛型编程实现抽象工厂(Implement abstract factory by generic programming)

抽象工厂传统实现方式的主要弱点是类型繁琐、类型依赖性强、可复用性弱。一个抽象工厂往往是为某一特定需求而设计，一般不能在其他场合复用。还有尽管抽象工厂可以将具体产品的创建委派给特定类型的工厂，但这种委派需要通过纯代码方式实现，没能充分利用语言所提供的抽象特性。

在设计模式的实现技术中，泛型编程能够提高编程效率、实现非侵入性实现，大大提高代码复用率。在泛型的参与下，许多设计可能更为精妙、更加优雅、更具扩展性[3]。面向对象编程(OOP，Object-Oriented Programming)试图将数据和方法封装在一起，各种操作都与数据类型相关。泛型编程(GP，Generic Programming)则试图将数据和方法分离开来，将类型参数化，使得同一种操作可以适用于不同数据类型，C++标准库(STL)就是一种典型应用[4]。

4 C++11实现可变参数泛型抽象工厂(Implement variable parameter generic abstract factory by C++11)

通过C++11新标准泛型编程技术，能够实现产品类型可变、参数可变、异类组合的泛型抽象工厂，当你需要特定类型抽象工厂时，可随时复用而无需再定义专门的抽象工厂实现。

4.1 结构图

C++11实现的可变参数泛型抽象工厂结构如图2所示，其完整实现代码附后。

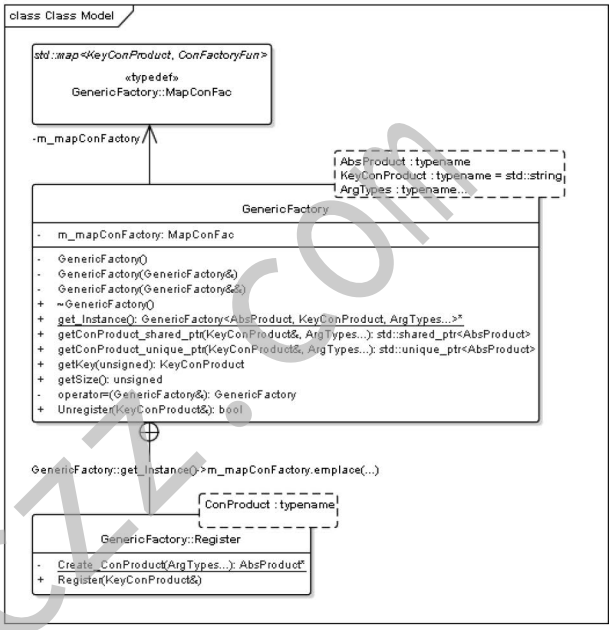

图2 C++11实现的可变参数泛型抽象工厂结构图

Fig.2 Structure chart of implement variable parameter generic abstract factory by C++11

具体实现非常简洁，只包含两个模板类：泛型工厂类GenericFactory、内嵌类具体产品注册类Register。这是基于C++11新标准实现的，必须在支持C++11新标准(2011年ISO发布)的编译器中才能正常编译使用，比如Visual Studio 2013等。

4.2 泛型工厂类GenericFactory

该类是该抽象工厂模式的核心，负责存储管理抽象工厂和各种具体工厂数据、具体产品的创建等工作[5]。在该类右上角的虚线框中，包含三个模板参数：抽象产品类AbsProduct、具体产品类标识KeyConProduct(缺省std::string)、产品类构造函数可变参数列表0—n项...ArgTypes。最后一个模板参数便是C++11支持的可变参数类型，支持可以数目、可变类型的0—n个参数[6]，下同。不同种类的抽象工厂无需通过不同的公共基类(即传统抽象工厂中的AbstractFactory)来表达接口的共性，能够很优雅地实现异类集合[7]。

该类通过静态变量和静态函数方式实现简单的单例模式(singleton)，各种构造器都是私有的，不允许外部构造。外部只能通过调用静态函数get_Instance获取唯一的静态实例unique_generic_factory。

该类只有一个数据成员mapConFactory，是std::map类型的私有数据成员，用于存放具体产品标识KeyConProduct和具体工厂构造函数指针ConFactoryFun组成的键值对。外部只能通过调用该类的公有函数存取mapConFactory。通过内嵌类Register类的实例化可以将数据存入mapConFactory；Unregister函数可以从mapConFactory中注销数据；getConProduct_shared_ptr、getConProduct_unique_ptr函数可以得到已注册标识为conp的类的一个实例；getSize函数可以得到已注册具体工厂的总数；getKey函数可以得到已注册索引为n的类的具体产品标识，只不过stl::map默认按键值类型自动排序(这里默认按字母顺序)，若需要控制排序方式，可以在应用程序中增加一个map<索引号,工厂标识>变量来实现。

4.3 具体产品注册类Register

该类是泛型工厂类GenericFactory的内嵌类，使用内嵌类可明显简化设计。该类作为抽象工厂的原料和接口，用于创建和注册具体工厂，将注册数据存入泛型工厂类的GenericFactory::mapConFactory。在该类右上角的虚线框中，只有一个模板参数：具体产品类ConProduct。

该类很简单，只包括一个构造函数Register和私有静态函数Create_ConProduct(包含可变参数列表)。该类与泛型工厂类GenericFactory的关系也很清晰，只是在创建该类对象时，由其构造函数调用泛型工厂类的GenericFactory::get_Instance函数，将创建具体产品的Create_ConProduct函数指针存入泛型工厂类的GenericFactory::mapConFactory。需要注意的是，该类并不实际创建具体产品对象，具体产品对象的创建，是由用户通过调用泛型工厂类的GenericFactory::getConProduct_shared_ptr、GenericFactory::getConProduct_unique_ptr函数来实现的。

这里由于使用了C++11新标准提供的新技术，使得代码更为精炼、更具扩展性。可变参数类型...ArgTypes使得该接口可以很优雅地支持具体产品类构造函数可变参数列表。智能指针shared_ptr、unique_ptr的使用实现了自动内存管理、无需delete释放内存，share_ptr通过引用计数共享所有权，unique_ptr独享所有权，可根据实际需要选用。使用C++11新特性emplace代替insert向STL容器添加新元素，可以在容器管理的内存中直接构造新元素，省去构造临时对象、减少内存开销，代码更为简洁高效。

5 实际使用(Actual use)

该泛型抽象工厂可以满足抽象工厂、简单工厂、可变参数、异类组合、具体产品数量繁多等情况的实现需求。实际使用很简单，首先创建各种具体工厂，方法就是创建各种具体产品注册类Register对象，将各种具体工厂的函数指针存入泛型工厂类的GenericFactory::mapConFactory。然后，便可以通过泛型工厂类的GenericFactory::get_Instance调用GenericFactory::getConProduct_shared_ptr、GenericFactory::getConProduct_unique_ptr函数来创建各种具体产品对象。可以通过函数封装实现抽象工厂的需要，将一系列相关产品封装在一个函数中，实现一次性创建一系列相关产品的需要。

5.1 具体工厂构造函数的参数可变

比如，现在已定义Shape基类和Square、Circle两个子类，便可以通过下面代码使用该泛型抽象工厂，实现具体工厂的注册和具体产品的创建。Square、Circle两个子类构造函数的参数可变，参数个数、类型都可以各不相同。这里，子类Square的构造函数有三个参数CPoint、CPoint、unsigned，子类Circle的构造函数有两个参数CPoint、int，其结构如图3所示。

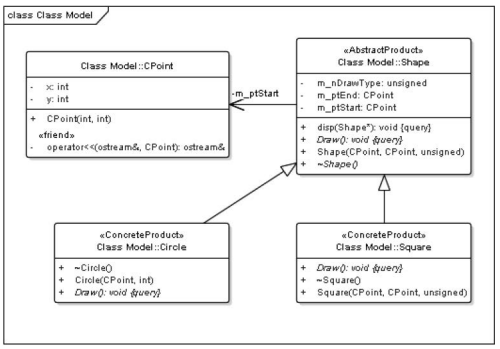

图3 具体工厂构造函数的参数可变

Fig.3 Parameters of concrete factory constructor are variable

5.2 具体工厂的异类组合

比如，跨国公司计算不同国家员工工资可能用到异类组合。假设美国员工工资包括奖金Bonus、津贴Subsidy、税收Tax等三个部分，中国员工工资包括奖金Bonus、津贴

Subsidy、税收Tax、住房公积金Found等四个部分。使用本文给出的C++11实现的可变参数泛型抽象工厂，通过函数封装不同抽象工厂的需要，便可轻松实现这种异类组合，其结构如图4所示。

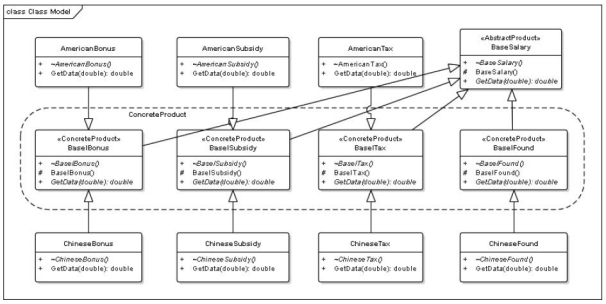

图4 具体工厂的异类组合

Fig.4 Heterogeneous combinations of concrete factory

6 泛型抽象工厂完整实现代码(Complete implementation code of generic abstract factory)

```
//GenericFactory.h，泛型抽象工厂头文件
#pragma once
#include <memory>
#include <string>
#include <map>

//泛型工厂类GenericFactory，包含3个模板参数：
//抽象产品类、具体产品类标识的类型、产品类构造函数可变参数列表0-n项
template <typename AbsProduct,typename KeyConProduct=std::string,typename...ArgTypes>
class GenericFactory
{
private:
    typedef AbsProduct*(*ConFactoryFun)(ArgTypes... args);
    typedef std::map<KeyConProduct,ConFactoryFun> MapConFac;
    MapConFac m_mapConFactory;
    GenericFactory(){}
    GenericFactory(const GenericFactory&)=delete;
    GenericFactory(GenericFactory&&)=delete;
    GenericFactory operator=(const GenericFactory& other)=delete;
public:
    ~GenericFactory(){}

    //具体产品注册类Register，有1个模板参数：具体产品类。使用内嵌类可明显简化设计
    template <typename ConProduct>
    class Register
    {
    public:
        Register(const KeyConProduct& conp)
        {
            GenericFactory::get_Instance()->m_mapConFactory.emplace(conp,Create_ConProduct);
        }
    private:
        inline static AbsProduct*Create_ConProduct(ArgTypes...args)
        {
            return new ConProduct(args...);
        }
    };
    bool Unregister(const KeyConProduct& conp)
    {
        return m_mapConFactory.erase(conp)==1;
    }
    std::shared_ptr<AbsProduct>getConProduct_shared_ptr(const KeyConProduct& conp,ArgTypes...args)
    {
        if(m_mapConFactory.find(conp)==m_mapConFactory.end())
            return std::shared_ptr<AbsProduct>();
        else
            return std::shared_ptr<AbsProduct>(m_mapConFactory[conp](args...));
    }
    std::unique_ptr<AbsProduct>getConProduct_unique_ptr(const KeyConProduct& conp,ArgTypes...args)
    {
```

```
            if(m_mapConFactory.find(conp)==m_mapConFactory.end())
                return std::unique_ptr<AbsProduct>();
            else
                return std::unique_ptr<AbsProduct>(m_mapConFactory[conp](args...));
        }
        unsigned getSize()
        {
            return m_mapConFactory.size();
        }
        KeyConProduct getKey(unsigned n)
        {
        MapConFac::iterator it=m_mapConFactory.begin();
        for(int i=0;it!=m_mapConFactory.end();i++,it++)
            if(n==i)        return it->first;
            return KeyConProduct();
        }
        inline static GenericFactory<AbsProduct,KeyConProduct,ArgTypes...>*get_Instance()
        {
            static GenericFactory<AbsProduct,KeyConProduct,ArgTypes...>unique_generic_factory;
            return&unique_generic_factory;
        }
    };
```

7 优点与缺点(Advantages and disadvantages)

C++11实现的可变参数泛型抽象工厂优点包括：(1)不同种类的抽象工厂无需通过不同的公共基类(也即传统抽象工厂中的AbstractFactory)来表达接口的共性，能够很优雅地实现异类集合；(2)通过C++11很优雅地实现具体产品类构造函数的可变参数列表；(3)通过智能指针shared_ptr、unique_ptr实现自动内存管理，无需delete释放内存；(4)泛型编程在编译期对所有的绑定操作进行检查，具有更好的类型安全性[8]；(5)运用C++11新特征和内嵌类使代码更为简洁高效；(6)本文给出的方法及代码具有实用性，可应用到软件项目中，其他相关模式也可参照实现。其缺点是：(1)使用中编译报错时，理解和查错较为困难；(2)实现代码虽小，但生成的执行代码可能较大。实际使用时，可根据其优缺点酌情选择。

8 结论(Conclusion)

综上所述，抽象工厂模式可以通过传统的动多态和静多态实现，也可以通过新兴的泛型和模板实现。抽象工厂模式的传统实现方式在处理具体产品构造函数参数不同、异类组合、具体产品数量繁多的情况时，都显得很繁琐、很困难、复用性很低。针对这一问题，本文以抽象工厂模式为例，应用C++11新标准和泛型编程技术，给出了一种C++11可变参数泛型抽象工厂的实现方式。该方式比传统实现方式更为简洁高效、复用性更强，优雅地实现了对产品类型可变、参数可变、异类组合的支持。

参考文献(References)

[1] Bemardi ML,Cimitile M,Lucca GD.Design pattem detection using a DSL-driven graph matching approach[J].Journal of Software Evolution&Process,2014,26(12):1233-1266.

[2] B Rasool G,Mader P.A customizable approach to design pattems recognition based 011 feature types[J].Arabian Journal for Science&Engineering,2014,39(12):8851-8873.

[3] 许涵斌,等.一种基于结构查询的UML设计模式识别方法[J].计算机科学,2014,41(11):50-55.

[4] B Matthew H.Austere(美).侯捷,译.泛型编程与STL[M].北京:中国电力出版社,2003.

[5] BLarman.C.(美).李洋,等,译.UML和模式应用[M].北京:机械工业出版社,2006.

[6] B Michael Wong(加),IBM XL编译器中国开发团队.深入理解C++11:C++11新特性解析与应用[M].北京:机械工业出版社,2013.

[7] B Gamma Erich(美),等.李英军,等,译.设计模式:可复用面向对象软件的基础[M].北京:机械工业出版社,2000.

[8] B Joshua Kerievsky(美).杨光,刘基诚,译.重构与模式(修订版)[M].北京:人民邮电出版社,2013.

作者简介：

闵　军(1966-)，男，硕士，研究员.研究领域：C++程序设计，设计模式，计算机网络.

罗　泓(1970-)，女，大专，工程师.研究领域：数据分析与处理，电路设计，信息管理系统.