文章编号：2096-1472(2017)-09-09-05

用lambda表达式和std::function类模板改进泛型抽象工厂设计

闵 军，罗 泓
(宜宾学院图书馆，四川 宜宾 644000)

摘 要：抽象工厂模式在软件设计中应用广泛，但抽象工厂模式的传统实现方式存在诸多不足。随着技术的发展，设计模式的实现方式也在不断改进。C++11新标准发布之后，涌现了许多改进方案。本文将在这些改进的基础之上，使用C++11的lambda表达式、std::function类模板等新技术，通过数据结构和代码结构的优化等方式进一步改进泛型抽象工厂设计，给出一种“新型泛型抽象工厂”的实现方式。实验结果表明，该方式更为简洁高效、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。该实现方式及代码实用性较强，可以在软件项目中实际使用。

关键词：C++11；lambda；function；泛型；抽象工厂
中图分类号：TP311.1 **文献标识码**：A

Improvement of the Generic Abstract Factory Design through the Lambda Expression and the std::function Class Template

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

Abstract: The abstract factory pattern has been widely used in software design,but there are still some shortcomings in the traditional implementation of the abstract factory design pattern.With the development of technology,the implementation of design patterns is constantly improving.After the release of new C++11 standards,many improvement have emerged. Based on these improvements,this paper further improves the generic abstract factory design by adopting the new technology of the C++11 lambda expression and the std::function class template,optimizing the data structure and the code structure. The implementation model of New Generic Abstract Factory is proposed.Experimental results show that this model is more concise and efficient with better reusability.This model can gracefully implement the support to variable product types,variable parameters,and heterogeneous combinations.With decent practicality,this implementation method and code can be actually applied in software projects.

Keywords:C++11;lambda;function;generic;abstract factory

1 引言(Introduction)

抽象工厂模式是最具一般性、最为抽象的一种工厂模式，由于该模式的使用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。不过，抽象工厂模式的传统实现方式存在诸多不足，诸如实现复杂、类型烦琐、类型依赖性强、可复用性弱等。随着技术的发展，人们不断使用多态机制、模板编程、泛型编程等技术改进设计模式[1]。C++11新标准发布之后，涌现了许多改进方案，比如将具体工厂构造函数保存到关联容器中实现自动注册、使用可变参数模板和类模板实现泛型工厂、使用内嵌类简化设计等。本文将在这些改进的基础之上，使用C++11的lambda表达式、std::function类模板等新技术，通过数据结构和代码结构的优化等方式进一步改进泛型抽象工厂设计，给出一种更为简洁高效的“新型泛型抽象工厂”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，简单地说，抽象工厂模式就是用于完成“多系列相互依赖的具体产品”的创建工作，避免客户程序和这种“多系列具体产品创建工作”的紧密耦合[2]。抽象工厂模式结构如图1所示。

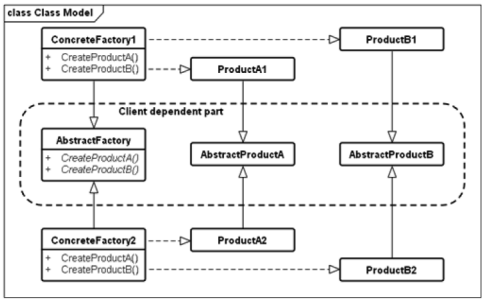

图1 抽象工厂模式结构图

Fig.1 Structure chart of abstract factory pattern

3 C++11实现泛型抽象工厂(Implement generic abstract factory by C++11)

泛型编程技术能够提高编程效率、实现非侵入性实现，大大提高代码复用率[3]。在设计模式的实现技术中，泛型编程技术是改进抽象工厂传统实现方式的一种有效手段。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。图2显示了一种C++11实现的可变参数泛型抽象工厂的结构，这种实现方式包含两个类模板：泛型工厂类GenericFactory、内嵌类具体工厂注册类Register[4]。

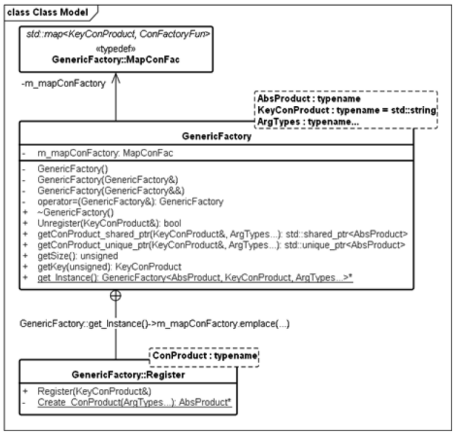

图2 C++11实现的可变参数泛型抽象工厂结构图

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

4 使用lambda表达式和std::function类模板设计“新型泛型抽象工厂”(Design *New generic abstract factory* by lambda expression and std::function class template)

上面提到的泛型抽象工厂设计方式，虽然使用了关联容器、可变参数模板和内嵌类等技术，但也存在可以优化的地方。下面，本文将在这些改进的基础之上进一步优化，介绍更为简洁高效的“新型泛型抽象工厂”的实现方式。

4.1 “新型泛型抽象工厂”的结构图

“新型泛型抽象工厂”的结构如图3所示。

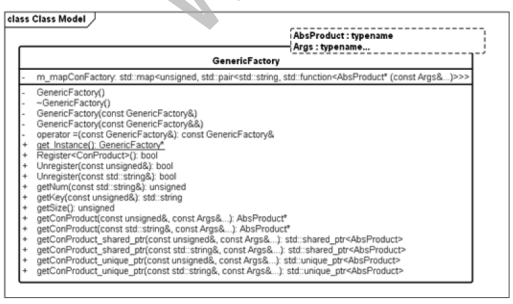

图3 “新型泛型抽象工厂”结构图

Fig.3 Structure chart of *New generic abstract factory*

4.2 用lambda表达式代替内嵌类

lambda表达式是C++11引入的最重要、最常用的特性之一，它具有简洁高效、声明式编程风格、可实现功能闭包等优势[4]。前面介绍的泛型抽象工厂设计方式，使用了内嵌类来生成具体产品的创建函数[5]，其结构图参见图2。在前面设计的基础之上，可以用lambda表达式来代替内嵌类，直接以lambda表达式作为具体产品的创建函数，其结构图参见图3，代码可参见后面的完整代码。两相对比，使用lambda表达式代替内嵌类，明显简化了代码结构，不过可读性也许会差一些。

4.3 用std::function类模板存储和操作lambda表达式

Lambda表达式是一种匿名函数对象(或称仿函数)，其具体类型是一种依赖于具体实现的、唯一的函数对象类型，这种类型的名字只有编译器才知道[6]。虽然某些简单的lambda表达式可以直接或间接地转换为函数指针，但一般都推荐使用auto关键字来标识lambda表达式的类型；若想获取lambda表达式的具体类型，可以用C++11提供的decltype类型操作符得到；如果用户要把lambda表达式用作参数传递，那就需要使用std::function对象进行捕获。

std::function类模版是一种通用的、多态的函数封装工具，它是对C++中现有各种可调用实体的一种类型安全的封装(像函数指针这类可调用实体是类型不安全的)[7]。通过这种封装，形成一种单一的可调用的std::function新对象，使得代码变得简单明了。

4.4 用std::function类模板存储和操作具体工厂信息

在“新型泛型抽象工厂”的设计中，关键数据成员m_mapConFactory用于存放具体产品标识和具体工厂创建函数指针的列表信息。使用lambda表达式代替内嵌类作为具体产品的创建函数，m_mapConFactory存储的数据类型就需要作相应改变。

m_mapConFactory中原来存储的是具体产品创建函数的函数指针(图2)。现在就不能直接保存为函数指针，而需要保存为以std::function类模板封装的lambda表达式，具体定义参见图3和后面的完整代码。

5 优化“新型泛型抽象工厂”的数据结构(Optimize data structure of *New generic abstract factory*)

为了优化“新型泛型抽象工厂”保存的数据结构，对关键数据成员m_mapConFactory的结构做了调整，具体可参见后面完整代码中m_mapConFactory的定义。通过这种调整，再配合其他相应修改，用户注册具体工厂变得更为便，只需指定具体产品的类型即可完成注册(参见后面示例)。具体工厂注册时，“新型泛型抽象工厂”类将根据用户提供的具体产品类型，自动生成唯一的具体产品序号、获取其类型名称，无须用户再从外部输入具体产品的类型标识。最后，将这些数据就地构造为容器元素，存入m_mapConFactory容器中。

6 “新型泛型抽象工厂”的代码优化(Code optimization of *New generic abstract factory*)

“新型泛型抽象工厂”的代码设计，也进行了一些相关

优化。该GenericFactory类的模板参数已作简化，只包含两个部分：抽象产品类、具体产品构造函数可变参数列表0—n项。这种模板参数的分配方式是比较合理的，实质上是规定了具体产品构造函数的具体类别：包括返回值和参数列表，返回值必须是AbsProduct*指针类型、参数类型和个数列表Args...args必须一致。当具体产品构造函数的参数类型和个数不同时，将产生不同版本的GenericFactory实例。

该GenericFactory类通过静态函数和静态变量的方式实现简单的单件模式(Singleton)，各种构造器都是私有的，不允许外部构造。外部只能通过调用其静态接口函数get_Instance获取唯一的静态实例Singleton_GenericFactory。

具体工厂注册函数Register只有一个模板参数：具体产品类型ConProduct，要求ConProduct类必须是AbsProduct的子类。在具体产品构造函数返回值为AbsProduct*指针类型、参数列表一致的前提下，可以注册不同实现细节的具体工厂。

当函数参数为常量引用时，用户传入临时对象或已创建的变量都可以，因此，GenericFactory类的成员函数都尽量使用常量引用方式传参，减少临时对象的构造和拷贝。另外，无须修改数据成员的成员函数都尽量声明为const类型，以提高代码的健壮性。

7 “新型泛型抽象工厂”完整实现代码(Complete implementation code of *New generic abstract factory*)

下面给出本文介绍的“新型泛型抽象工厂”的完整实现代码。需要注意的是，本文给出的代码是基于C++11新标准实现的，必须在支持C++11新标准的编译器中才能正常编译使用，比如Visual Studio 2013及以上版本。

```
//GenericFactory.h，泛型抽象工厂头文件v2.0.3
#pragma once
#include<functional>
#include<memory>
#include<string>
#include<map>
//泛型工厂类GenericFactory包含2个模板参数：
//抽象产品类、具体产品类构造函数可变参数列表0—n项
template<typename AbsProduct,typename...Args>
//AbsProduct为Abstract Product缩写
class GenericFactory
{
private:
    GenericFactory() {}
    ~GenericFactory() {}
    GenericFactory(const GenericFactory&)=delete;
    GenericFactory(GenericFactory&&)=delete;
    GenericFactory& operator=(const GenericFactory&)=delete;
    GenericFactory& operator=(GenericFactory&&)=delete;
    //std::map<具体产品序号,std::pair<具体产品类型标识字符串,具体产品创建函数的function指针>>
    std::map<size_t,std::pair<std::string,std::function<AbsProduct*(const Args&...)>>>m_mapConFactory;
public:
    inline static GenericFactory*get_Instance()
    {
        static GenericFactory Singleton_GenericFactory;
        return &Singleton_GenericFactory;
    }
    //Register只有1个模板参数：具体产品类ConProduct，要求ConProduct类必须是AbsProduct的子类
    //具体产品构造函数返回值为AbsProduct*类型、参数列表一致的前提下，可注册不同实现细节的具体工厂
    template<typename ConProduct>//Conp、ConProduct为Concrete Product缩写
    bool Register()
    {
     //得到依次递增的产品序号，代替成员变量m_nNo
        size_t nConp=(0==m_mapConFactory.size()?0:((--m_mapConFactory.end())->first)+1);
        //得到具体产品类型名称
        std::string strConp=typeid(ConProduct).name();
        //得到具体产品创建函数的指针(Lambda表达式)
        auto funCreator=[](const Args&...args){return new ConProduct(args...);};
        //就地构造容器元素，减少临时对象拷贝。ConFactory为Concrete Factor缩写
        auto pairRet=m_mapConFactory.emplace(nConp,make_pair(strConp,funCreator));
        return pairRet.second;
    }
    bool Unregister(const size_t&nConp)
    {
      return m_mapConFactory.erase(nConp)==1;
    }
    //当函数参数为常量引用时，用户传入临时对象、或已创建的变量都可以。
```

```
//因此，成员函数都尽量使用常量引用方式传参，减少临时对象的构造和拷贝。
bool Unregister(const std::string&strConp)
{
        return Unregister(getNum(strConp));
}
//无须修改数据成员的成员函数都尽量声明为const类型
size_t getNum(const std::string&strConp) const
{
  for(auto m:m_mapConFactory)
  {
    if(strConp==m.second.first)return m.first;
        }
        return SIZE_MAX; //SIZE_MAX在32位为UINT_MAX，64位为_UI64_MAX，以适应size_t类型
}
std::string getStr(const size_t& nConp)
{
   return(m_mapConFactory.find(nConp)==m_mapConFactory.end())?
  std::string():m_mapConFactory[nConp].first;
}
size_t getSize() const
{
        return m_mapConFactory.size();
}
AbsProduct*getConProduct(const size_t& nConp,const Args&...args)
{
return(m_mapConFactory.find(nConp)==m_mapConFactory.end())?
nullptr:m_mapConFactory[nConp].second(args...);
}
AbsProduct*getConProduct(const std::string& strConp,const Args&...args)
{
return getConProduct(getNum(strConp),args...);
}
std::shared_ptr<AbsProduct>getConProduct_shared_ptr(const size_t& nConp,const Args&...args)
{
 return std::shared_ptr<AbsProduct>(getConProduct(nConp, args...));
}

std::shared_ptr<AbsProduct>getConProduct_shared_ptr(const std::string& strConp,const Args&...args)
{
        return std::shared_ptr<AbsProduct>(getConProduct(getNum(strConp),args...));
}
std::unique_ptr<AbsProduct>getConProduct_unique_ptr(const size_t& nConp,const Args&...args)
{
        return std::unique_ptr<AbsProduct>(getConProduct(nConp,args...));
}
std::unique_ptr<AbsProduct>getConProduct_unique_ptr(const std::string& strConp,const Args&...args)
{
        return std::unique_ptr<AbsProduct>(getConProduct(getNum(strConp),args...));
}
};
```

8 “新型泛型抽象工厂”的实际使用(Actual use of *New generic abstract factory*)

8.1 “新型泛型抽象工厂”的使用方式

“新型泛型抽象工厂”设计比较合理周全，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[8]。使用也很简单，首先创建各种具体工厂，方法就是通过GenericFactory::get_Instance调用其Register注册函数，将各种具体工厂的创建函数指针存入m_mapConFactory容器中。接下来，用户便可以通过GenericFactory::get_Instance调用各种公共接口函数，更为灵活方便地使用各种功能。

用户可以调用getNum获取某具体产品类型的序号；调用getStr获取某具体产品类型的标识字符串；调用getSize获取现有具体工厂数目。创建和注销具体工厂的各种接口函数都设计了相应的重载函数，用户可以通过具体产品序号或具体产品类型字符串完成所需工作。需要注意的是，getConProduct接口返回的是容器内部分配的堆内存指针，用户需要管理其生命周期，建议使用getConProduct_shared_ptr或getConProduct_unique_ptr接口，它返回的是智能指针，这样，用户就不用管理其生命周期。

若用户需求比较复杂，可以通过函数封装方式实现抽象工厂的需要，将一系列相关产品封装在一个函数当中，实现一次性创建一系列相关产品的需要(参见后面示例8.3)。

8.2 具体产品构造函数的参数可变

如果已经定义了Shape基类和Rect、Circle两个子类，便可以通过下面代码使用“新型泛型抽象工厂”，实现具体工厂的注册和具体产品的创建。Rect、Circle两个子类的构造函数参数可变，参数个数、类型都可以各不相同。这里，子类Rect的构造函数有三个参数unsigned、CPoint、CPoint，子类Circle的构造函数有两个参数CPoint、double[9]。比如，下面代码用于完成注册具体工厂、创建具体产品对象并调用其Draw函数的工作：

```
//注册和创建Rect类对象并调用其Draw函数。首先获取GenericFactory类指针
//类型参数是：抽象产品类，具体产品类构造函数可变参数列表0-n项
auto pGF0=GenericFactory<Shape,unsigned,CPoint,CPoint>::get_Instance();
pGF0->Register<Rect>();          //注册具体工厂
//创建具体产品并调用其Draw函数
pGF0->getConProduct_shared_ptr(0,0,CPoint(3,4),CPoint(5,6))->Draw();
//注册和创建Circle类对象并调用其Draw函数
auto pGF1=GenericFactory<Shape,CPoint,double>::get_Instance();
pGF1->Register<Circle>();        //注册具体工厂
//创建具体产品并调用其Draw函数
pGF1->getConProduct_shared_ptr(0,CPoint(7,9),3.3)->Draw();
```

该示例的结构参见图4。

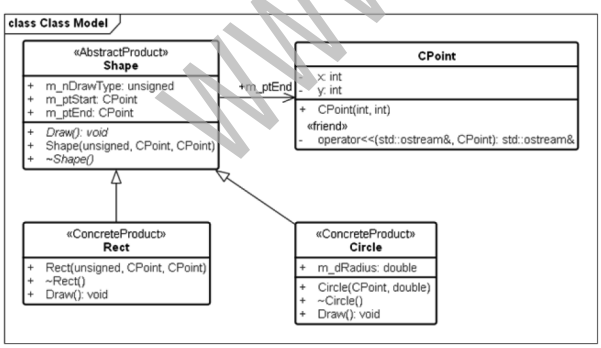

图4 具体产品构造函数的参数可变

Fig.4 Parameters of concrete factory constructor are variable

8.3 具体工厂的异类组合

在抽象工厂应用中，经常提到一个典型案例，跨国公司计算不同国家员工工资时可能用到异类组合的例子。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。使用本文改进的“新型泛型抽象工厂”，通过函数封装不同抽象工厂的需要，便能更为优雅地实现这种异类组合[9]。比如，下面代码用于完成注册具体工厂、创建具体产品对象并调用特殊业务逻辑的工作：

```
//注册具体工厂并调用特殊业务逻辑：ChineseFactory
void GetChineseSalary(double dBasicSalary)
{
    //为简化程序，定义泛型抽象工厂实例的指针变量
    auto pGF=GenericFactory<BaseSalary>::get_Instance();
    //注册具体工厂
    pGF->Register<ChineseBonus>();
    pGF->Register<ChineseSubsidy>();
    pGF->Register<ChineseTax>();
    pGF->Register<ChineseFound>();    //住房公积金
    //调用特殊业务逻辑
    double dOtherSalary=
        pGF->getConProduct_unique_ptr(0)->GetData(dBasicSalary)+
        pGF->getConProduct_unique_ptr(1)->GetData(dBasicSalary)-
        pGF->getConProduct_unique_ptr(2)->GetData(dBasicSalary)-
        pGF->getConProduct_unique_ptr(3)->GetData(dBasicSalary);  //住房公积金
    //输出信息
    cout<<"ChineseFactory:"<<endl;
    cout<<"dBasicSalary="<<dBasicSalary<<endl;
    cout<<"dActualSalary="<<dBasicSalary+dOtherSalary<<endl;
}
```

该示例的结构参见图5。

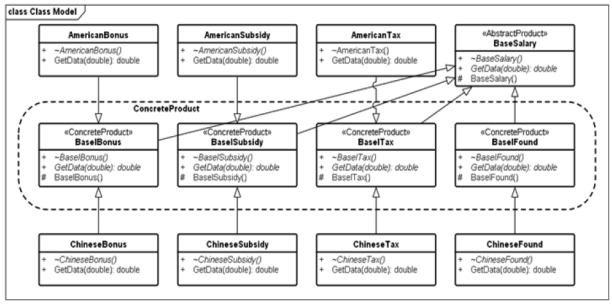

图5 具体工厂的异类组合

Fig.5 Heterogeneous combinations of concrete factory

9 结论(Conclusion)

综上所述，抽象工厂模式的实现方式一直都在不断改进。C++11新标准发布之后，涌现了许多改进方案。本文在这些改进的基础之上，使用C++11的lambda表达式、std::function类模板等新技术，通过数据结构和代码结构的优化等方式进一步改进泛型抽象工厂设计，给出了一种更为简洁高效的“新型泛型抽象工厂”的实现方式。实验结果表明，该方式更为简洁高效、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。该实现方式及代码实用性较强，可以在软件项目中实际使用。

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 on feature types[J].Arabian Journal for Scienc e&Engineering,2014,39(12):8851-8873.

[3] Stephen Prata.C++ Primer Plus,Sixth Edition[M].USA:Addison-Wesley Professional,2011.

[4] 祁宇.深入应用C++11:代码优化与工程级应用[M].北京:机械工业出版社,2015.

[5] 闵军,罗泓.C++11实现可变参数泛型抽象工厂[J].软件工程,2017,20(05):18-22.

[6] B Michael Wong(加),IBM XL编译器中国开发团队,著.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] Marc Gregoire(美),著.张永强,译.C++高级编程(第3版)[M].北京:清华大学出版社,2015:519-521.

[8] B Gamma Erich(美),等,著.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.

[9] 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,2013:51-59.

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，专科，工程师.研究领域：数据分析与处理，电路设计，信息管理系统.

(上接第20页)

赛/创新活动等进行过程跟踪，对能力成长进行评估。

对学生的实验、实训、竞赛、创新活动等进行信息记录和轨迹跟踪，并通过实验教学云平台的大数据分析和挖掘技术，帮助老师掌握学生的学习路径、学习情况等信息，构建学生社会需求—实践项目—能力达成度—创新创业能力等的信息反馈和持续改进机制，为教学过程的持续改进提供数据依据，最终促进学生实践创新能力培养。

构建面向行业实际岗位的卓越工程师岗位能力成熟度模型，该模型借鉴软件工程的CMMI模型，详细定义达成相应岗位各级能力所应该具备的技术能力和水平。利用基于岗位能力成熟度模型的能力评估工具，将学习路径流程化和图形化，通过云端行为记录和大数据分析自动对模型中学生的学习过程进行记录和统计，并与岗位素质模型的要求进行比较，为学生的学习和教师的授课提供科学指引。

4 结论(Conclusion)

我校多维协同创建的计算机类专业人才实践与创新能力培养模式、“六平台三层次一机制”的实践与创新能力培养体系已经过3—5年的实践，并将其部分成果汇聚于计算机实验教学中心网站(http://cslab.whut.edu.cn)中。该体系对提高学生的实践创新能力，提高毕业生的就业质量和就业率效果明显，同时对同类院校计算机专业的建设有一定参考价值。

参考文献(References)

[1] The Joint Task Force on Computing Curricula of ACM\IEEE. Computer Science Curricula 2013 Ironman Draft (Version 0.8) [EB/OL].http://ai.stanford.edu/users/sahami/CS2013/.

[2] Kong X,et al.Ubiquitous auction learning system with TELD(Teaching by Examples and Learning by Doing) approach:A quasi-experimental study[J].Computers & Education,2017,111(C):144-157.

[3] Qu SY,et al.Experimental Teaching Centre Platform "New Engineering" Practice Teaching Mode[J].EURASIA JOURNAL OF MATHEMATICS SCIENCE AND TECHNOLOGY EDUCATION,2017,13(7):4271-4279.

[4] 王晓勇,等.以创新创业能力培养为核心的计算机专业实践课程教学改革[J].计算机教育,2011,9(9):1-8.

[5] 饶文碧,王云华,袁景凌.基于云平台的计算机开放式实验教学与管理模式研究[J].计算机教育,2016,14(10):137-140.

[6] 杨焱超,熊盛武,饶文碧.基于云计算的计算机类实验教学平台搭建与应用[J].实验技术与管理,2016,54(10):147-151.

作者简介：

饶文碧(1967-)，女，博士，教授.研究领域：普适计算，智能软件技术.

熊盛武(1966-)，男，博士，教授.研究领域：数据挖掘与知识发现，智能计算与自然计算.

袁景凌(1975-)，女，博士，教授.研究领域：绿色计算，智能方法.