计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(6):218−223 [doi: 10.15888/j.cnki.csa.007449]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于模板方法与抽象工厂的复合模式①

林欣欣, 郭元术, 运杰伦, 苏欣欣

(长安大学 信息工程学院, 西安 710064)
通讯作者: 林欣欣, E-mail: 1298831986@qq.com

摘　要: 面向对象的程序设计越来越追求程序的可复用性和灵活性, 对于经验较少的程序设计者直接得到具有良好复用性和灵活性的程序是具有一定难度的, 软件设计模式就是提取面向对象程序设计者的经验, 并对其进行总结. 模板方法模式中父类定义一个算法的框架, 用模板方法规定算法的执行步骤, 将可变的步骤延迟到子类实现, 每一种不同的实现都需要定义一个新的子类, 系统会越来越庞大, 系统的可维护性以及可读性越来越差. 因此将抽象工厂模式嵌入到模板方法模式形成一个复合模式, 复合模式的设计核心是为每一个延迟到子类的可变的步骤提供一个创建对象的接口, 该接口对一个完整的产品族进行了定义. 复合模式既保证了算法结构的稳定性, 又分离了具体的实现类, 增强了程序的健壮性、可复用性以及灵活性.

关键词: 设计模式; 抽象工厂; 模板方法; 复合模式; 程序设计原则

引用格式: 林欣欣,郭元术,运杰伦,苏欣欣.基于模板方法与抽象工厂的复合模式.计算机系统应用,2020,29(6):218–223. http://www.c-s-a.org.cn/1003-3254/7449.html

Compound Pattern on Template Method and Abstract Factory

LIN Xin-Xin, GUO Yuan-Shu, YUN Jie-Lun, SU Xin-Xin

(School of Information Engineering, Chang'an University, Xi'an 710064, China)

Abstract: Object-oriented programming is increasingly pursuing the reusability and flexibility of the program. It is difficult for programmers with less experience to obtain reusable and flexible programs directly. The software design pattern is to extract the experience of object-oriented programmers, and then to summarize them. In the template method pattern, the parent class defines an algorithm framework, uses the template method to specify the execution steps of the algorithm, and delays the variable steps to the subclass implementation. Each different implementation needs to define a new subclass. The system will be larger with maintainability and readability. Therefore, embedding the abstract factory pattern into the template method pattern forms a composite pattern. The core of the composite pattern design is to provide an interface for creating objects for each variable step that is delayed to the subclass. The interface is for a complete definition of product family. The composite mode not only ensures the stability of the algorithm structure, but also separates the specific implementation class, which enhances the robustness, reusability and flexibility of the program.

Key words: design pattern; abstract factory; template method; composite pattern; programming principle

引言

自 Erich Gamma、Richard Helm 等提出 23 种"设计模式"[1,2]后, 人们对于设计模式方面的研究越来越重视, 程序设计人员对 23 种设计模式进行了学习[3–5], 并

① 基金项目: 河南省交通运输厅重点项目 (220024140173)
Foundation item: Major Project of Department of Transportation, Henan Province (220024140173)
收稿时间: 2019-11-05; 修改时间: 2019-11-28; 采用时间: 2019-12-17; csa 在线出版时间: 2020-06-10

将其运用于实践.

软件设计模式描述的是在面向对象软件设计过程中针对特定问题提出的解决方案, 应用该解决方案可以得到高内聚、低耦合并且复用性更高的程序. 这些方案并不是人们一开始就采用的设计方案, 而是为增加软件的灵活性和可复性进行的总结[1].

面向对象程序设计原则[6]包含开闭、依赖倒置、单一职责等 7 条原则. 为保证软件的灵活性、可复用性以及健壮性各个原则均是需要努力保证满足的.

程序设计人员在实践过程中发现独立的应用某一种设计模式, 并不能很好的满足程序设计的需求. 如文献《模板方法模式的改进》[7]通过将策略者模式嵌入到模板方法模式, 从而对模板方法模式进行改进, 该模式是将延迟到子类的算法的不同实现方式写入到各个类, 各个类之间可以相互替换.

本文将讨论模板方法及抽象工厂模式, 并将抽象工厂模式放在模板方法模式中进行讨论, 最终给出复合模式. 本文的复合模式侧重点在于延迟到子类的算法如果使用到具体的其他的类, 如何在保证“单一职责”的原则下进行其他具体类对象的引用, 并且使得各个子类的替换较为便捷. 复合模式还在一定程度上解决了模板方法模式由于变化的子类实现方式过多而造成系统庞大的问题, 并且给出了复合模式的 UML 结构图, 其他程序设计人员可以直接复用该复合模式.

1 模板方法模式

模板方法模式[8]UML 结构图[9–11]如图 1 所示.

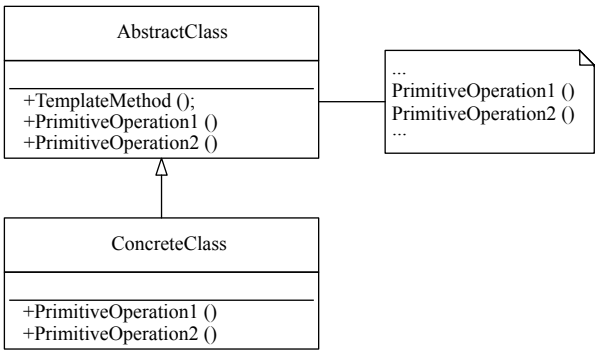

图 1 模板方法模式结构图

模板方法模式中包括两类参与者:

(1) 算法定义类 (AbstractClass): 该类包括模板方法、具体方法、抽象方法. 在结构图中 TemplateMethod() 为模板方法, PrimitiveOperation1() 与 PrimitiveOperation2() 为模板方法中的两个抽象操作.

(2) 具体实现类 (ConcreteClass): 重新定义模板方法中的抽象操作以提供具体的行为. ConcreteClass 类实现方式的不同可以使得算法的实现各不相同.

2 抽象工厂模式

抽象工厂模式[12,13]UML 结构图如图 2 所示.

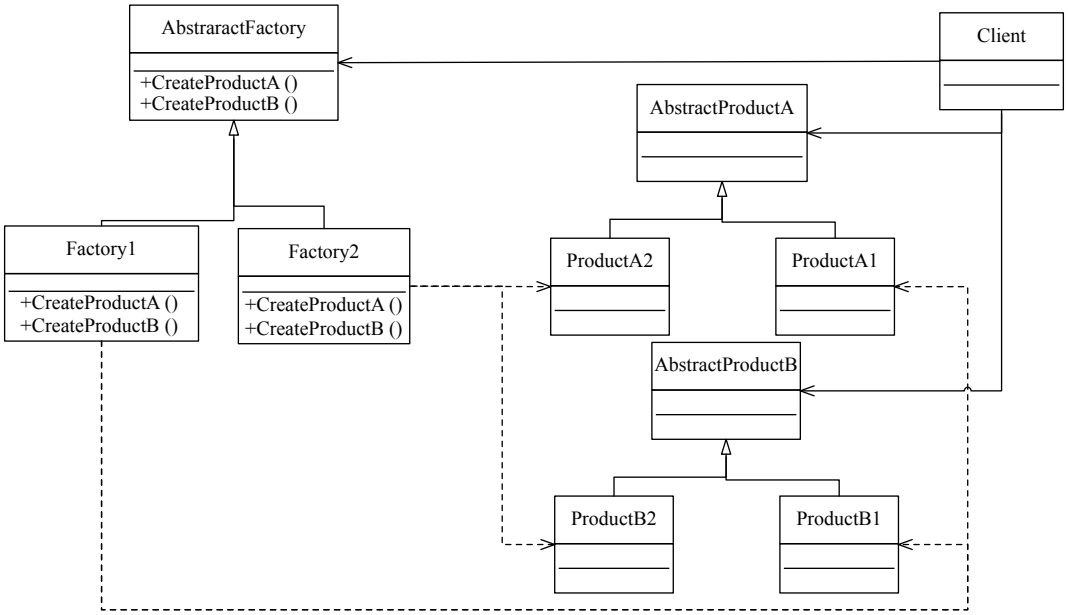

图 2 抽象工厂模式结构图

抽象工厂模式中包括 5 类参与者:

(1) 抽象工厂 (AbstractFactory): 以抽象的方式创建一个完整的产品族.

(2) 具体工厂 (Factory1, Factory2): 创建具体的产品对象, 不同的具体工厂对配置不同的具体产品进行生产.

(3) 抽象产品 (AbstractProductA, AbstractProductB): 对一类产品进行抽象, 为一类产品对象声明一个接口.

(4) 具体产品 (ProductA1, ProductA2, ProductB1, ProductB2): 有着自己独特配置的产品, 被相应的具体工厂所生产.

(5) 代码调用者 (Client): Client 不是人, 而是代码的调用者.

3 复合模式的设计

模板方法模式的意图是: 父类定义一个算法的框架, 用模板方法规定算法的执行步骤, 将可变的步骤延迟到子类实现, 每一种不同的实现都定义一个新的子类, 父类调用子类, 子类对父类进行灵活的拓展. 在本文的模板方法模式中, 如果延迟到具体子类 ConcreteClass 的抽象操作 PrimitiveOperation1() 是从多个产品族中选取一种产品族实现算法步骤, 如果将所有类型以实例化的形式写在 ConcreteClass 中, 代码复用性不高、灵活性不强. 因此考虑将抽象工厂模式嵌入到模板方法模式中. 抽象工厂模式的意图是: 在抽象工厂类中以抽象的形式创建一个完整的产品族, 在创建产品族的过程中不关注产品的具体实现类. 用户只需根据自己所需产品族的类型选择具体工厂, 而不需要关注产品族中产品的创建. 创建一个抽象工厂 AbstractFactory, 该抽象工厂创建一个完整的产品族, 其子类具体工厂 ConcreteFactory1、ConcreteFactory2 等生产产品类型各不相同的具体产品, 并且当需要添加新的产品类型时只需添加 AbstractFactory 的子类即可, 而无需改动其他的类, 延迟到 ConcreteClass 中的算法步骤就可以在不指定产品类型的情况下创建一个 AbstractFactory 的引用, 并将该引用指向子类对象 ConcreteFactory1 或 ConcreteFactory2, 从而获得产品对象[14]. 在此过程中, 产品对象的获得不是通过直接 new 的方式, 而是通过函数的调用, 这样使得对象的创建和对象实现的业务相分离, 降低系统的耦合度. AbstractClass 通过对 ConcreteClass 的调用完成全部的算法流程.

复合模式 UML 结构图如图 3 所示.

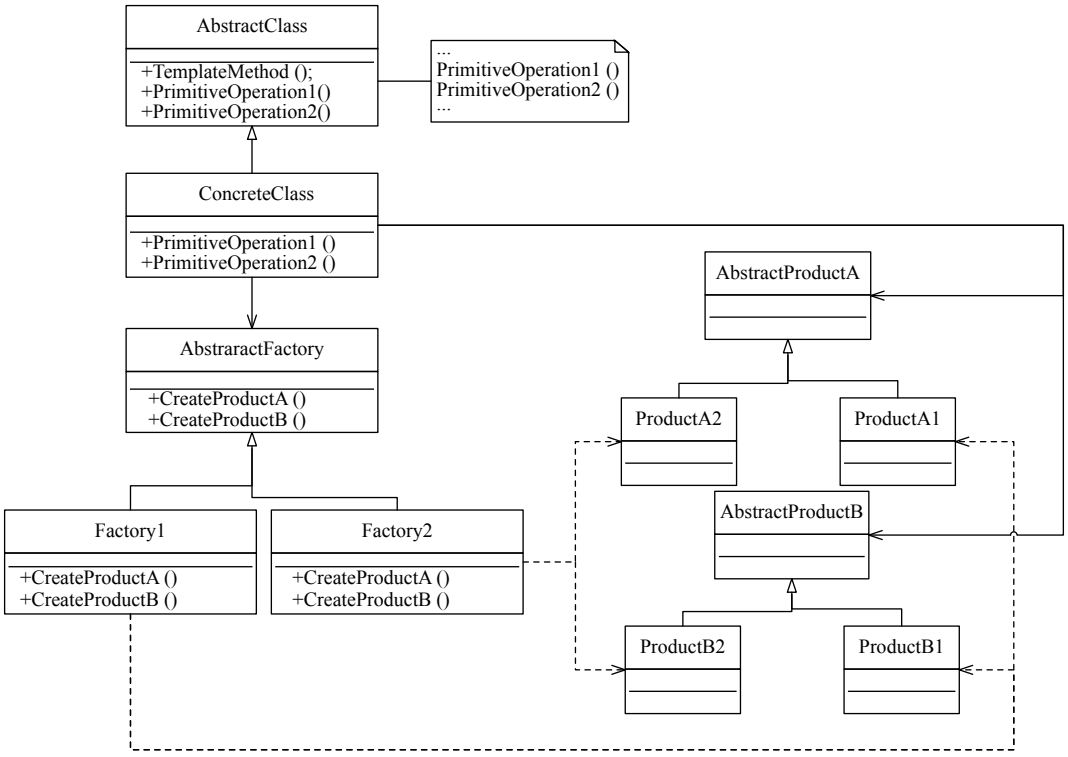

图 3 复合模式结构图

复合模式中包括 6 类参与者:

(1) 算法定义类 (AbstractClass): 作用同模板方法模式中的算法定义类.

(2) 具体实现类 (ConcreteClass): 作用同模板方法模式中的具体实现类, 同时, 又充当抽象工厂模式中的 Client 接口.

(3) 抽象工厂 (AbstractFactory): 作用同抽象工厂模式中的抽象工厂类.

(4) 具体工厂 (Factory1, Factory2): 作用同抽象工厂模式中的具体工厂类.

(5) 抽象产品 (AbstractProductA, AbstractProductB): 作用同抽象工厂模式中的抽象产品类.

(6) 具体产品 (ProductA1, ProductA2, ProductB1, ProductB2): 作用同抽象工厂模式中的具体产品类.

4 复合模式的实现

考试期间有个别同学作弊是一直存在的问题. 某老师为解决该问题, 给出的解决方案是: 按照个人学号分配试卷, 试卷题目相同, 但出现顺序不同. 该方案确实能一定程度上解决同学之间相互抄袭的问题, 但并不能完全杜绝. 并且, 该方案对改卷老师并不友好, 要求改卷老师能明确知道每道题目的答案. 批改试卷的效率较低, 难度较大.

对于各高校而言, 存在足够的资源进行上机测试. 复合模式应用于在线测试系统的后台程序编程中, 并将结果放到页面显示. 后台结构图如图 4 所示.

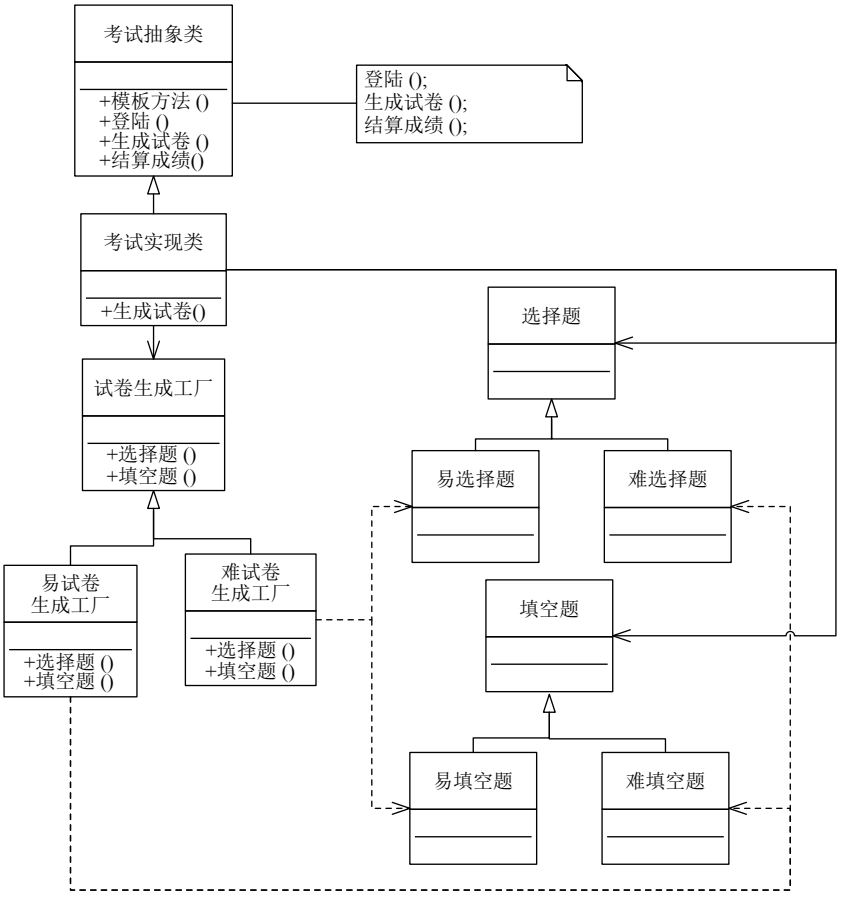

图 4 考试模块结构图

算法中包括 6 类参与者:

(1) 考试抽象类: 模板方法定义完成考试的步骤. 其中生成试卷功能交给子类具体实现.

(2) 考试实现类: 通过使用试卷生成工厂及其子类对象完成试卷生成功能.

(3) 试卷生成工厂: 定义产品族, 产品族包括选择题与填空题.

(4) 易试卷生成工厂、难试卷生成工厂: 分别返回难易程度为易与难的选择题对象与填空题对象.

(5) 选择题抽象类、填空题抽象类: 分别定义选择

题与填空题的共性以便子类进行继承.

(6) 易选择题、难选择题、易填空题、难填空题: 从数据库中查询并返回对应难度的选择题或者填空题.

4.1 核心功能代码

(1) 考试抽象类:

```
public abstract class AbstractClass {
    //模板方法
    public void tempLate() {
        login();
        List<XuanZe> list=CreateXZ();
        score();
    }
    public abstract List<XuanZe> CreateXZ();
}
```

(2) 考试实现类:

```
public class ConcreteClassA extends AbstractClass {
        @Override
        public List<XuanZe> CreateXZ() {
            AbstractFactory abstractFactory1=
new ConcreteFactory1();
            List<XuanZe> list=abstractFactory1.
createProductA();
            return list;
        }
}
```

(3) 试卷生成工厂:

```
public abstract class AbstractFactory {
    public abstract List<XuanZe> createProductA();
}
```

(4) 易试卷生成工厂:

```
public class ConcreteFactory1 extends Abstract
Factory{
    private XuanZe xz=new XuanZe();
    @Override
    public List<XuanZe> createProductA() {
        List<XuanZe> list=xz.getAll();
        return list;
    }
}
```

(5) 选择题抽象类:

```
public interface AbstractXuanZe {
    public List<XuanZe> getAll();
}
```

(6) 易选择题:

```
public class XuanZe extends AbstractProductA {
    @Override
    public List<XuanZe> getAll() {
        Session session=HibernateUtils.getCurrent
Session();
        String sql="select * from xuanzeti where
nanyichengdu='易' ";
        SQLQuery query=session.createSQLQuery
(sql);
        query.addEntity(XuanZe.class);
        List<XuanZe> list=query.list();
        return list;
    }
}
```

4.2 运行结果

本文首先以试卷列表的形式展示生成的试卷. 如图 5 所示, 试卷因难易程度不同而题目编号各不相同.

所有试卷生成信息列表

试卷编号: 难易程度: 查找

序号	试卷编号	难易程度	选择题数	选择题分值	填空题数	填空题分值	选择题ID	填空ID	开始考试
1	002	难	3	10	3	10	34,35,36	65,63,64,	开始考试
2	001	易	3	10	3	10	47,40,43	62,60,54	开始考试

上一页 下一页

图 5 试卷列表

难度系数为易、难的选择题如图 6 所示.

在本项目中, 若用户需要试卷难易程度为“中等”的试卷, 只需创建新的“中等选择题”类、“中等填空题”类与“中等试卷生成工厂”类, 新建的类各自继承其父类, 重写父类的方法. “试卷生成工厂”以上的程序结构均不需要改变, 该过程充分体现了程序的灵活性. 因为该程序遵循“单一职责”原则, 即每个类只具有单一的功能, 当其他项目需要题目类, 如选择题类, 可以直接

使用该类及其子类, 即该程序具有可复用性. 新增“中等”难度的试卷如图 7 所示.

5 结束语

复合模式的优点: 本文中复合模式是指将抽象工厂模式嵌入到模板方法模式中从而形成的集成两者优点的新模式. 模板方法模式的反向控制机制保证了算法整体的稳定性, 反向控制机制的原理为: 父类调用子类, 子类对父类进行灵活的拓展. 抽象工厂模式的优点一: 它将具体的类进行了分离, 工厂对具体产品的创建过程进行了封装, 在不实例化的情况下创建产品对象. 抽象工厂模式的优点二: 产品族的替换方便, 抽象工厂以抽象的形式实现了一个完整的产品族的创建, 每个具体工厂生产配置不同的产品族. 具体工厂类只在创建抽象工厂对象的时候出现一次, 因此具体工厂的替换很容易. 复合模式集成了二者的优点, 对延迟到子类的算法步骤的实现进行了灵活的扩展, 保证了算法结构的稳定性, 又分离了具体的实现类, 使得系统的健壮性、灵活性以及可复用性得到了增强.

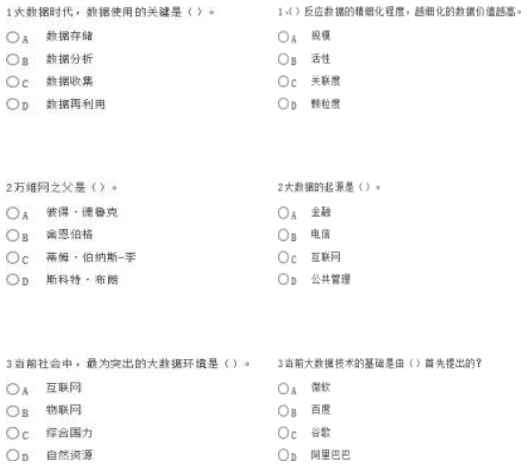

图 6 选择题题目

所有试卷生成信息列表

试卷编号： 难易程度： 查找

序号	试卷编号	难易程度	选择题数	选择题分值	填空题数	填空题分值	选择题ID	填空ID	开始考试
1	003	中等	3	10	3	10	38,39,37	7,8,6,	开始考试
2	002	难	3	10	3	10	34,35,36	65,63,64,	开始考试
3	001	易	3	10	3	10	47,40,43	62,60,54	开始考试

上一页 下一页

图 7 试卷列表

复合模式的缺点: 难以支持新种类的产品.

复合模式的适用性: 需实现算法的逻辑框架相同, 但每个步骤根据对象不同而实现的细节不同, 且该细节是从多个产品族中选取的一个系列进行实现.

参考文献

1 Gamma E, Helm R, Johnson R, 等. 设计模式: 可复用面向对象软件的基础. 李英军, 马晓星, 蔡敏, 等译. 北京: 机械工业出版社, 2007.

2 Gamma E, Helm R, Johnson R, *et al*. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley Publishing, 1995.

3 Martin RC. 敏捷软件开发: 原则模式与实践. 邓辉, 译. 北京: 清华大学出版社, 2003.

4 Shalloway A, Trott JR. 设计模式精解. 熊节, 译. 北京: 清华大学出版社, 2004.

5 Stevens P. Software design patterns. Computing & Control Engineering Journal, 2000, 11(4): 160–162.

6 韩万江. 软件工程案例教程. 北京: 机械工业出版社, 2007.

7 闫伟华. 模板方法模式的改进. 计算机应用, 2011, 31(S1): 135–137.

8 Dockins K. Template Method. Dockins K. Design Patterns in PHP and Laravel. Berkeley: Apress, 2017.

9 陈琴, 朱正强. UML 在设计模式描述中的应用. 计算机工程与设计, 2003, 24(4): 81–84. [doi: 10.3969/j.issn.1000-7024.2003.04.025]

10 Le Guennec A, Sunyé G, Jézéquel JM. Precise modeling of design patterns. Proceedings of the 3rd International Conference on the Unified Modeling Language: Advancing the Standard. Berlin, Germany. 2000. 482–496.

11 何成万, 何克清. 基于角色的设计模式建模和实现方法. 软件学报, 2006, 17(4): 658–669.

12 葛萌, 欧阳宏基. 工厂设计模式的研究与应用. 计算技术与自动化, 2017, 36(2): 136–140. [doi: 10.3969/j.issn.1003-6199.2017.02.029]

13 武光明. 设计模式在全球化软件开发中的应用. 计算机应用与软件, 2014, 31(1): 9–10, 17. [doi: 10.3969/j.issn.1000-386x.2014.01.003]

14 张璞, 夏英. 软件设计模式在 Java 程序设计课程教学中的应用研究. 软件工程, 2017, 20(7): 15–18. [doi: 10.3969/j.issn.1008-0775.2017.07.005]