第16卷 第5期
2006年5月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol.16 No.5
May 2006

物流信息系统中设计模式的应用

刘国静,余青松,郑 骏
(华东师范大学 计算中心,上海 200062)

摘 要:随着信息系统的不断发展,客户对软件系统提出了更高的要求。由于软件开发固有的复杂性、软件实现的不易复制性,软件的生产过程面临许多问题。设计模式提供了对问题簇的设计精良的解决方案。文中通过在物流信息系统中引入设计模式,具体分析了三种代表性的设计模式在物流信息系统中的应用,证明了模式设计的引入有助于提高软件的可复用性、可维护性以及稳定性和安全性。

关键词:设计模式;单例模式;适配器模式;策略模式;PDIS
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2006)05-0211-03

Application of Design Patterns for Logistics System

LIU Guo-jing, YU Qing-song, ZHENG Jun
(Computer Center, East China Normal University, Shanghai 200062, China)

Abstract: With the development of information system, customers bring out more requirements on software application. Due to complication of software development, a lot of problems raised in the process of the software application development. Design patterns provide well designed solutions for the problems. Based on the experience of a physical distribution information system (PDIS) development, this thesis thoroughly analyzes the three representative design patterns, its application in the logistics information system, and proves that using design patterns is contributive to reusability, easy maintenance, stability and safety.

Key words: design patterns; singleton pattern; adapter pattern; strategy pattern; PDIS

0 引 言

由于软件开发过程的复杂性、软件实现的不易复制性,软件的生产过程无法实现同传统企业一样的大规模生产。与此同时随着客户对于软件的认知趋于成熟,他们对软件质量提出了更高的要求:可复用性、易于维护、网络性能好、稳定、安全等等。另外,软件本身所处的环境也越来越复杂,比如分布式、Internet环境使编写软件要考虑更多更复杂的情况。在这种状况下,设计模式应运而生。

设计模式可以帮助程序设计师针对日常系统设计工作所遇到的很多设计问题给出结构合理、易于复用、易于维护的示范方案。在较复杂的应用系统中,如何使代码良好地组织起来,并封装成类,将类组织成系统性的类库,这都需要设计师在高于对象这一层次来把握整个系统,设计模式的综合运用正好适应了这一需要。

1 设计模式

1.1 设计模式

设计模式[1]是一个抽象的概念,它是针对面向对象系统中重复出现的设计问题,提出一个通用的设计方案,并予以系统化的命名和动机解释。它描述了问题、解决方案、在什么条件下使用该解决方案及其效果。它还给出了实现要点和实际运用的规则以及约束条件。该解决方案是解决该问题的一组精心安排的通用的类和对象,再经定制和实现就可用来解决特定上下文中的问题。

据研究,一个企业应用系统中,某个软件功能的60%~70%和其他软件的功能相似,40%~60%的代码在其他的软件应用中是可复用的,60%的设计在其他的软件应用中也是可复用的[2]。因此几乎所有软件应用的大部分都可以用预先定义的模式集成。通过在大规模的系统项目中实现设计模式,最大化了复用的潜在优势。并且,由于它们是封装了完整功能的独立单元,设计模式解决了可替代性问题。当业务发生变化时,可以很容易地利用满足新业务需求的构件替代旧的构件而几乎不会影响系统的其他部分。

1.2 设计模式的分类

设计模式依据其目的分为创建型(Creational)、结构型(Structural)和行为型(Behavioral)模式。创建型模式与对象的创建有关;结构型模式处理类或对象的组合;行为型模式针对类或对象怎样交互和怎样分配职责进行描述。

(1)创建型模式是对类的实例化过程的抽象化。一些系统在创建对象时,需要动态地决定怎样创建对象,创建

收稿日期:2005-08-22
作者简介:刘国静(1979-),女,甘肃兰州人,硕士研究生,研究领域为Web应用技术;余青松,高工,硕士,研究领域为计算机系统集成、Web应用技术。

哪些对象,以及如何组合和表示这些对象。创建模式描述了怎样构造和封装这些动态的决定。

创建型模式包括以下几种:简单工厂模式、工厂方法模式、抽象工厂模式、单例模式、多例模式、建造模式、原始模型模式等。

(2)结构型模式描述如何将类或者对象结合在一起形成更大的结构。结构型类模式采用继承机制来组合接口或实现。一个简单的例子是采用多重继承方法将两个以上的类组合成一个类,结果这个类包含了所有父类的性质。

结构模式包括以下几种:适配器模式、缺省适配模式、合成模式、装饰模式、代理模式、享元模式、门面模式、桥梁模式等。

(3)行为模式是对在不同的对象之间划分责任和算法的抽象化,描述了在运行时难以跟踪的复杂的控制流。它使用对象复合而不是继承,一些行为对象模式描述了一组对等的对象怎样相互协作以完成其中一个对象都无法单独完成的任务。

行为模式包括:不变模式、策略模式、模版方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、解释器模式、调停者模式等。

1.3 基于 Web Service 的 MVC 模式

MVC 模式即模型 - 视图 - 控制器(Model - View - Controller)模式[3]。其结构图见图 1。

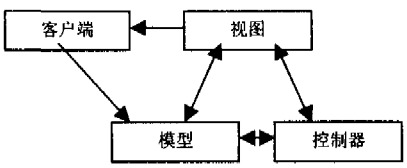

图 1 MVC 构架模式

其中,模型用来封装核心数据和功能;视图给用户显示信息;每个视图都有一个相关的控制器组件。它接受鼠标移动、鼠标点击或者键盘的输入等事件,事件被转换为服务请求,而服务请求又被传送给模型或视图。文中论述的物流信息系统是基于 J2EE 体系架构完成的。

2 设计模式在物流信息系统中的应用

物流信息系统通过对于物流相关信息的加工处理来达到对物流、资金流的有效控制和管理,并为企业提供信息分析和决策支持的人机系统。它具有实时化、网络化、系统化、规模化、专业化、集成化、智能化等特点。文中论述的物流信息系统采用 Sun One 体系下的 J2EE 解决方案,构建基于 Web Service 的 MVC 模式下。由于物流信息系统是个比较庞大的系统,涉及内容很多,限于篇幅,这里仅就前面所提到的其中几种设计模式分析一下设计模式在物流信息系统中的应用。

2.1 单例模式(Singleton)的使用

作为对象的创建模式,单例模式确保某一个类只有一个实例,而且自行实例化并向整个系统提供这个实例。这个类称为单例类[5]。系统中某些类只有一个实例,这个实例应当在程序启动时被创建,并且只有在程序结束时才被删除。它的特点是:①该类只有一个实例;②它必须自行创建这个实例;③它必须自行向整个系统提供这个实例。

在物流信息系统中有一些配置常量,这些常量如果是存储在程序内部的,那么每一次修改这些常量都需要重新编译程序,将这些常量放在配置文件中。系统要取得配置常量,就可以通过修改这个配置文件而无需修改程序便达到更改系统配置的目的。属性是系统的一种"资源",应当避免有多于一个的对象读取,特别是存储属性。此外,属性的读取可能会在很多地方发生,所以,属性管理器应当创建自己的实例,并且自己向系统提供这一实例。单例模式正好符合这些要求。其 UML 类图如图 2 所示。

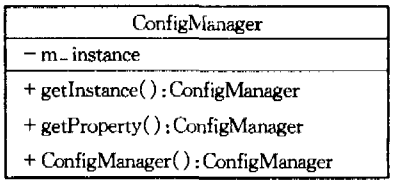

图 2 ConfigManager 类图

在属性管理器类 ConfigManager 中,一段关键代码如下:

```
private ConfigManager()
{ //初始化代码 };
private static ConfigManager m_instance = new ConfigManager ();
synchronized public static ConfigManager getInstance()
{ return m_instance;}
```

在这段代码中,ConfigManager 类的构造器被声明为 private,这保证了外界无法直接实例化,避免外界利用构造器直接创建任意多的实例。当这个类被加载时,静态变量 m_instance 会被初始化,此时类的私有构造器被调用,ConfigManager 类的惟一实例就被创建起来了,如果需要调用这个实例,只需调用 public 的 getInstance()方法即可,这也是取得 ConfigManager 类实例的惟一方法。

通过单例模式在物流信息系统中的应用,得到单例模式的如下优点:

● 对惟一的实例提供受控访问。因为单例模式封装它的惟一实例,所以它可以严格控制客户怎样以及何时访问它。

● 缩小名字空间。单例模式是对全局变量的一种改进。它避免了那些存储惟一实例的全局变量污染名字空间。

● 允许对操作和表示的精化。单例模式可以有子类,而且用这个扩展类的实例是很容易的。软件工程师可以用所需要的类的实例在运行时配置应用。

2.2 适配器模式(Adapter Pattern)的使用

适配器模式[4]把一个类的接口变换成客户端所期待的另一种接口,从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作。

在物流系统中商品的价格经常变动,而且计算商品价

格的方法也经常变动。为此,这里在原有计算商品价格模块的基础上,设定商品的标准价格,分为客户、门店对促销、赠品三部分区分管理,并对促销和赠品进行审核。促销和赠品可以设置期限,在期限内的促销和赠品不需要审核,只在促销和赠品期开始和结束时各审核一次。价格设置时按某一种包装计量类型输入,其他计量类型下的价格按入数折算。进价为采购的加权平均价。这里希望尽可能少地修改程序,以便能够利用已经存在的计算程序,同时必须考虑到将来便于采用别的方式替代目前的计算方法,因此采用适配器模式。UML 类图如图 3 所示。

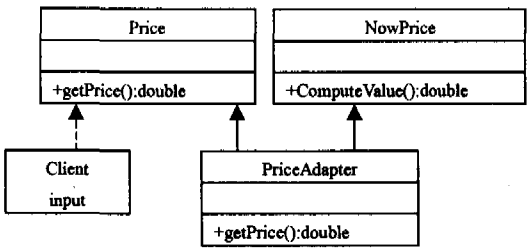

图 3　商品价格类图

在此,为包含计价函数的类命名为 Price 类,其中:

getPrice(double originalPrice, double discountPercent, double days)是 Price 这个抽象类定义的一个方法。系统已经存在的应用程序的相应功能由 NowPrice 类的 ComputeValue(double days, double discountPercent, double originalPrice)方法实现。为使客户端能够使用 Nowprice 类,提供一个中间环节,即类 PriceAdapter,把 Nowprice 的 API 与目标类 Price 的 API 衔接起来。PriceAdapter 类是从 Nowprice 类中继承而来的,这就决定了这个适配器模式是类的。Java 代码如下:

```
public class PriceAdapter
  extends NowPrice implements Price{
    NowPrice legacyAdapter = null;
    //类的构造器…
    /* * 下面的方法使用了现存类的 ComputeValue 方法 * */
    double getPrice(double originalPrice, double dicountPercent, double days){
    return legacyAdapter.ComputeValue(days, discountPercent, originalPrice);
    }//end of method getPrice()
}//end of class PriceAdapter
```

新的应用程序代码编写时使用的是 Price 对象,而在运行时也支持 PriceAdapter 对象。在图 4 所示的对象模型图中,PriceAdapter 对象同时也是 Price 对象。例如,应用程序中有这样一个方法,它的参数类型为 Price: void executePrice(Price aPrice);

这句代码可以按下面的声明执行:

executePrice(new PriceAdapter());

所有对 Price 的 getPrice()的调用都将转为对已有函数 ComputeValue()的调用。

同样让应用程序改用另一个类重新实现 Price()方法也很简单,只需要修改 PriceAdapter 的代码,应用程序的其余部分则不会受到影响。

通过适配器模式在物流信息系统中的应用,总结使用适配器模式的优点如下:

* 适配器模式提供了集成已有软件的一个普遍方法,可以将已有软件的接口适配成符合目标系统接口规则的接口,无缝地集成了已有软件。

* 为以后扩展预留了一个友好的接口。如果在开始就有意识地使用适配器模式建模,易于将来扩展。

* 为异构系统之间数据通信提供了统一的方式。异构系统最大的问题就是数据结构的不同,而适配器模式具有将这些数据结构统一为用户所需要的数据结构的功能。

2.3　策略模式的应用

策略模式[5]的目的是针对一组算法,将每一个算法封装到具有共同接口的独立的类中,从而使得它们可以相互替换。该模式使得算法变化而不影响到客户端。

物流信息系统中,商品价格的折扣有多种方式,折扣的算法根据客户的需要也经常变化,因此该模块可以采用策略模式构建。该模式把行为和环境分割开来。环境类负责维持和查询行为类,而具体策略类中提出各种算法。由于算法和环境独立开来,算法的增减和修改都不会影响到环境和客户端。其 UML 类图如图 4 所示。

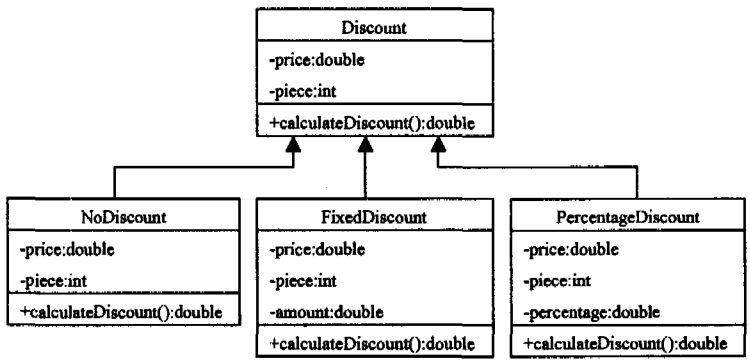

图 4　折扣算法类图

其中,Discount 类是一个抽象类,它定义了一个类型等级结构,由于本类是一个抽象类,不可能有实例,本类给出的接口是供具体子类实现的。三个具体折扣类都继承自 Discount 抽象类,NoDiscount 类表示商品没有折扣,FixedDiscount 类表示固定数目折扣,PercentageDiscount 类表示按百分比折扣。Discount 类提供了对外一致的接口,而在具体折扣类中实现具体算法,这里可以添加任意多的折扣算法,但必须保证继承自抽象类 Discount 类,并且实现 Discount 类中的 calculateDiscount()方法。

从上面适配器模式在物流信息系统中的应用,总结使用适配器模式的优点如下:

* 策略模式提供了管理相关算法簇的办法。策略类的等级结构定义了一个算法和行为簇。恰当使用继承可

(下转第 216 页)

折叠种子由线性反馈移位寄存器(LFSR)在折叠控制器的控制下按折叠关系展开成折叠序列[5],再经相移器(FSM)、循环扫描寄存器(CSR)和扇出结构进行多扫描链解压,解压后的测试序列(包含原测试集)同时送入各核的测试扫描链,进行扫描测试。

3 实验结果与分析

本方案采用 ISCAS-89 和 ISCAS-85 组合逻辑电路部分做了一系列实验,其中,仅对那些在 10000 随机模式应用之后,仍然存在不可测故障电路进行了分析,在实验中,将每两个电路统一优化考虑,给出了选取不同的扫描链个数时,本方案的实验结果如表 1 所示;并将最好的结果与近年来国际上较好的混合码压缩方案[2]的实验结果进行了比较,两者在同样条件下的实验结果如表 2 所示。

表 1 文中建议方案的实验结果

电路	$m=64$		$m=128$		$m=200$		$m=300$	
	总存储位数 bits	压缩率%	总存储位数 bit	压缩率%	总存储位数 bit	压缩率%	总存储位数 bit	压缩率%
C2670 C7552	12208	76.88	8136	84.59	8136	84.59	…	…
S5378 S9234	7980	89.91	5992	92.43	5992	92.43	…	…
S13207 S15850	1804	99.18	2190	99.03	1464	99.34	1416	99.36
S38417 S38584	69888	97.37	67925	97.44	49770	98.13	55860	97.90

表 2 文中建议方案与混合码方案[2]的实验结果比较

电路	硬故障测试集(bits)	混合码压缩方案[2]		文中建议压缩方案		Ⅱ/Ⅰ
		压缩后存储位数(bits)	压缩率Ⅰ%	压缩后存储位数(bits)	压缩率Ⅱ%	
C2670 C7552	52799	10178	80.72	8136	84.59	1.048
S5378 S9234	79104	14824	81.26	5992	92.43	1.137
S13207 S15850	384248	18052	95.30	1416	99.36	1.043
S38417 S38584	2658240	84628	96.82	49770	98.13	1.014

由实验结果可以看出文中建议方案明显优于混合码压缩方案[2],其平均压缩率比混合码压缩方案[2]高出 5 个百分点。文中在实验中仅仅是将两各电路统一考虑,若将多个电路统一考虑,压缩率将会更高。同时,文中方案由于多核共享而节省了大量用于测试的硬件开销。

4 结 论

文中提出了一种 SOC 芯片内部多核共享测试数据的数据压缩与解压方案。该方案利用相容技术和折叠技术,将 SOC 芯片中多个芯核的测试数据整体优化压缩和生成,并且能够实现多个芯核的并行测试,具有很高的压缩率;同时由于是多核共享,所以节省了很多硬件开销。文中建议的方案结构简单、解压方便、硬件开销低,实验证明是一种非常好的 SOC 芯片的 BIST 方案。

参考文献:

[1] 梁华国,蒋翠云.基于交替与连续长度码的有效测试数据压缩和解压[J].计算机学报,2004,27(4):548-554.

[2] Wuertenberger A, Tautermann C S, Hellebrand S. A Hybrid Coding Strategy For Optimized Test Data Compression[A]. Proceedings IEEE International Test conference[C]. Charlotte, NC, USA:[s.n.],2003.

[3] Hellebrand S, Rajski J, Tarnicks, et al. Courtois: Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers[J]. IEEE Trans. on Comp.,1995,44(2):223-233.

[4] Liang Hua-Guo, Jiang Cui-yun. Mixed Mode BIST Using Bi-Seed Copression[J]. Journal of Computer Research and Development,2004,41(1):215-220.

[5] Liang Hua-Guo, Hellebrand S, Wunderlich H J. A deterministic BIST scheme based on reseeding of folding counter[J]. Journal of Computer Research and Development,2001,38(8):931-938.

(上接第 213 页)

以把公共的代码移到父类里面,从而避免重复的代码。

* 策略模式提供了可以替换继承关系的办法。
* 使用策略模式可以避免使用多重条件转移语句。

3 结束语

信息技术的高速发展是有目共睹的。如今软件企业的数量正急剧增长,软件开发项目不仅数量越来越多,而且规模也越来越大。程序设计与编码的复杂性和工作量也越来越大。此时,设计模式显示了它的优越性。文中通过在物流信息系统中引入设计模式,从实践证明了设计模式的有效应用可以提供一个已经得到实践验证的设计框架,有助于软件的开发,提高软件的稳定性和可维护性,使软件工艺有基础性的质量保证,缩短项目开发周期和降低了开发成本。

参考文献:

[1] Gamma E, Helm R, Johnson R, et al. Design Patterns-Elements of Reusable Object-Oriented Software[M]. 北京:机械工业出版社,2002.

[2] Fowler M, Beck K, Brant J, et al. Refactoring: Improving the Design of Existing Code[M]. [s.l.]:Addison-Wesley,1999.

[3] Buschmann F, Meunier R, Rohnert H, et al. 面向模式的软件体系结构:模式系统[M].北京:机械工业出版社,2003.

[4] 阎 宏. Java 与模式[M].北京:电子工业出版社,2002.

[5] 胡剑鸿.基于 Web Service 的企业级应用中设计模式的研究与实现[D].上海:华东师范大学,2004.