第25卷 第3期 计 算 机 工 程 1999年3月
Vol.25 № 3 Computer Engineering March 1999

·软件技术研究与实现·

中图分类号 TP31

基于设计模式的软件重用

许幼鸣 徐 锦 赵文耘 钱乐秋
(复旦大学计算机系 上海 200433)

摘要 针对软件重用，提出应用设计模式来记录软件设计知识，这可以使软件重用从构架重用提高到软件开发各阶段知识的重用。此外结合通用仓库/销售系统的实际对设计模式的用法进行了讨论。

关键词 设计模式 软件重用 设计知识

Design Pattern Based Software Reuse

Xu Youming Xu Jin Zhao Wenyun Qian Leqiu
(Department of Computer Science, Fudan University Shanghai 200433)

【Abstract】 In order to achieve widespread reuse of software, we introduce using design pattern to record software design knowledge because it can improve software reuse from architecture level to every phases of developing. We discuss the way to use design pattern through an practical effort in our Universal Storage/Sale Management System.

【Key words】 Design pattern; Software reuse; Design knowledge

软件重用不仅指程序的重用，也包括对软件开发知识的重用。软件开发知识包含了设计知识，软件开发者也越来越认识到它在软件系统开发中的重要性。

1 设计模式

设计模式是记录设计知识的一种特殊方式，其目标是使那些在特定环境中良好工作的设计得以在相似的环境中被其他人再一次应用。不管对经验老到的开发者还是对新手，设计模式都可以帮助他们识别那些可以重用或应该重用设计的环境。

简单地说，设计模式是在具体环境中处理问题的方法。设计模式的基本结构包括模式名、问题说明、出现问题的环境、解决问题的方法和一些相关的信息等等。这种形式包括结构化的叙述和一些草图，如OMT图和对象关系图。使用设计模式可以清楚地描述设计知识，使其让更多的开发人员共享。

要达到对复杂软件构件的广泛重用，必须将精力集中在寻找软件中潜在的基本设计模式上。在面向对象设计模型的层次之上，设计模式向开发者描述构件结构和构件间的合作关系，从而帮助开发可重用软件。如果对软件系统设计和实现中潜在的设计模式没有彻底的了解，软件重用在很大程度上仍然不能成功。

2 设计模式的应用

在以往的开发中，即使是相似的领域，一旦有新的应用就不得不开发新的系统，已经完成的程序不能重用，造成极大浪费。使用已经成熟的设计模式来开发可重用的系统，并将设计模式贯穿于系统开发的整个过程，就可以避免上述问题。

我们在开发通用仓库/销售管理系统中就大量使用了设计模式。通用仓库/销售管理系统的核心思想是开发一个允许改变仓库特性，针对不同仓库结构只要做一些扩充就能正常运行的可重用、并具有良好维护性能的系统。由于不同类型的仓库具有不同管理模式及应用，因此构件级的重用已较困难。只有通过设计模式的应用，才能得到一系列可维护性较高的仓库管理系统。

本节以通用仓库/销售管理系统为例实现设计模式的应用。

2.1 层次结构

层次结构由一些由高至低顺序排列的层组成。某一层中对象的行为一般由同一层或低层的对象行为来表示。低层对象通常向高层对象提供服务。

在OO程序设计广泛使用之前，层次概念已经在很长时间内是软件设计的一个特征了。在可移植系统中基于层次的应用占主导地位，比如ISO/OSI七层协议和近来的Windows NT。这种系统包括一个或更多独立于平台的层次，用来向高层提供与平台无关的服务。这使得居于系统高层的应用可以用可移植的方式来实现。

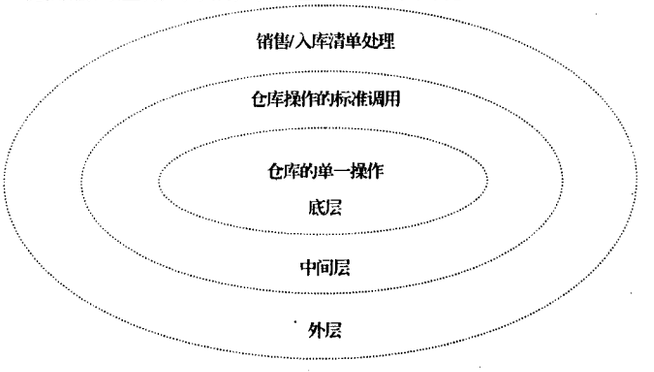

图1 仓库/销售管理系统的层次结构

在仓库/销售管理系统的设计中采用了层次模式，该层次模式的结构如图1，描述了整个系统中涉及仓库的操作部分。仓库/销售管理系统的层次结构将涉及仓库的操作部分分为3层：最高层是可移植的对象，提供通用的销

*许幼鸣 男，26岁，研究生，主攻方向：软件工程
收稿日期：1998—05—20 修回日期：1998-07-04

售/入库清单处理，可以使用低两层提供的服务。清单引起对仓库的各种操作，清单的录入和修改等都涉及仓库中项目数量的变化。中间层称为桥层，使用从最底层得到的操作特性，向可移植对象提供与具体仓库无关的入/出库操作。这种与仓库无关的操作的接口是标准的，由于使用标准接口，因此一旦最底层仓库特性有所改变，最外层也不会随之改变，从而达到重用的目的。最底层是仓库访问层，将仓库和改变仓库库存的操作封装起来，对仓库提供直接的操作。

这样的层次结构给最外层的开发者一个选择：全部使用中间层提供的与仓库无关的入/出库操作编制可移植的清单处理模块，或者直接使用仓库访问层来编写非移植性程序，以便提高效率，或适应特定仓库的特殊要求。当然，如果没有特殊要求，严格遵守调用的层次顺序是必要的。因为有层次顺序的调用有更清晰的结构，所以有更好的维护性。

2.2 适配器(adapter)

适配器模式的主要部分是一个对象，它基于另一个对象提供服务，向客户提供一个接口。适配器模式通常用于按照新的应用要求为已存在的类匹配接口。我们在系统中使用该模式时有一些变化，利用适配器模式为已存在的应用匹配新类的接口，这样已存在的应用不需要修改或者仅仅做一点点修改。这里已存在的应用是指清单处理，新类表示一个新的仓库类需要的新操作。

比如，一般仓库的增加库存原子操作addstore()只要在对应项目上增加库存数量就可以了。但有一种新仓库需要记录销售库存，那么增加库存就有不同的操作。这种仓库记录两种库存，一种是销售库存，记录未开出清单的存货数量，销售库存不能为负数，目的是保证开出的清单都能提到货；另一种是实际库存，记录仓库的实际库存数量。这时增加库存的操作就包括两个动作，增加销售库存及增加实际库存，并且两个动作是不可分的，必须同时完成或同时撤消，因此它们构成了一个原子操作。当增加这种新的仓库类时，也同时增加了新的原子操作。使用适配器模式后，只要修改或添加适配器类就可以了，而不会影响清单处理。

2.3 桥模式(bridge)

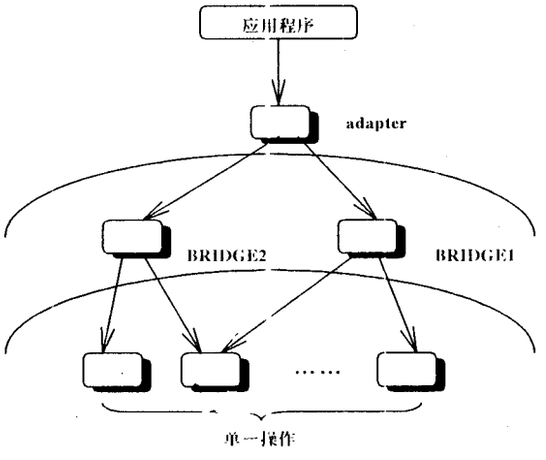

图2 桥模式

层次结构中的中间层大体上由桥模式风格的对象组成。这些桥对象对系统提供的可移植性是很重要的，因为它们为系统高层的可移植对象和具体仓库之间提供了与仓库无关的接口。桥对象负责使用特定仓库的单一操作实现一组独立于仓库的功能集，一个adapter需要这个功能集来实现它的控制。

adapters实际上通过使用合适的桥来实现，抽象超类adapter-Policy的不同子类提供这些方法在不同仓库合适实现。在中间层的桥层中bridges与adapters之间的关系如图2。

2.4 策略模式(strategy pattern)

策略模式允许封装一个算法，支持动态改变算法的某一部分。也就是说，策略模式允许在一个单独的接口中定义算法的多种实现，使得算法的改变独立于使用该算法的用户。一个对象可能根据所接收消息的发送者来选择自己的回答。Receiver类有一个方法m，代表receiver接收到消息后作出的回答。strategy类用来提供方法m的不同实现，每一个strategy子类代表一个具体的特定实现。receiver根据消息发送者选择不同的回答。

策略模式在设计阶段和实现阶段都可以使用。我们在通用销售/仓库管理系统的开发设计中就使用了策略模式。以仓库管理系统的入库操作为例，不同的仓库有不同的入库方式。仓库是一个对象，入库操作调用仓库的一个原子操作addstore()，该操作用来增加库存，不同的入库方式就调用不同的原子操作。这里共有3种增加库存的原子操作：

1) 常用方式。输入进货项目类型、数量、单价。为对应的仓库项目增加库存，只需增加数量。该操作假定项目已存在，如果单价不同作为新项目也已加入仓库中。这个假定在其他的原子操作中保证。

2) 平均单价方式。输入进货项目类型、数量、总金额。为对应的仓库项目增加库存，总金额，用金额和数量重新计算单价。这里仓库项目类型相对固定。

3) 销售库存方式。适用于同时记录销售库存和实际库存的仓库。基本操作与常用方式相同，但需要同时增加仓库项目的销售库存。

可以看到，使用策略模式有利于系统的重用。当增加一种入库方式时，原来的设计不会被影响，只须增加一种处理方法就可以了。并且策略模式将Store类封装，增加入库方式也不会影响Store类的结构。事实上，上述的原子操作都是由上一节的桥来完成的。

2.5 状态模式

状态模式允许一个对象根据自身的内部状态改变自己的行为。一个对象在不同的状态下可能对同一个消息作出不同的反应。状态模式的本质是在一个对象的属性值和它的行为之间定义了一种关系，这需要通过动态模型和对象模型在概念层次上的联系来建模。在通用销售/仓库管理系统中，清单就是一种有状态的对象。下面分别介绍清单的动态模型和对象模型。

图3是清单对象的状态变迁图。一张清单有3种状态，待校对状态(cardbooked)，已校对状态(cardchecked)

(下转第36页)

滤器(Filter)与之相连。发出的事件都必须先经过过滤器的过滤，符合条件后方可发送出去。

在HMMP客户服务环境中，客户方可以向HMMP服务方登记当某一特定条件发生时，可以从服务方获得特定事件的通知，这种事件就称为指示信息(Indications)。HMMP支持确认和非确认的指示信息的传送。对于登记和指示信息的传送都要附加上相应的安全认证。

消费者向生产者注册事件的步骤如下：

首先生成一个"__HMMPEventConsumer"对象。然后生成一个"__EventFilter"类的实例，并且设置合适的过滤查询。再生成一个"__FilterToConsumerBinding"类的实例，该类用于将一个事件与一个过滤器捆绑在一起。

生产者生产事件的过程：

当一个事件实例发生时，生产者首先查找与这个事件类或其父类相联系的"__EventFilter"对象。然后对该事件实例执行"__EventFilter"对象中的过滤查询。如果过滤结果不为空，则产生相应的指示信息(indication)，并生成 "HmmpIndRequestPDU。

参考文献

1 Rose M T.The Simple Book：An Introduction to Networking Management. Prentice-Hall，1996：100-250

2 Todd S. HyperMedia Management Protocol，Protocol Operations. IETF Drafts，1997-06

3 Todd S.HyperMedia Management Protocol，Protocol Encoding. IETF Drafts，1997-06

4 Todd S. HyperMedia Management Protocol，Query Definitions. IETF Drafts，1997-06

5 Todd S. HyperMedia Management Protocol，Security and Administration.IETF Drafts，1997-06

6 Todd S.HyperMedia Management Protocol，HMMP Events. IETF Drafts，1997-06

7 Common Information Model，Core and Common Schema. Desktop Management Task Force，Version 1，1997-04

(上接第14页)

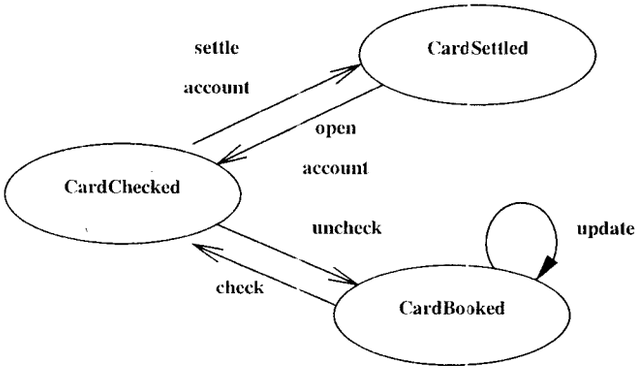

图3 业务清单的状态变迁

和结帐状态(cardsettled)。一张清单总是处于上述3种状态中的一种。有5种对清单的操作使清单对象在这3种状态中变换。待校对的清单可以修改，在校对后清单变为已校对状态，而已校对的清单就不能修改了；已校对的清单可以通过取消校对操作回到待校对状态以便修改，也可以经过结帐到达结帐状态。处于结帐状态的清单可以通过取消结帐操作回到校对状态。需要注意这里结帐操作和校对操作是有不同的，结帐是对本期间所有清单的统一操作，取消结帐则对上一期间的所有清单全部取消结帐，而校对与取消校对只是对某一张清单的操作。

3 结语

在设计模式中，层次模式、适配器模式、桥模式通常与整个系统的结构相关，策略模式和状态模式则和实现联系较多。在通用仓库/销售系统中，通过使用这些设计模式(当然也应用了诸如代理等其他设计模式)，形成了一系列仓库/销售管理系统，提高了设计知识的重用及目标系统的可维护性。

在使用设计模式的过程中，认识到设计模式提供了一种交流方法，使设计者能够迅速互相了解设计思想。设计模式使用一种简洁的方式获取设计中的基本思想，可以用来记录成功应用以便以后重用。通过应用，我们感到，对设计模式还可以从下列3个方面作进一步研究：

(1) 将设计模式与框架和其他设计方法结合

软件开发者会面临不少设计时的折衷。一个重要的问题是确定框架中的构件应该是可变的还是静态的。许多设计模式都试图减少软件中的静态部分。一些最有用的模式就是用来描述框架的。框架与模式间的一个区别是模式描述独立于语言的行为，而框架一般是基于某种特定的语言。我们需要研究模式与框架的结合。

(2) 模式语言

当使用模式的经验越来越丰富时，开发者需要将一组联系紧密的模式综合起来构造模式语言。模式语言定义结构化的风格指导设计者通过模式构造整个系统。一个模式语言可以生成软件系统，或者指导任何软件系统的组织、加工、人机界面和教学等。

(3) 将设计模式与当前的软件开发方法结合

在软件的生存期内，设计模式可以在许多阶段减少软件的复杂度。举例来说，设计模式可以帮助开发者在某个特定阶段查看各种可替代的选择。在分析和设计阶段，设计模式可以引导开发者选择那些被证明是成功的软件构架。同样，在实现和维护阶段，设计模式在高于源代码和模块模型的层次上描述软件系统的策略特征。设计模式还可以在领域分析和构架设计阶段和实现与维护阶段之间起桥梁作用。

参考文献

1 Pree W. Design Patterns for Object-oriented Software Development. Addison-wesley，New York，1994

2 Buschmann F，Meunier R，Rohnert H，et al. Patternori ented Software Architecture.Wiley &Sons，1996

3 Riehle D.Composite Design Patterns. OOPSLA '97 ACM SIGPLAN Notices,1997－10：218－227