第26卷 第3期 石家庄铁道大学学报(自然科学版) Vol. 26 No. 3
2013年9月 JOURNAL OF SHIJIAZHUANG TIEDAO UNIVERSITY (NATURAL SCIENCE) Sep. 2013

使用泛型技术消除观察者模式类型依赖

赵正旭， 张登辉， 刘 甜

(石家庄铁道大学 信息科学与技术学院,河北 石家庄 050043)

摘要: 观察者模式是一种在用户界面设计中广泛使用的设计模式。在传统的观察者模式实现中,虽然目标和观察者之间的耦合性得到降低,但是目标仍须知道观察者的类型,即所有的观察者必须处于同一继承体系下。在既有代码或第三方库中引入观察者模式,通常的做法是使用多重继承,这种方法不仅不利于对象间的解耦,而且在一些面向对象语言中,多继承功能是受限的。应用泛型技术在强类型语言中设计一种容器,消除模式中的类型依赖,在不改变观察者模式外部接口的前提下,让任意类型都可以充当观察者。在航天可视化遥操作子系统中采用这种方法对系统进行重构,提高了系统的可扩展性和复用性。

关键词: 设计模式; 观察者模式; 泛型技术; 面向对象

中图分类号: TP311 **文献标识码:** A **文章编号:** 2095-0373(2013)03-0048-06

0 引言

面向对象技术自1967年在Simulab67语言中被提出以后,有效地解决了传统结构化开发方法中客观世界描述工具与软件结构的不一致性问题,缩短了开发周期,解决了从分析和设计到软件模块结构之间多次转换映射的繁杂过程。面向对象技术是对自然界对象的抽象,但它并没有给出各个对象间有效的协作方法,也就无法有效实现软件的可复用性[1]。在面向对象技术的发展过程中,开发者针对特定问题提出了有针对性的解决方案,同时又对将来的问题和需求也有足够的通用性,从而避免重复设计。而这正是设计模式所要解决的问题。

设计模式[2]是将面向对象软件的设计经验以有效复用的形式记录下来。每一个设计模式系统地命名,解释和评价了面向对象系统中的一个重要的和重复出现的设计。设计模式使人们可以更加简单方便地复用成功的设计和体系结构。将已证实的技术表述成设计模式也会使新系统开发者更加容易理解其设计思路。设计模式有助于做出有利于系统复用的选择,避免设计损害了系统复用性。观察者模式属于设计模式中的行为型模式,它被广泛应用于界面系统设计中,可以有效降低软件系统中各个层次的耦合关系[3]。

泛型技术最先在C++中得到应用[4-5],以STL为首的众多C++库让泛型技术迅速崛起,泛型技术是一种新的编程模型。这种模式也影响到了Java和.NET语言。.NET从2.0版本开始就已经在虚拟机层面支持泛型技术,Java从1.5版已在语言层面支持泛型技术[6]。泛型技术扩大了代码可复用范围,它将类型信息的指定推迟到最终用户的使用中,泛型技术因此在框架设计中得到了广泛使用。

1 观察者模式

观察者模式定义了对象间的一种一对多的依赖关系,当一个对象的状态发生改变时,所有依赖它的对象都得到通知并被自动更新,许多用户界面系统将视图与下层的模式分离,模型与视图可以独自利用,也可以协同合作[3]。在模型视图框架中一个数据模型可以对应若干个视图对象,数据模型的状态发生任何改变时,所有视图都应得到通知。观察者模式描述了如何建立二者之间的这种关系。观察者模式的主

收稿日期: 2013-01-07
作者简介: 赵正旭 男 1960年出生 教授

要参与者是目标和观察者,一个目标并不依赖观察者的数量,一旦目标的状态发生改变,所有的观察者都会得到通知,作为响应,各个观察者将查询目标状态以保持数据的同步。

观察者模式结构图如图 1 所示,图中各个类协同合作,共同实现观察者模式的目标。

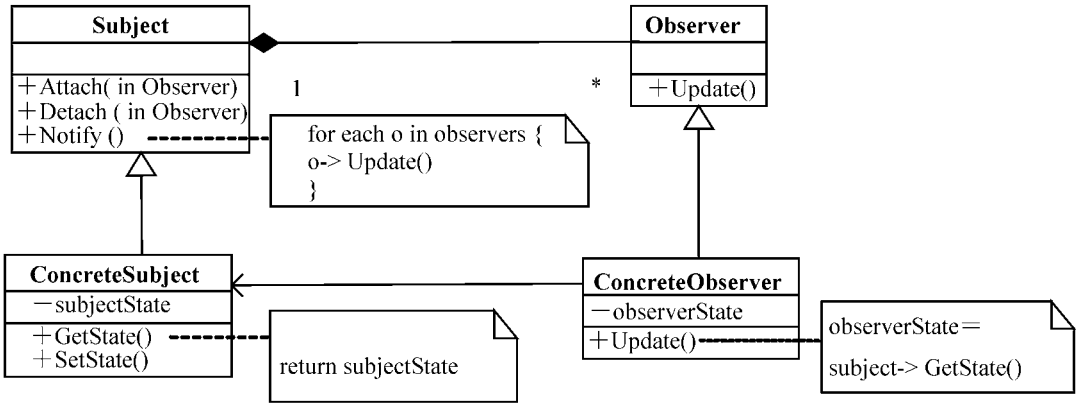

图 1　观察者模式结构图

(1) 目标 Subject 类,目标保存有观察者的引用集合,可以有任意多个观察者观察同一目标,目标同时提供注册与删除观察者对象的接口。

(2) 观察者 Observer 类,为那些在目标发生改变时需获得通知的对象定义一个更新接口,以供目标回调。

(3) 具体目标 ConcreteSubject 类,存储观察者感兴趣的状态,当状态发生变化时,向各个观察者发出通知。

(4) 具体观察者 ConcreteObserver 类,当具体目标对象改变时,具体观察者对象会得到通知,以使自己的状态与目标对象保持同步。

由观察者模式的结构图可知,观察者模式实现了目标与观察者之间的抽象耦合,目标只需要保存观察者对象的引用,而并不需要知道观察者所属的具体类型,观察者只需遵守模式的约定,实现观察者接口即可。这样目标和观察者之间的关系变为抽象的,并且耦合性得到降低,但所有的观察者仍必须有一个公共同基类。

2　设计与实现

为了提高软件的开发效率与软件的复用性,在系统的开发过程中常会引入第三方库或复用既有代码,而这也符合面向对象技术不重造“轮子”的思想[6-7]。对于这些既有代码,在复用其提供的功能时,常常还会加以调整以满足特定需要。在用户界面系统开发过程中,观察者模式适用场景非常广泛,而既有代码可能并不满足观察者模式各个类之间的协定。在观察者模式的传统实现中,虽然目标并不需要知晓所有观察者的类型,但所有观察者对象却必须保证实现观察者接口,即观察者必须处在同一继承体系下。为了让现有代码可以复用观察者模式。一个解决方案是引入多继承,即在继承已有代码的基础之上,同时实现观察者接口。但多继承并不被所有面向对象语言所支持,而且使用多继承也会增加开发者的负担,在使用既有代码的同时,每个观察者对象还需要继承一个不提供任何功能,只为实现约定的一个“桩”类。针对这一情况,有必要使用新的方法来消除观察者模式类型依赖。

泛型技术本质上是将类型也作为一种参数传递。在强类型语言中,会在编译期对参数或数据进行类型检查,这就限定了在使用方法或数据时必须为其指定具体类型,造成的一个后果就是相同的算法必须针对每个类型定义一个版本,对于某些语言,如 C++,可以通过宏来解决这一问题,但引入宏的一个负作用就是失去了类型检查的功能,导致类型错误只能在运行时发现。泛型技术解决了这一问题。在 STL 中,可以定义一个泛型数组 vector<T>,在使用 vector 时可以指定 T 的类型,编译器会执行强类型检测,这样在定义算法或数据时就无需指定具体类型,只需在使用时指定一个类型。在面向对象语言如 C++

和Java中,使用泛型技术实现了各种算法和容器。这种做法虽然有效地降低了类型依赖,但在使用这些容器或算法之前,仍必须指定具体类型,对于观察者模式,仍不能完全消除类型依赖。通过泛型技术,设计了一种新的容器来完全消除观察者模式中的类型依赖,使任意类型的观察者都可以向目标注册。

2.1 观察者角色的泛型实现

在观察者模式的通常实现中,所有具体观察者类都依赖Observer这一基类,通过泛型技术设计了一个类型无关的观察者容器类ObserverStub。ObserverStub有两个内部类EmptyType,Stub,以及一个成员函数m_content。EmptyType是一个"稻草人"基类,它不做具体工作,只为提供一个接口。Stub是EmptyType的子类,它实现了EmptyType的接口,并保存对真正观察者的引用,而它正是去除观察者模式类型依赖的关键所在。以下为观察者角色的具体实现。

```
//目标类的桩类
class ObserverStub
{
public:
//构造函数为模板函数,用于传递类型信息
template <typename T>
ObserverStub(const T& value)
:m_content(new Stub<T>(value))
{
}
//名称约定,给目标类提供一个回调函数
void update()
{
m_content->update();
}
private:
//纯虚类,定义接口
class EmptyType
{
public:
virtual void update()=0;
};
//桩类,保存真正的观察者
template <typename T>
class Stub:public EmptyType
{
public:
Stub(const T& value)
:m_stub(value)
{}
//通知观察者
void update()
{
m_stub->update();
```

```
}
private:
//真正的观察者,类型由外部决定
T m_stub;
};
//观察者桩类的引用
EmptyType*  m_content;
}
```

2.2 目标类的实现

与通常的观察者模式一样,目标类也有一个数组型成员变量,不同的是它提供的注册接口 attach 是一个模板成员函数,类型 T 会通过编译器的类型推导技术一步步传导到 Stub 类,并在 Stub 类中保有一个与观察者同类型的引用 m_stub。当目标状态发生变化时,观察者的桩类(ObserverStub) 会首先得到通知,最终 ObserverStub 会把更新消息传导至真正的观察者 m_stub,以下为目标类的具体实现。

```
//目标类
class Subject
{
public:
...
//状态更新时通知所有观察者
void notify( )
{
for ( unsigned int i = 0; i < m_stubVec. size( ) ;  + + i)
{
m_stubVec [i] - > update( ) ;
}
}
//注册方法,为模板成员函数,可接受任意类型对象的注册
template < typename T >
void attach( T o)
{
m_stubVec. push_back( new ObserverStub( o) ) ;
}
private:
//观察者引用数组
std: : vector < ObserverStub*  > m_stubVec;
};
```

通过泛型技术,可以向外部隐藏上述细节与封装技术,观察者只需和通常一样向目标类注册,而不必知晓目标类如何存储对观察者的引用。对观察者的唯一要求是实现 EmptyType 中的名称约定,当然这一名称不必为 update,而这是观察者必须付出的代价,因为作为观察者必须有一个方法来接收目标的状态更新通知。通过这一技术,在消除类型依赖的基础上,还保持了强类型语言的特点,观察者模式所有的参与者类型都会得到检验,而如果观察者类没有实现约定的方法,在编译就可以发现错误,避免了把类型错误推迟到运行期影响系统的运行。

3 航天可视化遥操作子系统中的应用

在航天遥操作可视化子系统中,因为存在大量的用户界面设计工作,所以广泛应用了观察者模式,同时为了提高系统的开发效率,在系统的开发过程中使用了第三方库。系统设计之初,在继承第三方库提供的既有功能类时,必须同时继承一个观察者基类。通过泛型技术对观察者模式进行了重构[8-10],如图2所示。

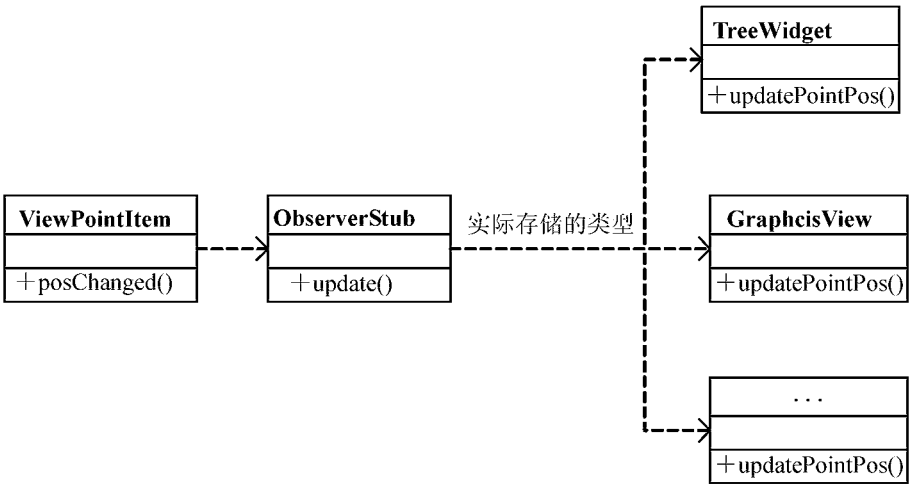

图2 引入泛型技术后的观察者模式

现在各个类如果想得到系统中二维点云(ViewPointItem)的位置变化通知,只需要提供一个接收通知的方法,各个类间不再有任何类型依赖。图中的TreeWidget等类都继承了第三方库中的既有对象,现在可以方便地使用第三方库,而不必再继承实现其它接口,同时观察者模式各对象仍和以前一样,在开发的过程中不必为这一重构付出代价,这样方法也有利于系统的扩展和新功能的添加,有新的类想接收目标的通知时,可以不再受类型约束。通过消除类型依赖,有效的提高了系统的可维护性和可扩展性。

4 结束语

GoF虽然总结了各种设计模式的经典实现方案,但这些方案无法适用于所有情形,针对系统或语言的特定情况,应灵活地使用设计模式,在复用设计模式减少系统重新设计的基础上,让前人的设计经验更有效地复用于系统的开发工作之中。

在航天可视化遥操作子系统的开发过程中,使用泛型技术对子系统中的观者模式进行了重构。实践表明,通过泛型技术消除观察者模式的类型依赖为系统带来了以下优点:观察者模式不必处在固定的类型系统中,其实现也变得整洁,易于维护。对于一些不支持多继承的语言,通过这一技术,仍可以在复用已有代码的基础上简单而优雅地使用观察者模式来解决问题。系统的可维护性和可扩展性也得到了提高。

参 考 文 献

[1] 陈叶旺,余金山. 泛型编程与设计模式[J]. 计算机科学,2006,33(4):253-257.

[2] Erich Gamma,Richard Helm,Ralph Johns,et al. Design patterns: elements of reusable object-oriented software[M]. BeiJing: China Machine Press,2002.

[3] 吴善明,沈建京,刘辉. 浅析Observer模式在GIS软件设计中的应用[J]. 计算机工程与设计,2007,28(18):4532-4535.

[4] 陈林,徐宝文. 基于源代码静态分析的C++0x泛型概念抽取[J]. 计算机学报,2009,32(9):1792-1803.

[5] Andrei Alexandrescu. Modern C++ design generic programming and design patterns applied[M]. 武汉:华中科技大学出版社,2002.

(下转第69页)

[5] 刘云霞,钟麦英. 基于等价空间的网络控制系统故障检测问题研究 [J]. 系统工程与电子技术,2006,28(10):1553-1555.

Fault Diagnosis Technology on DK-2 Pneumatic Brake System

Nan Jie[1], Huang Zhiwu[2]

(1. China Shenhua Track Mechanical Maintenance Company, Tianjin 300457, China;
2. School of Information Science and Engineering, Central South University, Changsha 410075, China)

Abstract: A parity space based method is proposed to diagnosis the pneumatic and electric faults for DK-2 brake pneumatic system with complexity and uncertainty. Firstly, the function models are built based on air fluid theory for the brake basic components such as the high-speed electric-pneumatic values and the mechanic pneumatic values; then the multiple brake operational modes are analyzed to obtain the relation of basic components in brake system, and the entire brake system model is constructed under the auto-brake, independent-brake and pneumatic-brake condition respectively. Based on it, the parity space-based fault diagnosis method is utilized to design residual generator and fault threshold, and the fault character matrix is constructed to relate the Boolean system state vector to fault information. The system fault is detected and isolated by researching the fault candidate set. The simulation presents the high efficiency and accuracy of the proposed method, which can meet the diagnosis requirement of DK-2 pneumatic brake system.

Key words: DK-2 Brake; fault diagnosis; system model; parity space relations　　(责任编辑　刘宪福)

(上接第 52 页)

[6] 赵正旭,龙瑞,郭阳, 等. 工程软件的小世界效应探究 [J]. 石家庄铁道大学学报: 自然科学版, 2010, 23(3):1-6.
[7] Guo Yang,Zhao Zhengxu,Zhou Yiqi. Complexity analysis with function-call graph on windows software [J]. International Review on Computers and Software, 2012,7(3):1149-1153.
[8] 王会进,陆裕奇,陈超华. 设计模式和泛型技术在系统重构中的应用研究 [J]. 计算机工程与设计, 2007,13(3):725-728.
[9] Joshua Kerievskya. 重构与模式 [M]. 杨光,刘基诚,译. 北京: 人民邮电出版社,2010.
[10] 殷定媛,高建华. 软件重构中 Visitor 设计模式和应用 [J]. 计算机工程与设计,2006,27(24):4817-4820.

Eliminating Type Dependence in Observer Design Patterns by Generic Technology

Zhao ZhengXu, Zhang DengHui, Liu Tian

(School of Information Science and Technology,ShijiaZhuang Tiedao University,Shijiazhuang 050043,China)

Abstract: Observer design pattern is widely used in the user interface design. Although the coupling between subject and observer gets lower in the traditional implementation of observer pattern, the subject still needs to know the type of the observer, i. e. , the observers have to be on the same inheritance system. To introduce observer pattern to the third party library, the usual practice is to use multiple inheritance which is not conducive to the decoupling between objects. For some object-oriented languages, support to multiple inheritance is limited. This paper proposes a container in the strongly typed language which eliminates the type dependence using the generic technology while not changing the object structure, and let any class have an ability to act as the observer. This method is used to refactor the Space Visual Remote Operation Subsystem,which improves the scalability and reusability of the system.

Key words: design patterns; observer; generic technology; object-oriented　　(责任编辑　刘宪福)