

MFC 框架中的设计模式分析

刘连喜 徐惠民

(北京邮电大学电信工程学院 北京 100876)

摘 要 分析 MFC 框架中的设计模式,有利于更好地理解、使用设计模式和 MFC。文章分析了 MFC 的 View/Document 结构中使用到设计模式,并详细介绍了各种模式在 MFC 中的实现方法。

关键词 MFC 设计模式 视图-文档框架

AN ANALYSIS OF DESIGN PATTERNS IN MFC FRAMEWORK

Liu Lianxi Xu Huimin

(School of Telecommunication Engineering Beijing University of Posts and Telecommunications Beijing 100876 China)

Abstract In this paper the design patterns used in MFC View-Document framework are analyzed. The principles and structures of the related design patterns are explained and the realization in MFC is described in detail.

Keywords MFC Design pattern View-Document framework

0 引言

设计模式是面向对象软件设计中已被证实了的设计经验的总结。利用设计模式可以更加简单方便地复用成功的设计和体系结构,帮助设计者更快更好地完成设计,并可以大大提高系统的可扩展性、可移植性,优化系统的设计结构。一般来说,一个模式有四个基本要素组成:模式名称、问题、解决方案和后果。

MFC(Microsoft Foundation Class)是微软推出的一套开发 Windows 平台软件的规模宏大的类库,是一套应用框架。之所以说 MFC 是一套框架,最重要的特征它所提供的 View/Document 结构能够将数据管理与显示分离。View/Document 是 MFC 的基石。

分析 MFC 框架中所使用的设计模式即有利于通过 MFC 的代码实例来理解设计模式,也有利于通过设计模式来理解 MFC 的内部机理,更好地使用 MFC。文章详细分析了 MFC 的 View/Document 结构中所用到的设计模式,并以 MFC 类库中提供的源码为例,阐明了各种模式在 MFC 中的实现原理,最后给出结论。

1 模板方法

模板方法是一种代码复用的基本技术。模板方法模式中,基类用一些抽象的操作定义了一个算法的骨架,子类重定义算法中的特定部分,以完成特定于子类的各种操作。模板方法模式的类结构如图 1 所示。

作为一种基本的代码复用技术,模板方法在 MFC 中得到大量的应用。如在 MFC 的源程序 VIEWCORE.CPP 中,类 CView(AbstractClass)对 Windows 消息 WM_PAINT 的相应函数 OnPaint(TemplateMethod)。

首先,通过宏语句 ON_WM_PAINT 将 WM_PAINT 消息

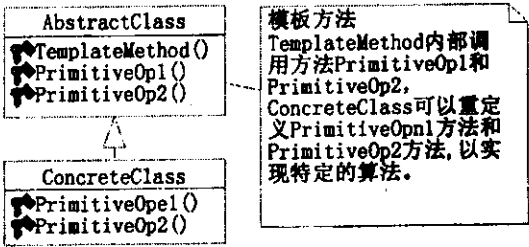


图 1 模式方法类结构图

的处理映射到函数 OnPaint。VIEWCORE.CPP 中的 OnPaint 定义了对 WM_PAINT 的处理骨架,如下所示:

```
void CView::OnPaint()  
{  
    // standard paint routine  
    CPaintDC dc( this );  
    OnPrepareDC( &dc );  
    OnDraw( &dc );  
}
```

OnDraw 被定义为纯虚函数,因此在生成代码框架时,AppWizard 为 CView 的子类自动生成了 OnDraw 函数。用户只需在 CView 的派生类中的 OnDraw 函数中编写代码就可实现数据的正确显示,而不必关心 OnDraw 如何被调用。OnprepareDC 函数可用于在 View 中显示数据前修改设备上上下文,或者打印时对打印机进行控制等。MFC 对 OnPrepareDC 提供了缺省实现,必要时,可以在派生类中重载该函数,实现用户特定的功能。

2 职责链

多个对象组成一个对象链,客户请求沿着对象链进行传播,

直到有一个对象处理它。在职责链中,发出请求的对象不明确哪个对象是请求的接受者,也不明确对象链的大小,链中的每一个对象都有可能对之进行处理。职责链中的对象大多按照从特殊到一般的顺序排列。类结构如图 2 所示。

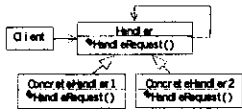


图 2 职责链类结构图

典型的对象链如图 3 所示。Windows 程序是消息驱动程序, MFC 使用职责链设计模式,将 Windows 的消息映射封装到了各个需要使用相关消息的类中。

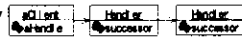


图 3 职责链中的对象链

在 MFC 中,所有派生自 CCmdTarget 的类都能够响应命令消息(WM_ COMMAND),所有派生自 CWnd 的类都能够响应标准 Windows 消息(除 WM_ COMMAND 之外的消息)。因 CWnd 派生于 CCmdTarget 类,故派生自 CWnd 的类也可响应命令消息。

MFC 使用消息映射表来将个消息与相应的处理函数对应起来。MFC 消息映射表是通过一组宏来实现的。实现消息映射表的类必须在其类定义(.h 文件)中调用以下宏声明消息映射表:

```
DECLARE_MESSAGE_MAP()
```

并在其实现文件中调用一组宏来实现消息映射表。比如,若要在视图类(假设工程名字为 Test,则视图类名为 CTestView)中将 WM_ CREATE 映射到函数 OnCreate,则应在视图类的实现文件中加入如下宏:

```
BEGIN_MESSAGE_MAP(CTestView,CView)
ON_WM_CREATE()
END_MESSAGE_MAP()
```

对于非 WM_ COMMAND 消息,消息沿着类的继承方向逆向传递,直到找到相应的处理函数为止。

对于 WM_ COMMAND,因与窗口无关的类也要能够处理,故有不同的传递路径。以单文档应用为例,当用户点击了按钮或菜单时,WM_ COMMAND 消息将会传送到 CFrameWnd 的子类 CMainFrame 的窗口过程函数(因这两个类都没改写 WindowProc,实际上为 CWnd 类的 WindowProc),然后被转发到 CFrameWnd 类的 OnCmdMsg 函数:

```
//WinFrm.cpp
BOOL CFrameWnd::OnCmdMsg( UINT nID,int nCode,void *
pExtra,
AFX_CMDHANDLERINFO *
pHandlerInfo)
{
    CPushRoutingFrame push( this );
    //pump through current view FIRST
    CView * pView = GetActiveView();
    if( pView != NULL &&
    pView->OnCmdMsg( nID,nCode,pExtra,pHandlerInfo))
        return TRUE;
    //then pump through frame
    if( CWnd::OnCmdMsg( nID,nCode,pExtra,pHandlerInfo))
        return TRUE;
    //last but not least pump through app
    CWinApp * pApp = AfxGetApp();
    if( pApp != NULL &&
    pApp->OnCmdMsg( nID,nCode,pExtra,pHandlerInfo))
```

```
return TRUE;
return FALSE;
}
```

由以上程序可知,消息将首先送到视图进行处理。若不能处理,则转回框架窗口,若还不能处理,则送到 CWinApp 对象处理。消息到达视图类后,视图类将首先检查自己的消息映射表,若没有相应的处理函数,则转入文档类进行处理。消息传送的对象链如图 4 所示。



图 4 WM_ COMMAND 消息传送链

3 观察者

观察者模式又称为发布—订阅模式,多个观察者与一个目标对象存在依赖关系,当目标对象改变时,所有观察者都得到通知并被自动更新。

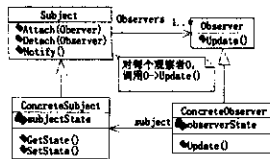


图 5 观察者模式类结构图

观察者模式中,目标和观察者分别封装在独立的对象中,目标可以不必清楚具体有多少个观察者,或具体的观察者对象是谁,只需知道观察者的抽象接口,从而两者可以独立地改变和复用。观察者模式的类结构如图 5 所示。

当 Subject 的状态发生改变时,目标对象调用 Notify(),通知与其关联的所有观察者。观察者得到通知后,调用目标对象的方法 GetState() 对自身状态进行更新。同时,观察者也可以主动改变目标的状态。

MFC 的 View/Document 结构的实现中采用了观察者模式。Document 为模式中的目标,管理应用程序中的数据,View 为模式中的观察者,以给定的方式显示所关联的 Document 中的数据。CDocument 类中定义了一个指针列表,用于保存对应的 CView 对象,并定义了一个函数用于对链表中的所有 CView 的对象进行更新。

```
//afxwin.h
class CDocument:public CcmdTarget
{
public:
    void UpdateAllViews( CView * pSender,LPARAM lHint = 0L,
        COBJECT * pHint = NULL );
    ....
protected:
    CPtrList m_viewList; //list of views
    ....
}
//DocCore.cpp
void CDocument::UpdateAllViews( CView *
pSender,LPARAM lHint,COBJECT * pHint )
{
    ASSERT( pSender == NULL || !m_viewList.IsEmpty() );
    //must have views if sent by one of them
    POSITION pos = GetFirstViewPosition();
    while( pos != NULL )
    {
        CView * pView = GetNextView( pos );
        ASSERT( !VALID( pView ) );
    }
}
```

```

if( pView != pSender )
    pView->OnUpdate( pSender ,Hint ,pHint );
}
}

```

在执行打开文件或新建功能后,UpdateAllViews 将会被自动调用,然后依次调用各个 CView 对象的 OnUpdate,进行显示更新。

4 桥 接

桥接模式用于将抽象部分与实现部分相分离,使它们都可以独立地变化。

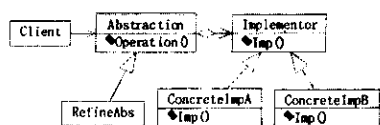


图6 桥接模式类结构图

使用桥接模式,抽象类的实现可以在运行时刻进行配置,一个对象甚至可以在运行时刻改变它的实现。接口与实现分离也有利于分

层,实现部分处理底层的实现细节,接口部分提供高层接口,并可单独进行修改、扩充。桥接模式的类结构如图6所示。

图中,右边的实现类 Implementor 实现较为低级的功能,可以只实现一个最小集,而由中间的接口类实现较为高级的功能。

MFC 中对对象的存取称为串行化。串行化设计中采用了桥接设计模式。MFC 使用 CArchive 类和 CFile 类(或 CFile 的派生类)配合实现对象的串行化。CArchive 实现了串行化时所使用的接口(比如重载操作符 >> 和 <<) ,而不关心数据保存在何处或何种介质上。CFile 或其派生类则具体实现了从各种介质中(比如内存、磁盘、网络等)读取数据的机制。

CArchive 在初始化时,通过构造函数的参数与一个 CFile 对象绑定起来,如下所示:

```

CArchive( CFile * pFile ,UINT nMode ,intnBufSize = 4096 ,
void * lpBuf = NULL );

```

MFC 框架中已经自动实现了文档的存取(使用模板方法),用户只需在其文档类中实现自己的 Serialize 函数即可。

CArchive 类和 CFile 类可以互不影响地单独进行修改,比如,CArchive 类可以派生出新的类以定义新的数据类型来重载 >> 和 << 操作符,实现新的数据类型的存取,或定义提供新的功能的函数,而 CFile 则可以派生出新的文件类,实现不同文件的读取和写入,比如数据库的存取。

5 单 件

单件模式保证一个类仅有一个实例,并且提供一个访问它的全局访问点。当一个类只允许有一个实例时可使用单件模式。

每一个 MFC 应用实例都派生于类 CWinApp。显然,每个应用程序都只应该有一个派生于 CWinApp 的实例。CWinApp 在设计中保证了一个应用程序不能生成多个实例,并且提供了一系列的函数用于对该唯一对象的一些属性的访问,包括 AfxGetApp, AfxGetInstanceHandle, AfxGetResourceHandle, AfxGetAppName 等。

MFC 是通过 ASSERT 来防止多次构造 CWinApp 对象的。在第二次构造 CWinApp 对象时,ASSERT 内的表达式为假,将会

弹出错误提示。而宏语句 ASSERT 只在 Debug 版本中有效,但采用 Release 版本编译时,会发现应用程序还是可能正确的运行,该实现并不完美。单件模式通常可以通过类的静态变量和静态函数来实现。

6 中 介

用一个中介对象来封装一系列的对象交互。中介模式将一系列对象间的多对多的通信转化为中介对象与各个对象的一对多的通信,从而使其耦合松散,而且可以独立的改变它们之间的交互。中介模式的类结构如图7所示。

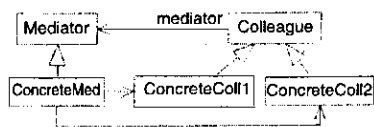


图7 中介模式类结构图

MFC 的对话框机制

使用了中介模式。对话框类作为中介类,对话框上的其它控件作为同事类。如编辑框、按钮、下拉列表框等。当对话框上的两个或多个控件需要通信时(如编辑框输入有效数据后使按钮使用),发送消息的控件首先将消息发送到它的父窗口(对话框),然后父窗口将收到的消息转发到各目标控件。

7 结束语

MFC 作为一套应用框架,使用了很多的设计模式。这些设计模式的使用,大大提高了类库的可复用性,可扩展性。比如,使用模板方法,开发人员只需对其感兴趣的方法进行重写即可,大大减少了开发人员的工作,并且增强了开发人员的灵活性。用职责链模式,将 Windows 复杂的消息处理机制分布到各个需要处理消息的类中,程序员只需使用几句简单的宏语句,即可实现 Windows 复杂的消息映射,大大简化了系统开发。

但 MFC 中独特的 RTTI(Runtime Type Identification)机制和动态创建功能减少了使用设计模式,特别是创建型模式的必要性。实际上,MFC 的对象的串行化机制和动态创建机制也可以认为是创建型模式中的抽象工厂模式和原型模式的变种,不过,MFC 不使用继承机制实现,而是通过一些复杂的宏语句实现。

除了 View/Document 框架中使用的模式外,在别的类中还包含了许多其他的模式,如 OLE 中大量使用了工厂模式,集合类模板库中大量使用了迭代器模式等。

但由于 MFC 的设计时间比较早和其复杂性,很多适合使用设计模式的地方却没有使用,或者不太明显,有的模式的实现也不是很合理。MFC 中宏语句的使用,大大减少了对对象继承的系统开销,但也减低了程序的可读性,灵活性。

参 考 文 献

- [1] Erich Gamma, Richard Helm 等著,李英军、马晓星等译,设计模式—可复用面向对象软件的基础,北京:机械工业出版社,2000.
- [2] 周之英,现代软件工程(下册),北京:科学出版社,2001.
- [3] 侯俊杰,深入浅出 MFC(第二版),武汉:华中科技大学出版社,2001.
- [4] T, Kulathu Sarma. MFC and Design Patterns. URL: http://www.codeproject.com/gen/design/chain_response.asp.
- [5] <http://search.csdn.net/expert/topic/50/5006/2001/6/15/159235.htm>.