四层应用软件系统的分析与开发

屈喜龙，赵慧娟
(西南交通大学 CAD 工程中心，四川 成都 610031)

摘要：从软件工程的角度提出了传统三层结构的应用软件系统的不足，分析了四层结构的应用软件系统架构.分别对三层与四层应用软件系统中的业务逻辑处理进行分析和对比，从而得出四层架构较传统的三层架构的优势所在，并在.net 平台下，开发出一个四层结构的真实应用软件系统.

关键词：net；业务外观层；业务规则层；数据访问层
中图分类号：TP311 **文献标识码**：A **文章编号**：1007－855X(2004)05－0071－05

Analysis and Development of Four－Layered Application Software System

QU Xi-long, ZHAO Hui-juan
(CAD Engineering Center, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: The defaults of the three－layered application software system ave put forward according to software engineering, and the four－layered structure of application software system is analyzed. The process of business logic of three－layered and four－layered application software system is analyzed and compared, then the advantags of the three－layered over the four－layered application software system are presented. Under the platform of.net, a real four－layered application software system is developed.

Key words: .net; Business Facade; business rule; DataAccess

0 引 言

三层的应用软件系统，由于其众多的优点，已经成为典型的软件系统架构，也已经为广大开发人员所熟知.在一个典型的三层应用软件系统中，应用系统通常被划分成以下三个层次：数据库层、应用服务层和用户界面层.

其中，应用服务层几乎集中了系统全部的业务逻辑的处理，因此，可以说是应用软件系统中的核心部分.软件系统的健壮性、灵活性、可重用性、可升级性和可维护性，在很大程度上取决于应用服务层的设计.因此，如何构建一个良好架构的应用服务层，是应用软件开发者需要着重解决的问题.

1 传统三层结构的缺陷分析

传统的三层结构在构建应用服务层的时候，把数据库操纵、业务逻辑处理甚至界面显示夹杂在一起，或者，把业务逻辑处理等同于数据库操纵.这些，都是有缺陷的做法.从软件工程的角度出发，应用软件系统的设计，必须遵循的最重要的原则就是高内聚和低耦合[1].软件分层的目的，就是提高软件的可维护性和可重用性，而高内聚和低耦合正是达成这一目标必须遵循的原则.降低系统各个部分之间的耦合度，是应用服务层设计中需要重点考虑的问题.内聚和耦合，包含了横向和纵向的关系.横向的内聚和耦合，通常体现在系统的各个模块、类之间的关系，而纵向的耦合，体现在系统的各个层次之间的关系.为了使应用服

收稿日期：2003－10－28. **基金项目**：国家 863/CIMS 主题资助项目(项目编号：2002AA414050).
第一作者简介：屈喜龙(1978.1～)，男，博士研究生. 主要研究方向：网络化制造系统研究. **E－mail**：quxilong@sina.com.

务层的设计达到最好的效果,所以,还需要对应用服务层作进一步的职能分析和层次的细分.

在Web应用程序中,有部分操作只是简单的从数据库根据条件提取数据,不需要经过任何处理,而直接将数据显示到网页上,以电子商务系统为例,比如查询数据库中某种商品的列表.而另外一些操作,比如计算订单中商品的总价并根据顾客的级别计算回扣等等,这部分往往有许多不同的功能的类,操作起来也比较复杂.可以想象一下,如果采用传统的三层结构,这些商业逻辑一般全部放在中间层,那么对内部的许多种类繁多,使用方法也各异的不同的类的调用任务,就完全落到了表示层,必定会增加表示层的代码量,且使业务逻辑处理与界面显示的代码夹杂在一起,将表示层的任务复杂化.这个与用户界面层只负责接收用户的输入并返回结果的任务不太相称,关键的是增加了层与层之间的耦合程度[2].

2 四层应用软件系统结构的分析

为了解决上述问题,提出了四层结构的应用软件系统模型,如图1所示.

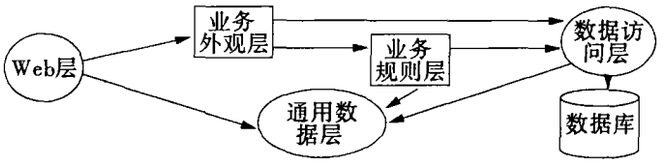

图1 四层应用软件系统的总体架构图
Fig.1 General structure diagram of four-layered application software system

这个四层就是Web层、业务外观层、业务规则层和数据访问层.通用数据层本身不能算作独立的层,它只是包含用于在各层间传递信息的数据集,我们从图中可以清楚的看到,浏览器首先调用的是表示层Web层,然后Web将请求发送给业务外观层,业务外观层对请求进行初步的处理,判断是否需要调用业务规则层,还是直接调用数据访问层获取数据.最后由数据访问层访问数据库并按照来时的步骤返回结果到浏览器.

2.1 各层的详细分析

这里对四层结构的软件系统的各层加以介绍和分析,尤其针对业务逻辑的处理,对其中的业务外观层与业务逻辑层加以详细分析.

Web层:Web层为客户端提供对应用程序的访问.

业务外观层:业务外观层用作隔离层,它将用户界面与各种业务功能的实现隔离开来.

除了低级系统和支持功能之外,对数据库服务器的所有调用都是通过此程序集进行的.

业务外观层的目的,是隔离系统功能的提供者和使用者,更明确地说,是隔离业务逻辑的软件的用户界面.这一层没有任何需要处理的逻辑,只是作为后台逻辑处理和前端用户界面的缓冲区,以达到如下目的:

- 将用户界面和系统业务逻辑处理分开,这样,当业务逻辑发生变化时,不用修改客户端程序,是一种支持变化的设计方法.
- 使同一个业务逻辑能够处理不同的客户端请求.
- 作为系统不同模块之间的调用接口.一个系统通常会包含很多模块,这些模块相对独立,又可能互相调用.为了减少各个不同部分之间的耦合度,所以,采用这种设计方法是必要的.
- 有利于项目团队的分工协作.业务外观层作为一个访问接口,将界面设计人员和逻辑设计人员分开,使得系统的开发可以实现纵向的分工,不同的开发人员可以关注自己的领域而不会受到干扰.
- 业务外观层的代码框架,在系统分析和设计完成后即开发完成.

业务规则层:主要负责验证从Web层传下来的数据和从数据库中提取返回给Web层的数据.

数据访问层:跟数据库真正打交道的一层,负责数据的增加、查询、修改和删除.

这样,Web层只需要复责表示和接收用户输入,而数据访问层只需要负责与数据库的简单交互,而业务规则层就只要负责验证数据的完整和有效性.可见,四层结构的软件系统比三层结构的软件系统有无可比拟的优势.

3　四层结构的真实应用软件系统的开发

为了对上述四层架构更清晰的分析和阐述.采用 Visual Studio.NET 作为开发工具，开发语言为 C#.开发一个真实系统.由于篇幅所限，这里只以一个简单的系统为例，就是实现对商品库的查询和添加.

商品的基本信息包括：商品编号(GoodsId，char，25，主键字)、订购商品名称(GoodsName，char，30)、订购商品数量(GoodsNumber，int，4)、商品生产厂商名称(GoodsManufacturer，char，35)、产地(GoodsMadedPlace，Char，20)、单价(GoodsPrice，Money，8)、计价单位(PriceUnit，char，10)和备注(Memo，varchar，1000).

这里，Web 层由 ASP.NET Web 窗体和代码隐藏文件组成.Web 窗体只是用 HTML 提供用户操作，而代码隐藏文件实现各种控件的事件处理.

商品的查询基本不需要什么逻辑处理，所以，业务外观层不需要调用业务规则层，而是根据由 Web 层传入的查询条件，直接调用数据访问层，把得到的数据集返回给 Web 层进行显示.但商品的入库，就有一些需要考虑的问题，例如，从 Web 层传入的商品信息中，某些字段是否为空，是否合法，如商品的数量不能是小数或小于等于零，等等，因此，需要调用业务规则层.

在.net 中，所有的文件都用类来封装的.首先，构建商品(GoodsData)类，也就是通用数据层的建造，下面是关键代码：

```
    public class GoodsData:DataSet
{    public GoodsData()
{
    DataTable table = new DataTable(GOODS-TABLE);
    DataColumnCollection columns = table.Columns;
    columns.Add(GoodsId,typeof(System.String));
    columns.Add(GoodsNumber,typeof(System.Int32));
    columns.Add(GoodsPrice,typeof(System.Decimal));
    columns.Add(Momo,typeof(System.String));
    ……
    this.Tables.Add(table);
    }
}
```

接下来看看数据访问层的代码实现，数据访问层中的(GoodsAccess)只负责有关数据的查询和存取.下面是这个类的关键代码：

```
public class GoodsAccess

{public GoodsData GetGoodsByCondition(String Condition,String Value)
    {
    GoodsData GoodsDataView = new DealerData();
    SqlConnection con = new SqlConnection(ConnectionString);
    SqlCommand com = con.CreateCommand();
    com.CommandText = "SELECT * FROM" + GoodsTableName + "WHERE" + Condition + " = '"
    + Value + "'";
    SqlDataAdapter sda = new SqlDataAdapter();
    sda.SelectCommand = com;
    sda.Fill(GoodsDataView,GOODS-TABLE);
    return GoodsDataView;
```

```
}//通过从 Web 层传来的查询条件查出相应结果以数据集形式传给上一层
    public bool InsertGoods(GoodsData Goods)
{
    SqlConnection con = new SqlConnection(ConnectionString);
    SqlCommand com = con.CreateCommand();
    foreach(DataRow myRow in Goods.Tables[Goods-Table].Rows)
    {
    com.CommandText = "INSERT INTO " + GoodsTableName + "";
    string comma = "";string prefix = null;string surffix = null;
    string properties = "";string values = "";
foreach(DataColumn myColumn in GoodsData.Tables[GOODS-TABLE].Columns)
    {
    if(myColumn.DataType = = System.Type.GetType("System.String"))
    {
    prefix = "'";//字段如果是字符串类型,前加'号
    surffix = "'";//字段如果是字符串类型,后也加'号
}
if(myColumn.DataType = = System.Type.GetType("System.Int32"))
    {
    prefix = "";//字段如果是数值型,前不加任何符号
    surffix = "";//字段如果是数值型后也不加任何符号
    }
    properties + = comma + myColumn.ColumnName;
    values + = comma + prefix + myRow[myColumn] + surffix;
    comma = ",";//comma 为逗号,分开各个字段值
    }
com.CommandText + = "(" + properties + ")VALUES(" + values + ")";
com.ExecuteNonQuery();
    }
    returntrue;//如果插入成功返回 true
    }
}
```

采用 DataAdapter 来将数据填充到定制的数据集中,然后返回该数据.底层都做到这份上了,那上层都做些什么呢? 答案是数据检查,上层基本上都在做一些很严密的数据合法性校验一些比较复杂的事务逻辑,先看业务外观层的类:GoodsSystem,其关键代码如下:

```
public class GoodsSystem
{    publicGoodsDataGetGoodsById(String IdStr)
    {
    GoodsAccess QueryAccess = new GoodsAccess();
    Return QueryAccess.GetGoodsByCondition("GoodsId",IdStr);
    }
    Public void AddGoods(GoodsData GoodsAdded)
    {
```

```
    Goods GoodsInsert = new Goods();
    GoodsInsert.VerifyGoods(GoodsAdded);
    }
}
```

其中,业务外观层的查询,直接调用数据访问层的 GetGoodsByCondition 方法,这里,假设 web 层传来的请求是根据商品的 ID 号查询,那么,我们在这一层传入的第一个参数就是"GoodsId",而第二个参数就是从 Web 层传入的 Id 号值.在这一层,还调用类 Goods 的 VerifyGoods 方法,下面对 Goods 类的关键代码:

```
public class Goods
{    public bool VerifyGoods(GoodsData GoodsAdded)
    {
    bool isValid = true;
    DataRowCollection itemRows = GoodsData.Tables[GOODS-TALBE].Rows;
    foreach(DataRow Row in itemRows)
    {
    Row.ClearErrors();
isValid & = IsNull(Row);//IsNull 函数判断不能为空的字段其值是否为空
isValid & = IsValidField(Row);//IsValidField 判断各个字段值是否为数据库中
    if(! isValid)     //对应的数据类型,且不超出长度范围
    {
    Row.RowError = "无效字段";
    Return false;
    }
    }
    GoodsAccess InsertAccess = new GoodsAccess();
    return InsertAccess.InsertGoods(GoodsAdded);
    }
}
```

4　结束语

当然,真实的系统中,系统的复杂程度远远不止如此,但是,通过这个简单的用例可以看出,四层结构确实增加了系统的内聚,减少了系统间的偶合程度,采用四层结构开发软件系统,与传统的三层结构相比,具有明显的优点和可借鉴之处.

参考文献:

[1] 张海藩.软件工程导论.北京:清华大学出版社,1992.59~62.

[2] http://www.csdn.net/develop/article/19/19702.shtm;csdn,2003,6.