C#. NET 下三层架构数据库应用系统开发

王海燕
(北京信息科技大学 机电学院,北京 100192)

摘 要:基于C#. NET的三层架构数据库系统由于其快速方便的开发模式、较高的可维护性和可重复性以及灵活的事务处理机制,在大型Web数据库应用系统开发中使用很广泛。为了解决C#. NET环境下三层结构数据库应用系统建立过程中的常见问题,按照自下而上的顺序结合实例论述了三层架构数据库应用系统的建立的标准流程,给出了因为数据库设计的变化而导致三层结构程序变化的修改方法,解决了最常出现且不易修改的编译错误,为Web数据库应用系统开发人员提供一定的借鉴。

关键词:三层架构;数据库应用系统;C#. NET
中图分类号:TP 315 **文献标识码**:A **文章编号**:1673-629X(2012)06-0078-04

Implementation of Database Application System Based on C#. NET Three-tier Architecture

WANG Hai-yan
(School of Electromechanical Eng., Beijing Information Science and Technology University, Beijing 100192, China)

Abstract: Database system based on C#. NET three-tier architecture has a wide range of use in large Web application system due to its rapid and easy development mode, high maintainability and repeatability and flexible transaction processing mechanism. It is aimed to solve the common problems that usually exit in the development of three-tier architecture database application system based on C#. NET. The standard process of three-tier architecture is described in bottom-up order combined with one application example. The procedure to change the three-tier structure in the program is shown when there are changes in database design. The method to solve the compile error that is most common and difficult to modify is given. The result can provide a reference for Web database application developers.

Key words: three-tier architecture; database application system; C#. NET

0 引 言

ASP. NET是一种用于创建基于Web的应用程序编程模型,它在Web服务器的环境中运行,并且根据服务浏览器请求指示在服务器上执行程序。C#语言是ASP. NET平台的第一语言,也是目前程序开发人员使用最广泛的开发工具,开发数据库应用系统也非常普遍[1]。

随着计算机技术和网络技术的发展,数据库应用系统的使用越来越频繁。企业中的ERP(Enterprise Resource Planning)系统、MES(Manufacture Executing System)系统、APS(Advanced Planning System)系统以及服务大众的气象系统、公交系统、铁路售票系统等的广泛应用,使数据库应用系统的规模越来越大,结构越来越复杂,代码出错性越来越高,维护性越来越困难。如何解决数据库应用系统的可拓展性、易维护性及健壮性在系统开发的过程中非常关键[2]。

软件发展的实践证明,模块化的分层设计模型是提高系统可用性和可维护性的主要途径[3-5]。通过分层模型设计,将整个软件系统划分为若干个相互独立的层次,层与层之间通过事先约定的接口相互通讯[6,7]。这样可以把复杂的问题分解,分而治之,降低复杂性,功能清晰,易于实现、修改和维护。

1 三层体系结构

所谓三层体系结构,是在客户端与数据库之间加入了一个"中间层",也叫组件层。这里所说的三层体系,不是指物理上的三层,不是简单地放置三台机器就是三层体系结构,也不仅仅有B/S应用才是三层体系结构,三层是指逻辑上的三层,即使这三个层放置到一台机器上[8,9]。

三层体系的应用程序将业务规则、数据访问、合法性校验等工作放到了中间层进行处理。通常情况下,

收稿日期:2011-11-24;**修回日期**:2012-03-01
基金项目:北京市中青年骨干人才项目(71A1111143)
作者简介:王海燕(1979-),女,山东青州人,讲师,硕士,主要研究方向为系统优化及制造业信息化。

客户端不直接与数据库进行交互,而是通过 COM/DCOM 通讯与中间层建立连接,再经由中间层与数据库进行交互。

具体如下:

(1)数据访问层实现与数据库的交互。一方面从数据库获取数据传递到业务逻辑层,满足业务应用的需要;另一方面将业务逻辑层的数据操作指令传递给数据库,实现数据的存储、修改、删除等操作。

(2)业务逻辑层承上启下,用于对上下交互的数据进行逻辑处理,实现业务应用。

(3)表示层实现与用户的交互,接收用户请求和返回用户请求的数据结果的表现,不涉及具体的数据处理[10]。

三层架构下建立在数据库服务器上的连接数量将大大减少。同时,因为业务规则、合法性校验存在于中间层,因此当业务规则发生改变时,只需更改中间层服务器上的某个组件(如某个 DLL 文件),而客户端应用程序不需做任何处理,因此可维护性得以提高,例如当数据库由 Oracle 转为 SQL Server 后,只需要在数据访问层将对应的数据库连接类进行修改即可,对其他层的代码没有影响,维护十分方便。三层架构也具有良好的可重用性。如果需要开发 B/S 应用,不必重新进行数据访问、业务规则等的开发,可以直接在 WEB 服务器端调用现有的中间层。最后,三层架构的事物处理也更加灵活,可以在数据库端、组件层、MTS(或 COM+)管理器中进行事务处理。

2　C#下三层架构数据库应用系统的建立

ADO. NET 提供了两种访问数据库的方法,一种是利用 Connection、Command 和 DataReader 对象访问数据库,只能从数据库读取数据,不能添加、修改和删除记录。如果只想进行查询,这种方式效率更高一些。还有一种方法,可以利用 ADO. NET 对象模型中的 Connection, Command, DataAdapter 和 DataSet 对象,方式比较灵活,不仅可以对数据库进行查询操作,还可进行增加、删除和修改等操作[11]。后种应用范围更广。

按照自下而上的三层结构建立数据库应用系统,步骤如下。

2.1　设置数据访问层

第一步,添加新项,选择数据集。

第二步:打开数据集,添加与数据库的链接。

第三步:将数据表拖至数据集中,右键点击 DataTable AdapterData,可以看到自动产生的 InsertCommand, UpdateCommand 和 DeleteCommand。可以先把三者的 commandText 内容拷贝出来,留待后用。因为 VS2008 系统本身存在的 bug,修改 GetData 后这些内容会消失,所以可以提前拷贝出来,在后面修改 sql 语句时,以此为基础。

第四步:修改 GetData( ) 的原有配置。右键点击 GetData( ),选择配置,将原先的 sql 语句进行修改,要显示的字段添加完备。以本人参与开发的项目为例,GetData( ) 中默认的 sql 语句为 select * from bisdevice,可以更改如下:

SELECT a. 设备 ID, a. 设备名称, b. 部门名称, c. 设备类别, d. 设备状态 FROM bisshebei a left join bisbumen b on b. 部门 ID = a. 所在部门 ID left join bisshebeileibie c on c. 设备类别 ID = a. 设备类别 ID left join bisshebeizhuangtai d on d. 设备状态 ID = a. 设备状态 ID。

这样做的目的是使 DATATABLE 可以显示所有的外键关联字段,因为有的时候客户不关心设备类别 ID 具体是什么,而关心具体的设备类别是什么,所以需要通过表连接把相关字段都显示出来。通过字段查询记录、删除记录、插入记录、修改记录等的方法类似。

2.2　设置业务逻辑层

数据访问层部分是数据处理的核心部分,一般变动比较小。在业务逻辑层部分,可以根据业务需求灵活进行修改。

首先,从网站里“添加新项”,然后“添加类”,最后编写类代码,如下所示:

```
public dsNainai. deviceDataTable
GetDevicesByLineIDDeviceID( int 投入生产线 ID, int 设备 ID) { if (投入生产线 ID < 0 && 设备 ID < 0) return Adapter. GetDevices( );
else if (投入生产线 ID > 0 && 设备 ID < 0) return Adapter. GetDevicesByLineID (投入生产线 ID);
else return Adapter. GetDeviceByDeviceID ( 设备 ID); }
```

如果没有中间业务层,而采用两层结构的方式,由表现层直接访问数据层,则显示模式比较固定,不够灵活。

2.3　设置表现层

在表现层,可以利用 GridView 显示整体概况,利用 DetailsView 显示每条记录的详细情况,设置两者的数据源并进行配置。右击 DetailsView,选择“显示智能标志”,然后选择“编辑字段”,然后在“字段”对话框中设置“选定的字段”,使选定的字段跟数据库表里的字段一致,不然会跟数据方法里的参数产生矛盾。

配置数据源选择业务对象时,将“只显示数据组件”前的“√”取消,这样可以选择逻辑层定义的类中的任何一个方法。可以看出,数据集里所有的方法都可用。所以没有逻辑层也是完全可行的,只不过逻辑

层处理更灵活,有时候一些业务逻辑仅依靠数据层实现不了,就必须由逻辑层进行细化。

2.4 数据添加、修改和删除

经过上述步骤后,只能在 DetailsView 中显示数据查询结果,还不能进行数据编辑,如添加、修改和删除。进一步设置如下:右击 DetailsView 选择“显示智能标志”,选择“启用插入、编辑、删除”,然后“编辑字段”,将所有需要编辑的字段连同 CommandField 一起转化为模板字段,具体方法就是:选定某一个字段,然后点击右下角的“将此字段转换为 TempleField”。注意:自动生成的关键字不用转换为模板字段,因为在添加记录时该字段是自动生成的,在编辑和删除记录时,该字段是索引字段。

编辑好字段后,右击 DetailsView 选择“显示智能标志”,然后选择“编辑模板”,分别选择“EditItemTemplate”和“InsertItemTemmplate”进行设置,如下所示:

```
<EditItemTemplate>
<asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("设备名称") %>'></asp:TextBox>
</EditItemTemplate>
<InsertItemTemplate>
<asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("设备名称") %>'>
</asp:TextBox> </InsertItemTemplate>
```

小窍门是因为编辑和插入的模式基本相同,所以可以先设置 EditItemTemplate,设置好后将代码拷贝至 InsertItemTemplate 位置即可,只有稍许地方需要修改。

需要注意的是,如果是有选择项,需要设置好“绑定字段”。如在模板中的“设备类别”项的“EditItemTemplate”中进行设置时,去掉 TextBox,添加一个 ListBox,右击显示智能标志,选择“编辑 DataBinding”,在下图中要选择绑定字段为“设备类别 ID”。同时 ListBox 的属性项中“DataValueField”要同样设置为“设备类别 ID”。

2.5 Listbox 的设置

添加、修改记录时,有些字段如“设备类别”这些需要从数据库里读取,不能随便填写,所以需要 ListBox 来进行。右击 ListBox 进行属性设置,注意 DataTextField 为“设备类别”,而 DataValueField 为“设备类别 ID”,因为在数据库中记录的是设备类别 ID。因为设备类别 ID 不是必填字段,所以有可能为空,设置方法如下:

将 AppendDataBoundItems 设置为“True”,在“Items”里添加新列,在“ListItem 集合编辑器”里,Value 项的值设置为空,如果不填,只是 Null,会报错,所以在. aspx 里修改如下:

```
<asp:DropDownList ID="DropDownList1" runat="server"
AppendDataBoundItems="True"
DataSourceID="ods_getleibies" DataTextField="设备类别"
DataValueField="设备类别 ID"
SelectedValue='<%# Bind("设备类别 ID") %>' Width="100px">
<asp:ListItem value="">(none)</asp:ListItem>
</asp:DropDownList>
```

这样设置的原因是在逻辑层的 csDevice 类里,有如下的代码:

```
public bool Adddevice(string 设备名称, string 设备型号, int? 所在部门 ID, string 操作人员, int? 设备类别 ID,int? 设备状态 ID, DateTime? 购置日期, string 投入工序, int? 投入生产线 ID, string 资产负责人, string 维修保养负责人, int? 制造厂商 ID, int? 检修状况 ID, DateTime? 生产日期,int? 经销商 ID, string 安装地点, string 图片, DateTime? 设备状态开始时间, string 机器编号,string 设备编号)
{ int rowsAffected = Adapter.InsertQuery(设备名称, 设备型号, 所在部门 ID, 操作人员, 设备类别 ID, 设备状态 ID, 购置日期, 投入工序, 投入生产线 ID, 资产负责人,维修保养负责人, 制造厂商 ID, 检修状况 ID, 生产日期, 经销商 ID, 安装地点, 图片, 设备状态开始时间, 机器编号,设备编号);
//如果刚好新增了一条记录,则返回 true,否则返回 false
return rowsAffected == 1;}
```

注意,int ? 中的“?”说明该项可以为空。

3 应用注意事项

3.1 数据库设计发生变化时三层结构的修改

以数据库表新增字段为例,当某个数据表的结构需要增加新字段,首先需要在数据层相应语句里进行修改。如在设备表里新增“机器编号”列,则在数据集的相关配置中进行修改。首先在“表中应装入的数据”配置页将“机器编号”添加上。同理,也要修改插入记录、更新记录以及查询记录的相应位置。有些配置不需要修改,如“DeleteQuery(@设备 ID)”配置不需要修改,因为实现的功能是根据“设备 ID”删除整条记录,跟新增字段“机器编号”无关,所以不需要改动。接下来是逻辑层的修改,同样的道理,逻辑层里与新增字段“机器编号”相关的处理需要修改,如“AddDevice(新增设备)”和“UpdateDevice(更新设备)”进行处理,而与“机器编号”无关的处理可以不用修改。最后是表现层。首先在 GridView 中右击“编辑字段”,然后通过“刷新架构”,可以将“机器编号”增加到可用字段里,然后添加到选定的字段,接下来的处理跟上文设置

表现层的步骤一样。

3.2 常见错误的处理

在开发中经常出现的一个问题是编译时出现“未能找到带参数的非泛型方法”,包括笔者在内的很多用户在基于 C#. NET 的数据库应用系统开发中都碰到了这个问题,微软的官方教程没有解决这个问题。究其原因,是在 ASP. NET 中 ObjectDataSource 自动配置数据源进行操作的时候,会生成两个字段,一个是[12]:

OldValuesParameterFormatString = " original_{0}"

另外一个是:(以删除操作为例)

<DeleteParameters>

<asp:Parameter Name = " original_XML_ID" Type = " Int32" />

</DeleteParameters>

OldValuesParameterFormatString 是根据设定的 SQL 语句中的参数确定的,而 Parameter Name 却是根据业务逻辑层中删除函数的第一个参数确定的。比如,业务逻辑层中删除组件是这样定义的,函数中的第一个参数就是 ObjectDataSource 生成的 Parameter Name,如下所示:

```
[ System. ComponentModel. DataObjectMethodAttribute ( System. ComponentModel. DataObjectMethodType. Delete, true) ]
public bool DelXML(int original_XML_ID)
{
Int rowsAffected =
Adapter. Delete( original_XML_ID) ;
return rowsAffected = = 1;
}
```

结合 Scott Mitchell 的 ASP. NET 2.0 教程及本人的项目开发研究,解决方法如下:

(1)数据集的设置中将所有 Original_A(A 代表字段)的地方去掉 Original_,只保留 A。

(2)所有的函数的参数都同数据集里一致,包括参数名称及参数个数。

(3)程序中删除所有的 OldValuesParameterFormatString = " original_{0}"。

4 结束语

随着信息技术的发展,WEB 数据库系统应用越来越广泛。基于 C#. NET 的三层架构数据库系统由于其快速方便的开发模式、较高的可维护性和可重复性以及灵活的事务处理机制,在大型数据库应用系统开发中占有非常重要的地位。目前微软的. net 2.0 提供了强大的数据访问组件,非常方便进行大型数据库应用系统的开发。但是由于系统本身存在的 BUG 及各种教程的滞后性,至今规范化、标准化的使用指导及常见编译错误的处理建议还比较少。

文中的工作可以帮助程序员快速开发 WEB 数据库应用系统,并提高应用程序的性能,减少开发工作量,方便进行软件维护,并为其他相关工作者提供可借鉴的经验。

参考文献:

[1] Troelsen A. C#与. NET 4 高级程序设计[M]. 北京:人民邮电出版社,2011.

[2] 高丕莲,侯德文,蔡小芳. 基于 Web 服务的电子商务中数据集成的研究[J]. 信息技术与信息化,2006(6):66-69.

[3] 叶兴茂,王静波. 软件系统开发中的组件框架技术研究、设计和应用[J]. 国土资源信息化,2005(3):15-19.

[4] Grundy J, Mugridge W. Constructing Component-based Software Engineering Environments[J]. Information and Software Technology,2000(4):13-15.

[5] 王卫军,付晓江. 基于三层体系结构电子政务系统的 JSP 技术[J]. 吉林大学学报,2003,21(1):87-91.

[6] 盛翊智,谢自美,曾喻江. 电子商务网络中的三层体系结构[J]. 信息技术,2001(11):32-34.

[7] Voruganti K. Adaptive Hybrid Server Architecture for Client-server Object Database Management Systems[D]. Canada: University of Alberta,2001.

[8] 郭　靖. ASP. NET 开发技术大全[M]. 北京:清华大学出版社,2009.

[9] Stallings W. Network Security Essentials: Applications and Standards[M]. [s. l.]:[s. n.],2003:233-271.

[10] 黄忠成. NET Framework3.5 数据库开发圣典-ASP. NET 篇[M]. 北京:电子工业出版社,2008.

[11] 密君英. 基于三层架构的 ASP. NET 项目实战教程[M]. 北京:中国电力出版社,2011.

[12] Mitchell S. ASP. Net 2.0 数据库指南中文版翻译[EB/OL]. 2006-12-04. http://www. cnblogs. com/lovecherry/archive/2006/07/02/440840. html.

(上接第 77 页)

[10] 谢　红,赵　光. 管道泄漏信号检测与定位技术研究[D]. 哈尔滨:哈尔滨工程大学,2003.

[11] Chapelle O, Vapnik V N, Bousquet O, et al. Multiple parameters forsupport vector Machines[J]. Machine Learning,2002,46(1):131-159.

[12] 廉小亲,苏维钧,田黎明. 基于负压波法的输油管道泄漏检测定位系统[J]. 计算机工程与设计,2007,28(9):2199-2200.

[13] Twining C, Taylor C. The use of Kernel principal component analysis to model data distributions[J]. Pattern Recognition,2003,36(1):217-227.