摘 要

典型的楼宇集成管理系统需要将智能建筑内实现各种功能的子系统和现场设备互连，通过资源的共享和信息的综合实现整个楼宇的统一协调管理运营。而在实际的集成工作中，各子系统往往由不同生产厂家提供，在应用程序接口、通信协议、数据库结构等方面存在的异构情况，因此计算机监控系统软件需要开发的设备通信驱动程序就越来越多，这样严重影响了各子系统的开放性和互操作性，为整体楼宇系统集成方案的制定和实施带来了很大的障碍。

而基于 COM/DCOM 技术的 OPC 提供了一个统一的标准，不同厂商只要遵循 OPC 标准就可以实现软硬件的互操作。OPC 采用了 CLIENT/SERVER 模式，针对硬件设备的驱动程序由硬件厂商完成，提供具有同意 OPC 接口的 SERVER 程序；软件厂商按照 OPC 标准访问 SERVER 程序，即可实现与硬件设备的通信。

本文首先研究了 OPC 技术、ATL、及 IDL 语言；然后深入研究了 OPC DA 2.05a 规范；随后研究了基于 VISUAL C++6.0 动态模板库（ATL）的楼宇子系统 OPC 服务器以及基于 VISUAL BASIC 客户应用程序的详细开发流程，并撰写了相应的 DEMO 程序；最后研究了 OPC 在智能楼宇中的应用实例。

研究表明，基于 CLIENT/SERVER 模式的 OPC 技术将软硬件厂商区分开来：硬件厂商熟悉自己的硬件设备，因而设备驱动程序性能更可靠；软件厂商可以减少复杂的设备驱动程序的开发周期，只需遵循 OPC 标准就可以实现与硬件设备的通信，因此只用专注各子系统功能的完善。这样进一步提高了各子系统的互操作性，进而提高了各子系统的开放性并为系统集成提供了方便。

关键词：OPC，COM，智能楼宇，系统集成，开放性

Abstract

A typical building integration management system needs the interlinkage between the subsystem and the field device. It implements the consolidate management of the whole building through the share of the resource and the integration of the information. But each subsystem may be provided by different manufacturers in the practical integration, and then the different structures may exist in the application procedure, communication protocol and the database. So the drivers of the equipments needed to be designed become more and more in the computer monitor system software. It badly reduces the opening and the interoperability of each subsystem, and then blocks the establishment and the implement of the whole building system integration precept.

At the same time, the OPC provides a uniform interface standard based on COM/DCOM technique. And the interoperability between the software and the hardware can easily be realized among different vendors while just needs to follow the OPC standard. The OPC adopts the modes of CLIENT/SERVER: the manufacturer provides the driver of the equipment and the standard OPC server code; the software vendor accesses the server by the OPC standard and then the communication with the equipment can be easily realized.

This paper first introduces the OPC technique, ATL and IDL language, and then lucubrates the OPC DA 2.05a specification. At last the detailed development flow of OPC server based on Visual C++6.0 and the client application based on Visual Basic is studied, at the same time the demo code has been written.

The study indicates that the OPC based on CLIENT/SERVER mode separates the responsibility between the software and hardware manufacturers. The hardware manufacturer is more familiar with their equipments, so the driver written by them is more dependable. While the software manufacturer can take no care of the device driver, so they can just pay attention to the perfection of the subsystem function. As a result, the interoperability between each subsystem and device is further improved and then opening of each subsystem is also improved. So the system integration becomes very convenient.

Key words: OPC, COM, Intelligent Building, System Integration, Opening

独 创 性 声 明

本人声明，所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得武汉理工大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：王小辉 日 期：2007.5.15

关于论文使用授权的说明

本人完全了解武汉理工大学有关保留、使用学位论文的规定，即学校有权保留、送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

(保密的论文在解密后应遵守此规定)

签 名：王小辉 导师签名：[illegible] 日 期：2007.5.15

第1章 绪论

1.1 课题研究的背景与意义

智能楼宇融计算机(Computer)、通讯自动化(Communication Automation)、自动控制(Control Automation)和IC卡(Intelligent Card)技术为一体，是数字化、网络化和信息化结合的产物。智能楼宇由自动控制系统、通信自动化系统和办公自动化系统通过综合布线和计算机网络有机集成[1]。

在传统的控制系统中，智能设备与控制软件之间的信息传递是通过驱动程序来实现的，即任何上位监控软件在使用某种硬件设备时都需要开发专用的驱动程序，如图 1-1 所示。因此在系统集成过程中存在着严重障碍：各子系统之间难以实现开放的、无缝隙的连接[2]。

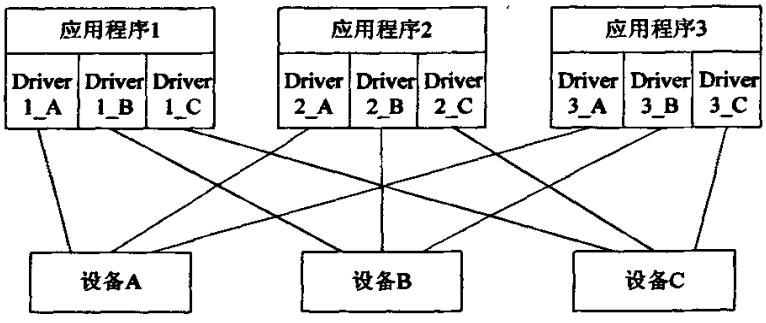

图 1-1 基于驱动程序的数据访问方式

为了解决上述问题，硬件制造商们一直试图开发出一种可以被任何客户使用的"万能 I/O 驱动"程序。但是由于客户端协议不一致，这项工作至今没有取得成功。与此同时，OPC（OLE For Progress Control）和现场总线标准的制定正好为上述问题的解决开辟了新的道路。采用 OPC 标准后，针对硬件的驱动程序不再由软件开发商开发，而是由硬件开发商根据硬件的特征提供统一的 OPC 接口程序。由于硬件开发商更熟悉自己的硬件特性，从而能够最大限度地挖掘硬件的潜力，提高驱动程序的性能和可靠性[3]。

基于 OPC 标准的数据访问方式如图 1-2 所示。采用 OPC 标准后，由硬件开发商提供统一的 OPC 接口程序，从而避免了重复开发驱动程序，因此大大降低了开发经费和开发周期。OPC 规范采用标准的 CLIENT/SERVER 模型，其实质是在硬件供应商和软件供应商之间建立统一的规范，只要遵循这套规范，数据交互对两者来说都是透明的。硬件供应商无需考虑应用程序的多种需求和传输协议，软件开发商也无需了解硬件的实质和操作过程。这样客户端应用程序可以灵活而有效地与设备之间读写数据。

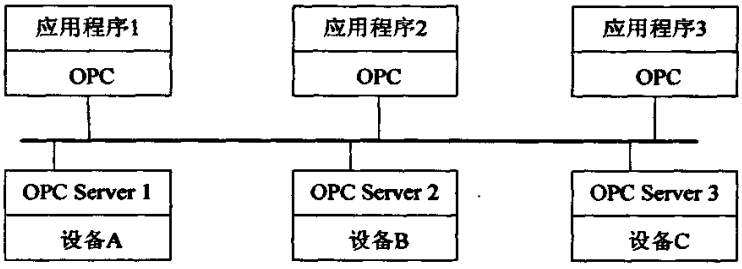

图 1-2　基于 OPC 规范的数据访问方式

1.2 课题的国内外动态

目前，在国外，特别是欧美以及亚洲的日本，OPC 技术已成为智能楼宇领域非常重要的一部分，这为不同厂家提供的设备和系统的数据访问提供了统一的平台，这样不仅减少了系统开发和升级的成本，而且更加提供了通信的安全系数。

OPC 技术作为一项工业标准在国内仍处于推广和初步应用阶段，近年来引起了广泛的关注。不少高等院校、研究机构和制造厂商都展开了对 OPC 技术的研究和应用。一些公司如北京华控公司也加入了 OPC 基金会，成为其成员单位。在应用方面，沈阳自动化研究所在开发新一代分布式控制系统时就采用了 OPC 技术，实现了上层应用软件通过 OPC 服务器访问现场设备信息的功能，同时还开发了 OPC 软件包和相应的控件。现在不少自动化仪表制造厂商在提供硬件的同时也提供相应 OPC 服务器。一些国内工控软件公司也充分利用 OPC 技术增强和扩展其软件功能，例如北京亚控公司从组态王 5.1 版本开始支持 OPC 技术。

遗憾的是 OPC 在智能楼宇中的应用还尚未得到重视，而且 OPC 服务器

的开发在国内也处于起步阶段，所以本文对 OPC 在智能楼宇中应用的研究及 OPC 服务器开发的介绍很有必要。

1.3 课题研究的主要内容和拟解决的关键技术

（1）本课题研究的主要内容包括以下几个部分：

① 智能楼宇系统的集成，重点研究通过现场总线进行智能楼宇系统集成的方法；

② OPC 协议，重点研究数据访问规范(OPC Data Access)；

③ OPC 服务器开发；

④ OPC 客户端应用程序开发。

（2）本课题拟解决的关键技术：

① 智能楼宇信息管理系统体系结构的建立；

② OPC DA 服务器的开发；

③ OPC 客户端应用程序的开发。

第 2 章　智能楼宇及相关技术背景

智能楼宇(Intelligent Building)是信息科学和计算机应用科学的必然产物。智能楼宇随着科技的发展不断完善，一般认为是利用系统集成的方法，将计算机技术、通信技术、信息技术和建筑艺术有机地结合起来，来完成对设备的自动监控、信息资源的有效管理和使用者的信息服务及建筑的优化组合。智能楼宇采用电子信息技术对建筑大楼的设备进行自动监控，对信息资源进行管理和对用户提供信息服务，可以大大提高大厦的管理效益和最大限度地降低大厦的各种运行费用。

2.1 智能楼宇概况

一个典型的楼宇自动化系统包括智能化的空调系统、热力系统、变配电系统、给排水系统、通风系统、环境检测系统、消防报警系统、安全防范系统、电梯系统、停车场管理系统等。不同类型的智能楼宇中智能化系统的总体结构不尽相同，但其核心功能、系统是相同的。智能系统所用的主要设备通常放置在智能化建筑环境内的系统集成中心，它通过综合布线或现场总线与各种终端设备相联接，感知建筑内各个空间中的信息及变化，并通过计算机处理给出相应的对策，再通过通信终端或控制终端(门禁、开关、阀门等)给出相应的反应，使整幢大楼系统处于动态实时监控状态，实现高度智能化。

智能楼宇从 1984 年出现至今经过短短 20 年的发展，从最初的小规模简单对象控制、分散管理控制、人工信息传递信息到现在的一体化集成时代和网络时代，主要经历了以下几个阶段[4]：

（1）1980-1985 年：单一功能专用系统时代，各子系统独立运转；

（2）1985-1990 年：多功能系统时代，子系统数减少，部分系统集成；

（3）1990-1995 年：系统集成时代，将集中控制变为分散控制；

（4）1995 年至今：一体化集成时代，利用现场总线技术将所有的子系统集成为一个网络系统。

目前国内楼宇监控系统大都采用集散式控制与通讯技术，即使用系统机为

上位机、单片机为核心的仪器作为下位机以及以 RS-232 或 RS-485 串行通讯总线作为系统链接。监控系统通过现场安装的传感器、控制器或相应的变送器和执行机构对各种被监控对象进行自动检测和控制。这种系统虽然可以满足大多数用户的要求，但由于下位机的工作完全由上位机控制，上位机一旦出现故障，整个系统将失控瘫痪，因此可靠性较差，而且又无统一标准，所以系统的可靠性、维修性、互换性以及可扩充性均难以达到理想效果和规范要求。

随着我国经济的发展，对大厦智能化的要求越来越高，需要监控的对象种类繁多，因此必须使用具有统一规范、组建灵活、高可靠性、良好的扩展性和维护性的方式来组建系统，现场总线和 OPC 技术正好满足了这个要求[5]。

2.2 现场总线技术

（1）现场总线概念

现场总线基金会(FF)定义：现场总线是一种用于智能化现场仪表和自动化系统的开放式、数字化、双向传输、多分支结构的通信网络。它是用于过程自动化和制造自动化最底层的现场设备或现场仪表互连的通信网络，是现场通信网络与控制系统的集成。现场总线技术的关键标志是能支持双向、多变量和总线式的全数字通信。

现场总线系统是计算机控制系统与通讯技术结合的产物，是新一代全数字、全分散和全开放的现场控制系统。其中，现场是指工作环境处于生产设备的一侧；现场设备、仪表是指位于生产现场的各种传感器、驱动器和执行器等设备；总线是指传送信息的公共路径，这些遵守相同联接规范的设备通过“公共路径”联接为系统，并实现相互操作。因此，现场总线是面向工厂底层自动化及信息集成的数字化网络技术，人们把基于这项技术的自动化系统称为基于现场总线的控制系统。现场总线技术的核心是它的通信协议，它必须依据国际标准化组织的计算机网络开放系统互连基本参考模型 OSI(Open System Interconnection)来制定，它是一种开放的七层网络协议标准，但多数现场总线技术只使用其中的一、二和七层协议[6]。

（2）现场总线发展现状

目前主要有四种现场总线在我国得到较为广泛的推广和应用，它们是 PROFIBUS、FF、CAN 和 LonWorks。机械部正在大力推广 PROFIBUS，成立

了 PROFIBUS 用户协会[7]。以各地仪表集团为基础，成立了中国仪器仪表行业协会现场总线专业委员会(CFFC)，已经成为 FF 的成员之一，冶金部自动化院于 1995 年、北京华控公司于 1996 年也加入了 FF。中国自动化协会正在力推 CAN 总线。而 LonWorks 技术目前已成功地应用在我国的电力、楼宇自动化等许多行业，其中航天部五零二所北京康拓公司、电力部南京自动化院等单位都在积极地进行研究开发[8]。

在众多的总线中较受欢迎的应属 LonWorks。由于 LonWorks 技术完善、使用方便灵活、开发周期短、实用性强，从它问世起就受到了众多计算机制造商、仪表公司等的认可和支持。迄今已有 4000 多家生产商使用了 LonWorks 技术，并已安装了 500 多万个节点[9]。并且已有很多家公司正在生产 LonWorks 产品或将其产品纳入 LonWorks 网络，如 Honeywell 将 LonWorks 技术用于楼宇控制系统，Cisco 公司已有 LonWorks 与 TCP/IP 协议的 IP 产品。Echelon 公司于 1995 年底在中国成立了代表处，旨在推广 LON 的应用。短短几年，已广泛地应用于电力系统综合自动化、船舶、楼宇自动化等诸多领域，足以证明 LON 的强大实力和受欢迎程度。1998 年底中国计算机学会工控专业委员会为推广 LON 而成立了分委员会，建设部也建立了 Lon Works 的用户协作网。这些现象代表了国内工控界对 LonWorks 技术发展潜力和前景的群体认可，证明 LON 是一种最有希望的现场总线[10]。

2.3 OPC 技术分析

2.3.1 OPC 技术背景：COM 技术

（1）OPC 技术产生背景

OPC 规范以组件对象模型和分布式组件对象模型（COM/DCOM）技术为基础，采用客户、服务器模式，定义了一组 COM 对象及其接口规范[11]。OPC 规范定义了客户程序与服务器程序进行交互的方法，但并没有规定具体的实现，OPC 服务器可由不同供应商提供，其代码决定了服务器访问物理设备的方式、数据处理等细节。但这些对 OPC 客户程序来说都是透明的，只需要遵循相同的规范或方法就能读取服务器中的数据[12]。

通过 COM 接口，OPC 客户程序可以和一个或多个提供商的 OPC 服务器连

接。同时一个 OPC 服务器也可以同多个客户程序相连，形成多对多的关系。任何支持 OPC 的产品都可以无缝地实现系统集成。由于 OPC 技术基于 DCOM，所以客户程序和服务器可以分布在不同的主机上，形成网络化的监控系统[13]。

（2）COM 技术简介及特点

COM（Component Object Model）即组件对象模型，是微软公司为了计算机工业的软件生产更符合人类的行为方式而开发的一种新的软件开发技术[14]。在 COM 构架下，可以开发出各种各样的功能专一的组件，然后将它们按照需要组合起来，构成复杂的应用系统。由此带来的好处是多方面的：可以将系统中的组件用新的替换掉，以便随时进行系统的升级和定制；可以在多个应用系统中重复利用同一个组件；可以方便地将应用系统扩展到网络环境下；COM 与语言，平台无关的特性使所有的程序员均可充分发挥自己的才智与专长编写组件模块等[15]。COM 是开发软件组件的一种方法。组件实际上是一些小的二进制可执行程序，它们可以给应用程序，操作系统以及其他组件提供服务。开发自定义的 COM 组件就如同开发动态的，面向对象的 API（Application Programming Interface）。多个 COM 对象可以连接起来形成应用程序或组件系统。组件可以在运行时刻，在不被重新链接或编译应用程序的情况下被卸下或替换掉[16]。

COM 所含的概念并不止是在 Microsoft Windows 操作系统下才有效。COM 并不是一个大的 API，它实际上像结构化编程及面向对象编程方法那样，也是一种编程方法[17]。主要技术特点如下：

① 二进制特性

接口规范并不建立在任何编程语言的基础上，而是规定了二进制一级的标准。任何语言只要有足够的数据表达能力，它就可以对接口进行描述，从而可以用于与组件程序有关的应用开发。

② 接口不变性

接口是客户程序和组件对象之间的桥梁，接口如果经常发生变化，客户程序和组件程序也要跟着变化，这对于应用系统的开发非常不利，也不符合组件化程序设计的思想。因此，接口应该保持不变，只要客户程序和组件程序都按照既定的接口设计进行开发，则可以保证两者独立开发结束后，它们之间的协作运行能力能达到预期的效果[18]。

③ 继承性（扩展性）

COM 接口具有不变性，但不变性并不意味着接口不再发展，随着应用系统和组件程序的发展，接口也需要发展。COM 的接口继承不同于类继承，类继承不仅是说明继承，也是实现继承。而接口继承只是说明继承，即派生的接口只继承了基接口的成员函数说明，没有继承基接口成员函数的实现，因为接口定义不包括函数实现部分[19]。

④ 多态性

COM 对象具有多态性，其多态性通过 COM 接口体现。多态性使得客户程序可以用统一的方法处理不同的对象，甚至是不同类型的对象，只要它们实现了同样的接口。如果几个不同的 COM 对象实现了同一个接口，则客户程序可以用同样的代码调用这些 COM 对象。

（3）COM 组件的分类

COM 组件按照代码模块的结构和代码模块与客户进程间的关系可以分为：进程内组件、进程外组件、远程组件三种。

① 进程内组件

进程内组件使用 COM 创建并且以动态链接库（DLL）的方式执行，运行时动态地装入到客户的进程空间中，和客户应用程序运行在同一进程空间中，所以进程内组件程序运行速度快、效率高。进程内组件不是一个完全可以执行的应用程序（EXE），所以进程内组件只能用在一个调用环境中，不能作为一个独立的应用程序执行。图 2-1 给出了进程内组件的逻辑结构[20]。

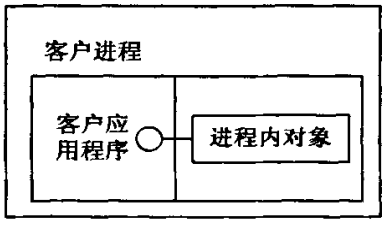

图 2-1　进程内组件的逻辑结构

② 进程外组件

进程外组件是基于 COM 创建，但是以可执行程序的方式来执行。这种组件之所以称为进程外组件是因为：它们每次运行时都是在自己的地址空间中，并通过一种特定的远程过程调用（LRPC）和调用程序进行通信[21]。图 3-2 给出

了进程外组件的逻辑关系结构。COM 通过本地过程调用 LPC (Local Process Call)实现了不同进程间的通信。RPC 标准是在开放软件基金会 OSF (Open Software Foundation)分布式计算环境 DCE (Distribute Compute Environment) RPC 规范中定义的，它使得不同机器上的进程可以使用各种网络传输技术进行通信[22]。

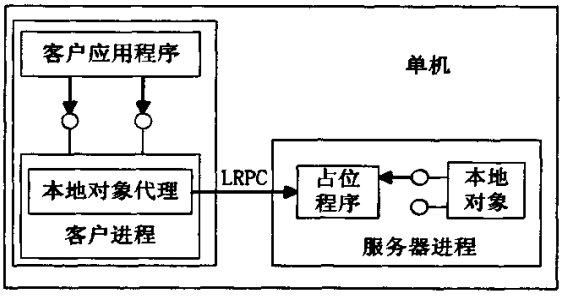

图 2-2 进程外组件的逻辑关系图

③ 远程组件

客户调用远程组件是通过网络来实现的，远程组件总是运行在另外一个进程中，远程组件可以以 EXE 或 DLL 的形式封装。当以 DLL 形式封装时，在远程组件的计算机上需要一个代理进程。由这种组件组成的系统可以被分为几个部分，其中每个组件都可运行在不同的计算机上。图 3-3 给出了远程组件的逻辑关系结构[23]。

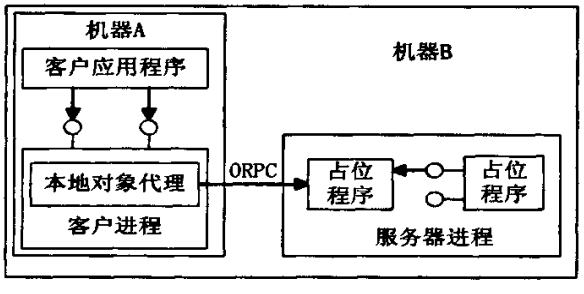

图 2-3 远程组件的逻辑关系图

（4）接口描述语言（IDL）及 COM 组件的接口

COM 规范在开放软件基金会分布式计算环境远程程序调用接口描述语言

（Open Software Foundation Distributed Computing Environment Remote Procedure Call Interface Description Language）的基础上，进一步扩展形成了COM IDL。接口描述语言提供了一种不依赖于任何语言的接口描述方法。因此，它可以成为组件程序和客户端程序之间的共同语言[24]。

由于IDL是一门专门的语言，限于篇幅，本文未对其进行详细的阐述，仅以一个利用IDL语言定义的接口实例来作简要说明。

```
[
  UUID(318B4AD0-06A7-11D3-9B58-0080C8E11F14),
  Helpstring("This is a video interface"),
]                    //注释
Interface IVideo: IUnknown
{
  HRESULT GetSingnalValue( [out, retval]   long* plRetval);
}
```

上面代码中，前四行用来注释，分别采用了UUID和Helpstring属性来注释；interface是一个关键词，用来定义接口，该接口由IUnknown派生而得到；HRESULT是函数的返回类型，它由32个字节组成，用来表示函数操作的成功或失败以及具体的信息代码；[out, retval]表示输出的参数是retval类型。

按照COM规范，COM对象和接口必须被唯一地表示。两者都由一个128位的全局唯一标识符GUID来标识，GUID用随机方法产生，可以保证全球范围内的唯一性。对象标识符称为CLSID，接口标识符称为IID。当一个客户要使用一个COM对象时，它首先通过CLSID来创建COM对象，再由IID获得COM对象的一个接口指针，该接口指针指向接口的实现代码（接口的方法和属性），通过接口指针，客户调用COM对象所提供的服务。从这个过程中可以看出，客户与COM对象只通过接口打交道，对象对于客户来说只是一组接口。COM对象的相关接口如图2-4所示[25]。

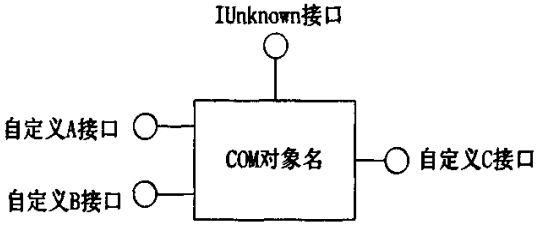

图 2-4 COM 基本对象图

图 2-4 中描述的 COM 对象的接口有两种：一种是 COM 对象自有的接口 IUknown，一种是 COM 对象自定义的接口（如 A、B、C 接口）。

IUknown 接口是 COM 对象必须支持的一个接口，是 COM 对象最基本的接口。当一个客户引用 COM 对象时，首先获得的就是指向该接口的指针，利用这个指针客户可以调用存在于 IUknown 接口中的 AddRef Release 和 QueryInterface 方法[26]。

COM 接口是包含一个函数指针数组和一个指向这个数组的指针的内存结构。客户程序用一个指向接口数据结构的指针来调用接口成员函数。如图 3-5 所示，接口指针实际上又指向另一个指针，这第二个指针指向一组函数，称为接口函数表，接口函数表中每一项为 4 个字节长的函数指针，每个函数指针与对象的具体实现连接起来。通过这种方式，客户只要获得了接口指针，就可以调用到对象的实际功能。

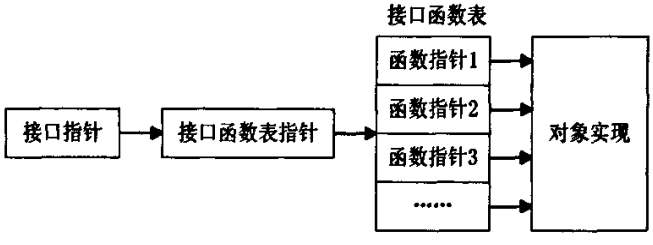

图 2-5 对象的接口指针示意图

因为接口被用于组件程序和客户程序之间的通信桥梁，所以接口应该具有接口不变性。组件对象一经定义，其接口定义就确定下来。具体说，接口的虚函数表 vtable 是确定的，因此接口的成员函数个数是不变的，而且成员函数的先后顺序也是不变的；对于每个成员函数来说，其参数和返回值也是确定的[27]。

当组件被创建以后，客户程序将从组件中得到一个接口指针。以后，组件与客户的一切调用都是通过这个接口指针来实现的。

COM 规范使用 IDL 来定义接口，所有的接口都是直接或间接地从 IUnknown 接口继承而来。其原因在于 IUnknown 接口提供了两个非常重要的特性：生存期控制和接口查询。

（5）COM 组件应用的关键技术：包容与聚合

包容和聚合实际上是一个组件使用另外一个组件的技术，即重用性。对于这两个组件，分别称为外部组件和内部组件。在包容的情况下，外部组件将包含内部组件；而在聚合的情况下，则称外部组件聚合内部组件。

在 COM 中，包容是在接口级完成的，如图 2-6 所示。外部组件包含指向内

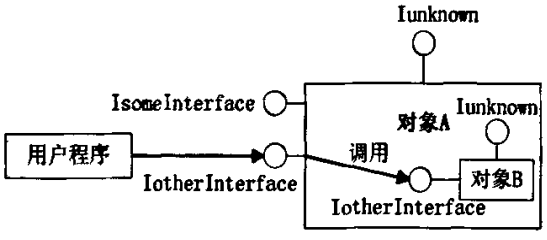

图 2-6 对象包容示意图

部组件接口的指针。此时外部组件只是内部组件的一个客户，它将使用内部组件的接口实现它自己的接口。外部组件也可以通过将调用转发给内部组件的方法重新实现内部组件所支持的某个接口。并且外部组件还可以在内部组件代码的前后加上一些代码以对接口进行改造。

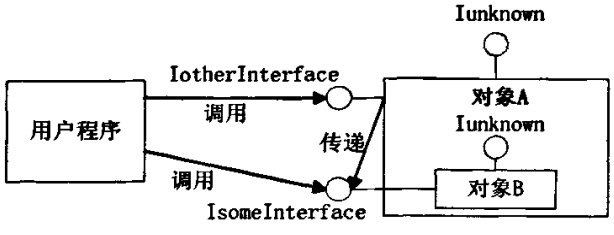

图 2-7 对象聚合示意图

聚合是包容的一个特例，如图 2-7 所示。当一个外部组件聚合了某个内部

组件的一个接口时，它并不像包容那样重新实现此接口并明确的调用请求转发给内部组件。相反，外部组件将直接把内部组件的接口指针返回给客户。使用此种方式，外部组件将无需重新实现并转发接口中的所有函数。当外部组件将内部组件的接口指针返回给客户之后，客户就可以直接同内部组件打交道了。但此时客户不应该知道它是在和两个不同的组件交互，否则无法满足封装的要求。为此内部组件实际上要实现两个 IUnknown 接口，一个称为非代理未知接口（IUnknown），将按通常的方式实现内部组件的 IUnknown 接口。另一个称为代理未知接口，负责把 IUnknown 成员函数的调用转发给内部组件的位置接口或转发给自己的非代理未知接口。

2.3.2 OPC 规范

按照功能的不同，OPC 基金会发布的 OPC 规范分为以下几类：

（1）OPC 数据访问规范(OPC DA, OPC Data Access)：完成对现场设备在线数据的存取。

（2）OPC 报警和事件规范(OPC AE, OPC Alarm&Event Access)：提供了当现场特定的事件和报警条件发生时 OPC 客户程序可从服务器程序得到通报的机制。

（3）OPC 历史数据访问规范(OPC HDA, OPC Historical Data Access)：提供一种通用的历史数据引擎，可以向感兴趣的用户和客户程序提供额外的数据信息。

（4）OPC 批处理规范(OPC Batch)：提供了一种存取实时批量数据和设备信息的方法。

（5）OPC 安全规范(OPC Security)：提供了重要的现场数据，如果这些参数被误修改将会产生无法预料的后果，因此需要防止未授权的操作，OPC 安全性规范就提供了这样一种专门的机制来保护这些敏感数据。

虽然 OPC 各规范都有着自己的特性和不同的要求，但其基础和核心都是 COM 技术。只要按照 COM 的开发流程，并遵循 OPC 对应的规范，都能开发出可靠的 OPC 服务器。本文主要是设计 OPC DA 服务器，所以下面将重点分析 OPC DA2.05a 规范。

OPC DA2.05a 规范是 OPC 基金会最初制定的一个工业标准，其重点是对现场设备的在线数据进行存取。该规范分为定制接口规范和自动化接口规范两部

分，两种接口完成的功能类似，本论文只介绍定制接口规范的基本对象和接口功能。

OPC DA2.05a 规范描述了由 OPC 服务器实现的 OPC COM 对象及相应接口。规范指出一个 OPC 客户程序可以连接到一个或多个由不同厂家提供的 OPC 服务器程序，而多个 OPC 客户程序也可以连接到一个 OPC 服务器程序上，服务器所要访问的设备、数据源、数据名及服务器程序如何进行数据的访问由厂商提供的代码决定。

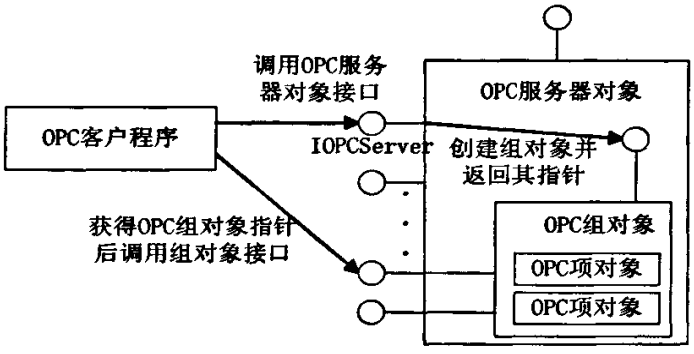

图 2-8 OPC DA 服务器主要对象的接口关系

OPC 数据存取服务器主要由服务器对象、组对象和项对象组成，如图 3-8 所示。OPC 服务器对象维护有关服务器的信息，并作为 OPC 组对象的包容器，可动态地创建或释放组对象；组对象除了维护自身有关的信息外，还提供包容和逻辑组织 OPC 项对象的机制：OPC 项对象则维护 OPC 服务器中与数据有关的信息，但它并不是数据源，仅仅是指向数据源连接。OPC 客户程序不能通过定制接口直接访问 OPC 项对象，因为 OPC 项对象没有引出接口，所有对 OPC 项对象的访问都是通过 OPC 组对象来完成的。

组对象可分为公有组和私有组(或局部组)。公有组用于多个客户程序的共享，私有组只用于一个客户程序。对于一个公用组有一些特定的可选接口完成特定的功能。每一个组对象中，客户可以定义一个或多个项对象。

与 OPC 项相关的信息有值(Value)，品质(Quality)和时间邮戳(Time Stamp)，值是 VARIANT 类型，品质表征了项的内在属性，时间邮戳指明了项值所对应的时间。OPC 规范只规定了 COM 接口的名称和接口向 OPC 客户程序提供的行为，但没有规定如何去实现它，OPC 服务器组件仅提供 OPC 对象接口并管理

这些接口。

OPC 规范中为 OPC 服务器规定了两套接口：定制接口(Custom Interface)和自动化接口(Automation Interface)。其中定制接口是 OPC 服务器必须提供的，而自动化接口是可选的。OPC 客户既可以使用支持 COM 的定制接口，也可以使用自动化接口。定制接口只支持用 C/C++等高级语言编写的客户应用，自动化接口则支持更上层的应用，如 Visual Basic 以及 Delphi 等应用程序，如图 2-9 所示。

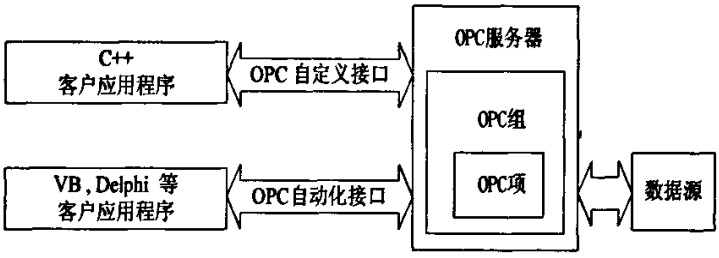

图 2-9 OPC 两种接口示意图

2.3.3 OPC DA 2.05a 规范的对象与各接口定义程序

（1）OPC 服务器对象及其接口定义程序

OPC 服务器对象是 OPC 服务器向外暴露的基本对象，其结构模型如图 2-10 所示。主要接口如下：

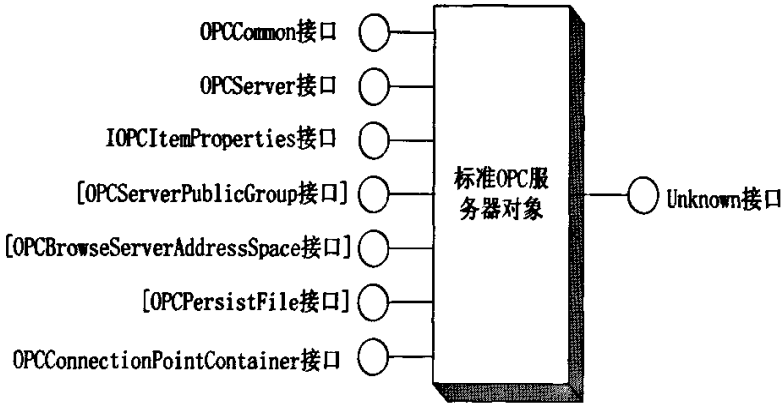

图 2-10 标准 OPC 服务器对象结构模型

·IUnknown

·IOPCCommon

·IOPCServer

·IOPCServerPublicGroups (可选)

·IOPCBrowseServerAddressSpace (可选)

·IPersistFile (可选)

·IOPCItemProperties

·IConnectionPointContainer

本课题从实用化和工程化的角度，仅实现了 OPC 服务器对象的四个必选接口：IConnectionPointContainer、IOPCCommon、IOPCItemProperties、IOPCServer，这四个接口都是由 IUnknown 这个基接口派生出来的。详细的 OPC 服务器对象的接口定义程序见附录 I。

① IUnknown 接口

此接口是所有 COM 对象必须实现的最基本接口，它是其它接口的基接口，主要用于组件对象的生命周期管理。

② IOPCCommon 接口

此接口可以被应用于各种类型的服务器，他们共享这个接口的设计。该接口的功能分别由下面 5 个成员函数提供：

SetLocaleID：设置服务器的区域位置；

GetLocaleID：查询服务器的区域位置；

QueryAvailableLocaleIDs：查询可用的服务器区域位置；

GetErrorString：返回错误信息；

SetClientName：注册客户端名称。

③ IOPCServer 接口

这是 OPC 服务器对象的主接口，它可完成对组对象的动态创建以及对其进行管理。该接口的功能由以下 6 个成员函数提供：

AddGroup：创建一个组对象，并返回客户请求的接口指针，其参数反映了组对象创建时的状态，这是组对象暴露给客户程序的唯一途径；

GetErrorString：获取当前 LocaleID 下的指定错误码的文字描述；

GetGroupByName：通过组名获得一个已有私有组对象的接口指针，从而与之建立连接。它主要用于当相应组对象的接口指针都被释放后重新与之建立连

接(当然如果此组对象已从内存中删除除外)；

GetStatus：返回服务器对象当前状态信息，如服务器运行状态，组对象的数目，版本号，厂商信息，以及当前时间和前次数据刷新时间等等。客户可对GetStatus 方法进行周期性调用来确定服务器是否连接和可用；

RemoveGroup：删除不再使用的组对象。当所有的组对象接口都释放后客户再调用此函数，使相应组对象在内存中彻底删除。但它不能用于公共组对象；

CreateGroupEnumerator：创建一个可列举当前服务器对象内的组对象的枚举器。可以列举组对象的名称，也可以列举指向组对象的 IUnknown 接口的指针。枚举器也是一个 COM 对象，它实现了相应类型枚举接口，如 IEnumString 和 IEnumUnknown 接口，OPC 服务器对它的实现进行了简化，由相应接口函数来创建它的对象，由客户程序来释放它。

④ IOPCItemProperties 接口

服务器对象上的此接口主要提供了相对于 IOPCShutdown 出接口连接点的访问支持。IOPCShutdown 出接口用于当服务器主动与客户程序断开连接时对客户程序进行通知。OPC2. 0 服务器必须支持此接口。它的实现与一般连接点对象的实现相同。其功能由以下两个成员函数提供：

EnumConnectionPoints：建立一个 OPC 服务器对象和客户程序之间所有支持的连接点的枚举器，此时一般只有一个 IOPCShutdown 出接口，当然如果需要，服务器开发者可以定义自己的回调函数；

FindConnectionPoint：查找 OPC 服务器对象和客户程序之间的特定的连接点，一般为对应 IOPCShutdown 出接口的连接点。

⑤ IConnectionPointContainer 接口

此接口用于浏览与 ITEMID(用于标识一个特定的项)相关的属性，也可读取这些属性当前的值。之所以设计本接口是因为许多 ITEMID 与其它像代表工程单位范围或对象描述或报警状态的 ITEMID 相关联。使用此接口可以方便的浏览、定位和读取与特定 ITEMID 相关的信息，可以在不创建 OPC 组对象的情况下读取。该接口的功能由以下 3 个成员函数提供：

QueryAvailableProperties：可返回与特定 ITEMID 相关的属性 ID(用于标识属性)列表及其描述。此列表对于特定的 ItemID 是“相对”稳定的，它会受相应系统配置改变的影响。

GetItemProperties：可返回与特定 ITEMID 相关的属性 ID 的当前值。

LookupItemIDs：可返回与特定 ITEMID 相关的属性 ID 对应的 ITEMIDs 列表，即此方法的目的是看哪些属性 ID 可以成为 OPC 项，可以通过 OPC 组对象添加到 OPC 项列表中。服务器应允许多数或所有项对象的属性被转换成特定 ItemID。

（2）OPC 组对象及其接口定义程序

OPC 组对象结构模型如图 2-11 所示，提供的接口如下。

·IUnknown

·IOPCItemMgt

·IOPCGroupStateMgt

·IOPCPublicGroupStateMgt（可选）

·IOPCSyncIO

·IOPCAsyncIO2

·IConnectionPointContainer

·IEnumOPCItemAttributes

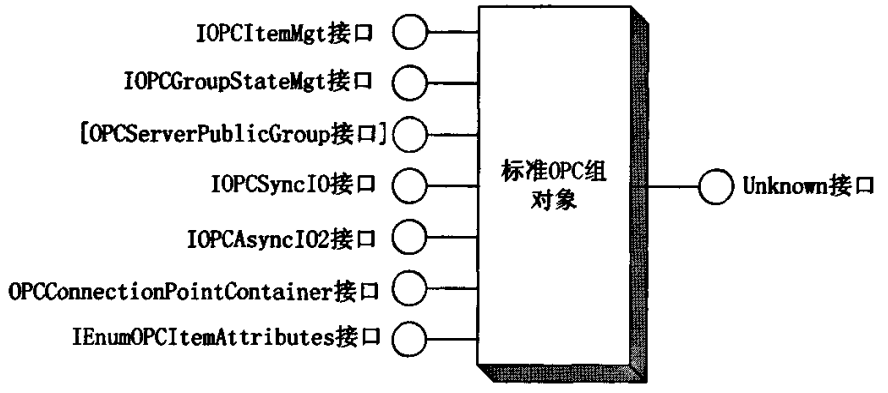

图 2-11 标准 OPC 组对象模型

① IOPCItemMgt 接口

此接口的功能是允许客户添加和删除项对象并可控制项对象的行为。

AddItems：向组对象中添加一个或多个项对象。可添加相同的数据项两次，但每一个项对象的 ServerHandle（项服务器句柄）唯一。

ValidateItems：判断数据项能否被合法的添加，不会对组对象造成任何影响。

RemoveItems：从组对象中删除项对象。从组对象中添加和删除项对象并不影响服务器和物理设备地址空间的数据项，它仅说明了客户是否关心那些数据项。

SetActiveState：设置项对象的激活标志。

SetClientHandles：改变项对象的客户句柄。一般来讲，客户程序在添加项对象时就设置了客户句柄，在以后不会改变它。

SetDatatypes：改变项对象要求的数据类型。

CreateEnumerator：在组对象内创建一个可以列举项对象的枚举器。

② IOPCGroupStateMgt 接口

此接口允许客户程序管理组对象的所有状态。最基本的是改变组对象的更新率和活动状态。

GetState：获得组对象的当前属性状态，如更新率，激活状态，组名，时区，死区，语言标识，客户句柄和服务器句柄等。

SetState：客户可设置组对象不同属性的状态，可改变的属性可从其参数知道。

SetName：设置私有组对象的名称，名字必须唯一。

CloneGroup：使用唯一组名建立某一组对象的另一个副本。新的组对象为私有的。组对象和项对象的几乎所有属性被复制，但新组完全独立于旧的组对象。此方法的主要应用是建立一个可以被客户修改的公共组对象的私有复制。

③ IOPCSyncIO 接口

此接口允许客户对服务器执行同步读写操作。

Read：同步读取组对象内的项的值、品质和时间邮戳等信息。可以从内存(CHCHE)中读取，也可以从设备内直接读取。只有组对象和项对象都处于激活状态时，才可从内存内读取数据。从设备内读取数据不受组对象和项对象的激活状态的影响。 一般情况从缓存中读数据将完成得很快，从设备中读取要花很长时间，从设备读取数据可能会妨碍其它客户操作的执行，所以在大多数情况下客户应从内存中读数据，从设备中读用于特殊的情况如诊断。

Write：对组对象中的项进行同步写操作，值最终被写到设备中，此方法在保证设备真正的接受或拒绝数据后才可返回。写操作不受组对象或项对象的活动状态的影响。设备写操作要花很长时间，建议多数情况下客户程序应进行异步写而不是同步写。

④ IOPCAsyncIO2 接口

此接口允许客户对服务器执行异步读写操作，操作被排队等候，函数立即返回。每项操作被看作一个事务，并被分配一个事务 ID，当操作完成时，客户 IOPCDataCallback 接口的回调将执行。回调中的信息指出了事务 ID 和操作结果。

对于异步 Read，Write 和 Refresh 任意一项操作，相应操作的所有结果都应被 IOPCDataCallback 接口的相应方法的一次回调返回。

一个服务器必须能为每种类型的操作排队等候至少一个事务。所有成功启动的操作都应该被完成，如果超时发生时，服务器应在回调中返回错误。

Read：从组对象内异步读取项的值，IOPCDataCallback::OnReadComplete 结果通过回调方法返回。

Write：对组对象内的项进行写操作。IOPCDataCallback::OnWriteComplete 回调方法返回结果。

Refresh2：强制对组对象内的所有活动项调用 IOnDataChange 回调。从功能上讲，它与对组对象内所有活动项的读操作完成的效果一样。

Cancel2：请求服务器取消一个正在进行的事务处理。一般来讲，如果操作成功，IOPCDataCallback::OnCancelComplete 将发生。

SetEnable：可控制 IOPCDataCallback::OnDataChange 方法的操作，通过输入参数为 FALSE，可禁止事务 ID 为 0 的 OnDataChange 回调(不禁止 Refresh 对 OnDataChange 的调用)。

GetEnable：获取当前的回调允许值。

⑤ IConnectionPointContainer 接口

数据存取规范 2.0 组对象必须实现此接口，与服务器对象的此接口的唯一区别是管理的出接口不一样，组对象管理的出接口是 IOPCDataCallback 接口，可使客户与服务器连接并进行最有效的数据传送。其接口方法的行为只是将服务器对象的 IOPCShutdown 接口换成 IOPCDataCallback 接口即可，可参考服务器对象的实现。

⑥ IEnumOPCItemAttributes 接口

IEnumOPCItemAttributes 接口允许客户找出组对象内的项以及项的相关属性 (以项属性结构作为枚举对象)，它不是组对象实现的一个接口，只能通过 IOPCItemMgt::CreateEnumerator 方法创建相应枚举器的实例，不可通过接口查

询获得。其接口方法与普通枚举接口方法功能相同。

Next：获取枚举器的后 **n** 个项属性结构对象。

Skip：跳过后 **n** 个项属性结构对象。

Reset：使枚举器返回到第一个项属性结构对象处。

Clone：创建枚举器第二个副本。新枚举器与当前枚举器的状态相同。

第 3 章　基于 OPC DA 2.05a 规范的 OPC DA 服务器设计

本章利用 Visual C++ ATL 以一个实例来说明具体的开发 COM 组件步骤，随后开发了基于 OPC DA 2.05a 规范的 OPC DA 服务器。

3.1 开发 OPC 服务器的动态模板库（ATL）

3.1.1 ATL 简介

自从 1993 年 Microsoft 首次公布了 COM 技术以后，Windows 平台上的开发模式发生了巨大的变化，以 COM 为基础的一系列软件组件化技术将 Windows 编程带入了组件化时代。但 COM 开发技术的难度和烦琐的细节令人望而却步，COM 编程一度被视为一种高不可攀的技术，。开发人员希望能够有一种方便快捷的 COM 开发工具，提高开发效率，更好地利用这项技术。

针对这种情况，Microsoft 公司在推出 COM SDK 以后，为简化 COM 编程，提高开发效率，采取了许多方案，特别是在 MFC（Microsoft Foundation Class）中加入了对 COM 和 OLE 的支持。但是随着 Internet 的发展，分布式的组件技术要求 COM 组件能够在网络上传输，而又尽量节约宝贵的网络带宽资源。采用 MFC 开发的 COM 组件由于种种限制不能很好地满足这种需求，因此 Microsoft 在 1995 年又推出了一种全新的 COM 开发工具 ATL。

ATL 是 ActiveX Template Library 的缩写，它是一套 C++模板库。使用 ATL 能够快速地开发出高效、简洁的代码（Effective and Slim code），同时对 COM 组件的开发提供最大限度的代码自动生成以及可视化支持。为了方便使用，从 Microsoft Visual C++ 5.0 版本开始，Microsoft 把 ATL 集成到 Visual C++开发环境中。1998 年 9 月推出的 Visual Studio 6.0 集成了 ATL 3.0 版本。目前，ATL 已经成为 Microsoft 标准开发工具中的一个重要成员，主要具有以下特点：

（1）ATL 的基本目标就是使 COM 应用开发尽可能地自动化，这个基本目

标就决定了 ATL 只面向 COM 开发提供支持。目标的明确使 ATL 对 COM 技术的支持达到淋漓尽致的地步。对 COM 开发的任何一个环节和过程，ATL 都提供支持，并将与 COM 开发相关的众多工具集成到一个统一的编程环境中。对于 COM/ActiveX 的各种应用，ATL 也都提供了完善的 Wizard 支持。所有这些都极大地方便了开发者的使用，使开发者能够把注意力集中在与应用本身相关的逻辑上。

（2） ATL 因其采用了特定的基本实现技术，摆脱了大量冗余代码，使用 ATL 开发出来的 COM 应用的代码简练高效。ATL 在实现上尽可能采用优化技术，甚至在其内部提供了所有 C/C++开发的程序所必须具有的 C 启动代码的替代部分。同时 ATL 产生的代码在运行时不需要依赖于类似 MFC 程序所需要的庞大的代码模块，包含在最终模块中的功能是用户认为最基本和最必须的。这些措施使采用 ATL 开发的 COM 组件(包括 ActiveX Control)可以在网络环境下实现应用的分布式组件结构。

（3） ATL 的各个版本对 Microsoft 的基于 COM 的各种新的组件技术如 MTS、ASP 等都有很好的支持，ATL 对新技术的反应速度大大快于 MFC。ATL 已经成为 Microsoft 支持 COM 应用开发的主要开发工具，因此 COM 技术方面的新进展在很短的时间内都会在 ATL 中得到反映。这使开发者使用 ATL 进行 COM 编程可以得到直接使用 COM SDK 编程同样的灵活性和强大的功能。

3.1.2 ATL 开发 COM 组件的实现

本节以一个 COM 组件的开发实例来说明基于 VISUAL C++6.0 ATL 开发组件的步骤，本实例的组件只有一个方法和一个属性：方法为发出嘟嘟的响声，属性为响声的次数。

（1）利用 ATL 应用程序向导创建应用程序

从 Visual Studio 菜单中选择 File -> New，选中 Projects 选项，然后选择 ATLCOM AppWizard，输入工程名和保存的路径，如图 3-1 所示。

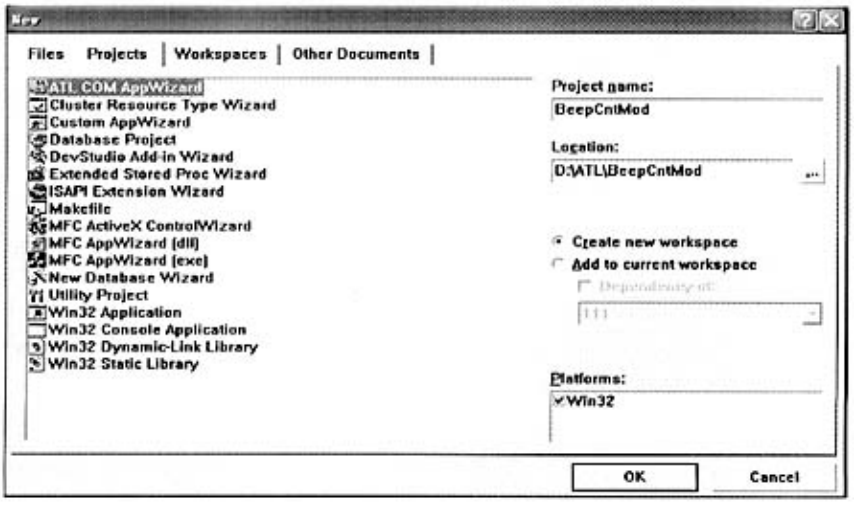

图 3-1 利用 ATL 应用程序向导创建工程界面

点击 OK，然后选择组件的服务类型为进程内组件：动态连接库(DLL)，如图 3-2 所示。

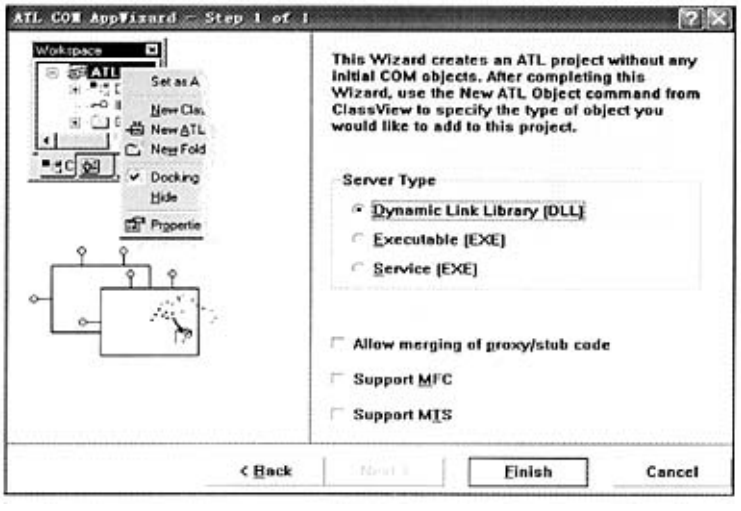

图 3-2 ATL 应用程序向导创建工程的类型选择界面

（2）工程源文件说明

经过 ATL 应用程序向导创建得到的应用程序仅仅是一个程序框架，生成的文件如下：

一个全局成员：CComModule _Module；

一个包含着最初的类型库的说明文件：BeepCntMod.idl；

一个.def 文件；

一个.rc 资源文件；

一个包含着资源 ID 定义的头文件；

Stdafx.h 和 Stdafx.cpp 文件。

可以看出，与 MFC 应用程序唯一不同的就是多了一个后缀为.idl 的文件。每一个 ATL 工程都有一个与工程同名的 IDL 文件，此文件记录了该工程所用到的 COM 接口或 COM 对象的定义。ATL 应用程序向导可以自动维护此 IDL 文件，同时我们也可以手动地修改此文件来加入需要的 COM 接口的 IDL 定义，在 3.2 节创建 OPC DA 服务器的时候就将手动修改自动生成的 IDL 文件来加入 OPC 的一系列接口。

（3）添加组件对象

最简单的添加组件对象的方法是通过 ATL Object Wizard。从 Insert 菜单下选择 NEW ATL Object...,将出现如图 3-3 所示的对话框。

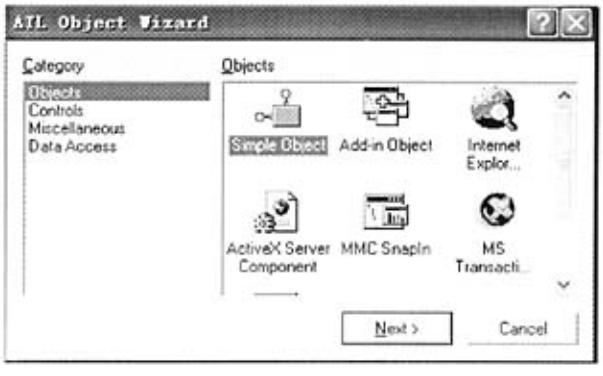

图 3-3　ATL 添加组件对象向导

选择 Simple Object,单击 Next,在 Names 页的 ShortName 字段输入组件的名称 BeepCnt,其它的 7 个编辑框的内容将自动生成，如图 3-4 所示。

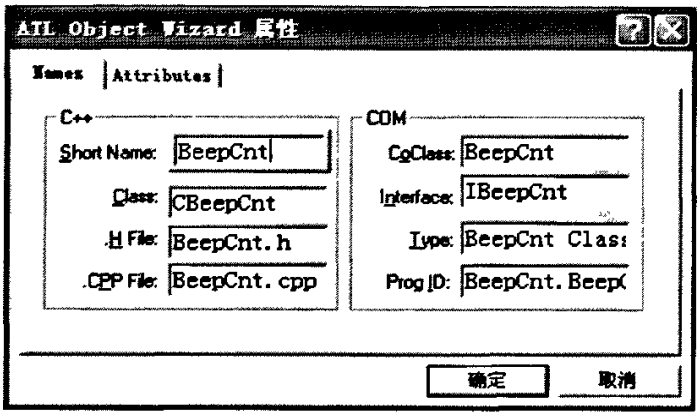

图 3-4 ATL 添加组件对象的属性向导（1）

单击 Attributes 页，出现如图 3-5 的对话框。Threading Model 选项选择 Apartment，表示这个对象可以被一个或多个线程调用。Interface 选项选择 Dual，表示创建的接口为双重接口：既支持自动化接口又支持自定义接口。Aggregation 选项选择 Yes，表示组件支持聚合，并且该聚合功能由 ATL 自动完成。ISupportErrorInfo 表示创建支持将错误信息返回给客户的接口，本例不选择该功能。Connection Points 选项表示通过对象的类导出而启用对象的连接点，本例不选择该功能。Free-Threaded Marshaller 选项表示支持创建自由线程封送拆收器对象，以有效地在同一进程中的两个线程之间封送指针，本例不选择该功能。

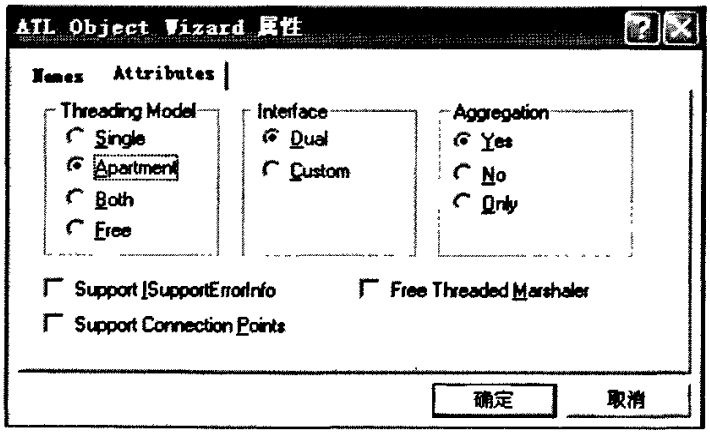

图 3-5 ATL 添加组件对象的属性向导（2）

创建完成后的组件对象支持双接口（自动化接口和自定义接口）、聚合和多线程访问。现在工程中又多了三个 ATL 向导创建的文件：beepcnt.reg、beepcnt.cpp 和 beepcnt.h。

beepcnt.reg 包含了 ATL 处理的代码的源脚本，大部分是为了 COM 组件运行时能够找到组件的注册入口。

beepcnt.cpp 文件基本是空的，因为该组件对象还没有添加任何属性和方法，所有添加的对象属性和方法都将在 beepcnt.cpp 文件中实现。

beepcnt.h 文件实现了对象类的定义：

```
class ATL_NO_VTABLE CBeepCnt :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CBeepCnt, &CLSID_BeepCnt>,
public IDispatchImpl<IBeepCnt, &IID_IBeepCnt, &LIBID_BEEPCNTMODLib>
{
  public:
   CBeepCnt()
     {
      }
  DECLARE_REGISTRY_RESOURCEID(IDR_BEEPCNT)
  DECLARE_PROTECT_FINAL_CONSTRUCT()
  BEGIN_COM_MAP(CBeepCnt)
  COM_INTERFACE_ENTRY(IBeepCnt)
  COM_INTERFACE_ENTRY(IDispatch)
  END_COM_MAP()
  // IBeepCnt
  public:
};
```

可以看出，CBeepCnt 类是由三个类派生出来的：CComObjectRootEx 操作了组件的引用记数；CComCoClass 用来定义该对象的默认类工厂和聚合模型；IDispatchImpl 提供了一个双重接口 IBeepCnt，接口 ID 是 IID_IBeepCnt，支持自动化接口和自定义接口。

（4）添加组件对象的属性和方法

上面生成的组件对象是一个空对象，下面为该对象添加属性和方法。从 Class View 中右击 IbeepCnt 接口，选择“ADD Method...”，出现如图 3-6 所示的对话框。

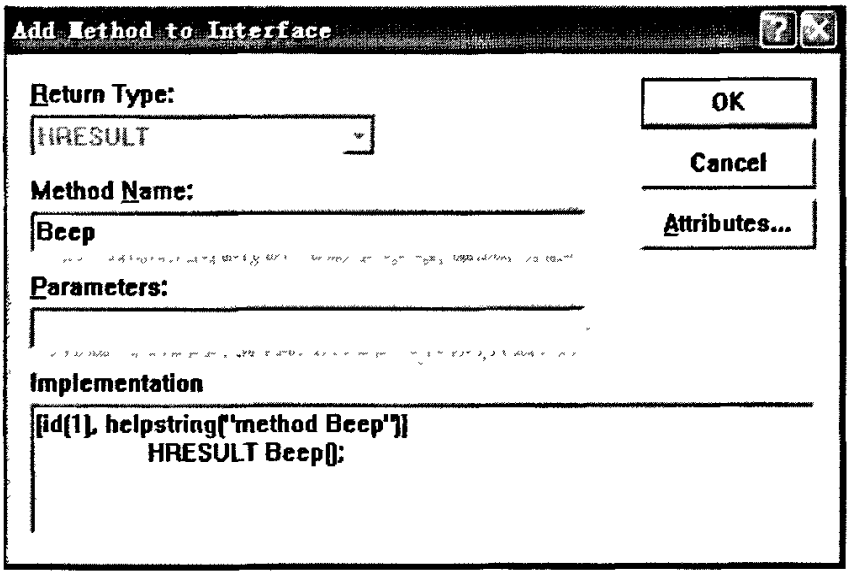

图 3-6 为组件对象添加方法

在 Return Type 选择返回类型为 HRESULT，方法名选项 Method Name 填 Beep，参数选项 Parameters 为空，单击 OK，完成 Beep 方法的添加。ATL 向导会把所有 Beep 方法的定义代码自动添加到 BeepCnt.cpp 和 BeepCnt.h 文件中。

添加组件属性的方法同上：右击 IbeepCnt 接口，选择“Add Property...”，出现如图 3-7 所示的对话框，选择合适的返回类型和属性的类型，输入属性的名称，单击 OK 即完成组件属性的添加。

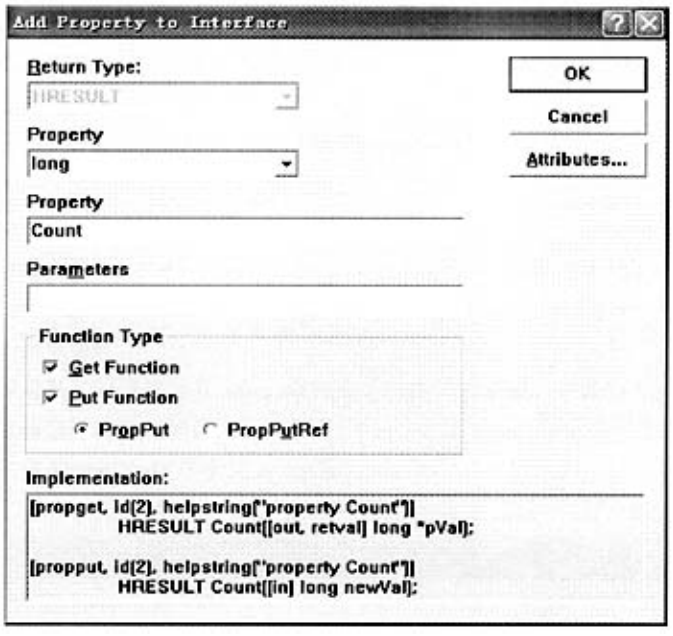

图 3-7　为组件对象添加属性

完成对组件方法和属性的添加后，ATL 向导向 idl 文件添加了方法和属性的定义。在 BeepCnt.h 中添加了对三个新函数的声明，BeepCnt.cpp 文件添加了这三个函数的框架。在这三个函数的框架里面分别添加一些代码：设置计数器、获得计数器、发出所设置数的嘟嘟声。添加代码后的函数如下：

```
STDMETHODIMP CBeepCnt::Beep()                    //发出所设置数的嘟嘟声
{
    for (int i = 0; i < cBeeps; i++)              //cBeeps 是定义的全局变量
        MessageBeep((UINT) -1);
    return S_OK;
}
STDMETHODIMP CBeepCnt::get_Count(long *pVal)   //获得计数器
{
```

```
    *pVal = cBeeps;
    return S_OK;
}
STDMETHODIMP CBeepCnt::put_Count(long newVal)   //设置计数器
{
    cBeeps = newVal;
    return S_OK;
}
```

（5）测试组件

添加完组件的属性和方法后，就完成了对组件的创建。下面用 Visual Basic 来实现对该组件功能的测试。

首先在 Visual Basic 中引用 BeepCntMod，如图 3-8 所示。

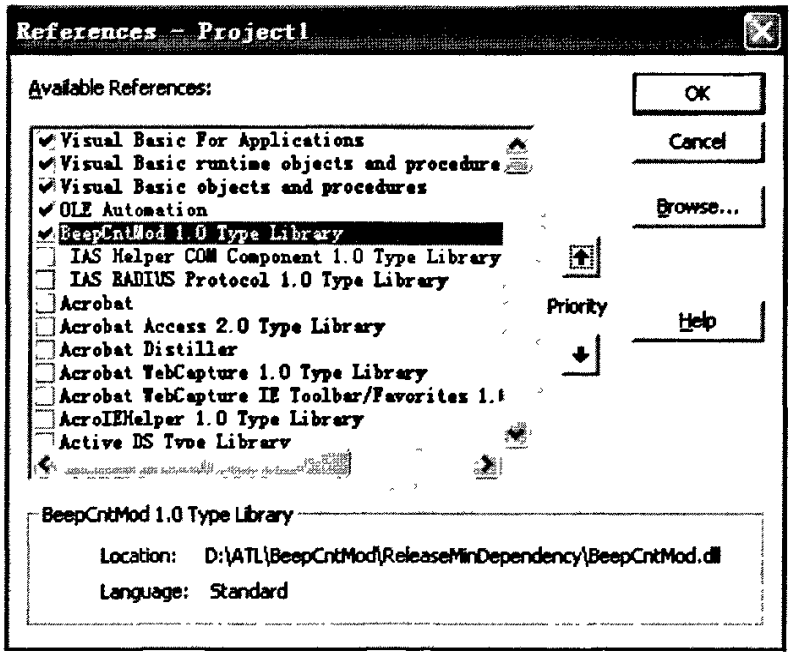

图 3-8 Visual Basic 引用组件 BeepCntMod

然后在 Visual Basic 中新建一个工程，创建一个如图 3-9 所示的对话框，Visual Basic 生成的代码修改后如下：

```
Dim BeeperCnt As BeepCnt
Private Sub Beep_Click()
    Text1 = BeeperCnt
    BeeperCnt.Beep
End Sub
Private Sub Set_Click()
    BeeperCnt = Val(Text1)
    Text1 = BeeperCnt
End Sub
Private Sub Form_Load()
    Set BeeperCnt = New BeepCnt
    Text1 = BeeperCnt
End Sub
```

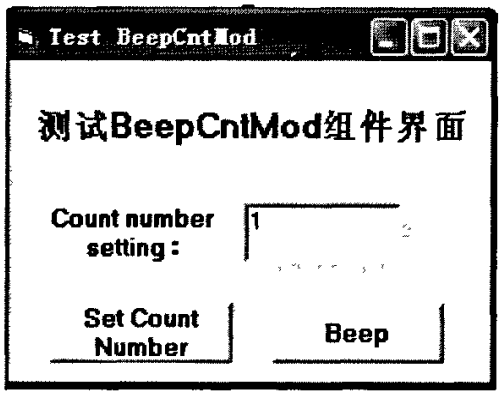

图 3-9 测试 BeepCntMod 组件的界面

运行这个程序，将听到嘟嘟声，而且响声的次数由 count 字段设置。

3.2 OPC DA 服务器的设计与实现

3.2.1 OPC DA 服务器的整体结构

OPC 服务器首先建立类厂，通过类厂，客户程序可以创建 OPC 服务器。

客户程序通过 OPC 服务器的名字找到 OPC 服务器在注册表的信息，然后调用 CoCreateInstance( )在服务器中创建一个 OPCServer。通过 OPCServer 的接口来实现一系列的功能，如创建 Group，获得服务器状态等。创建 Group 对象后，通过 Group 的接口来实现一系列的功能，如添加 Item，删除 Item 等。图 3-10 详细说明了对 OPC 服务器进行访问时的流程[25]。

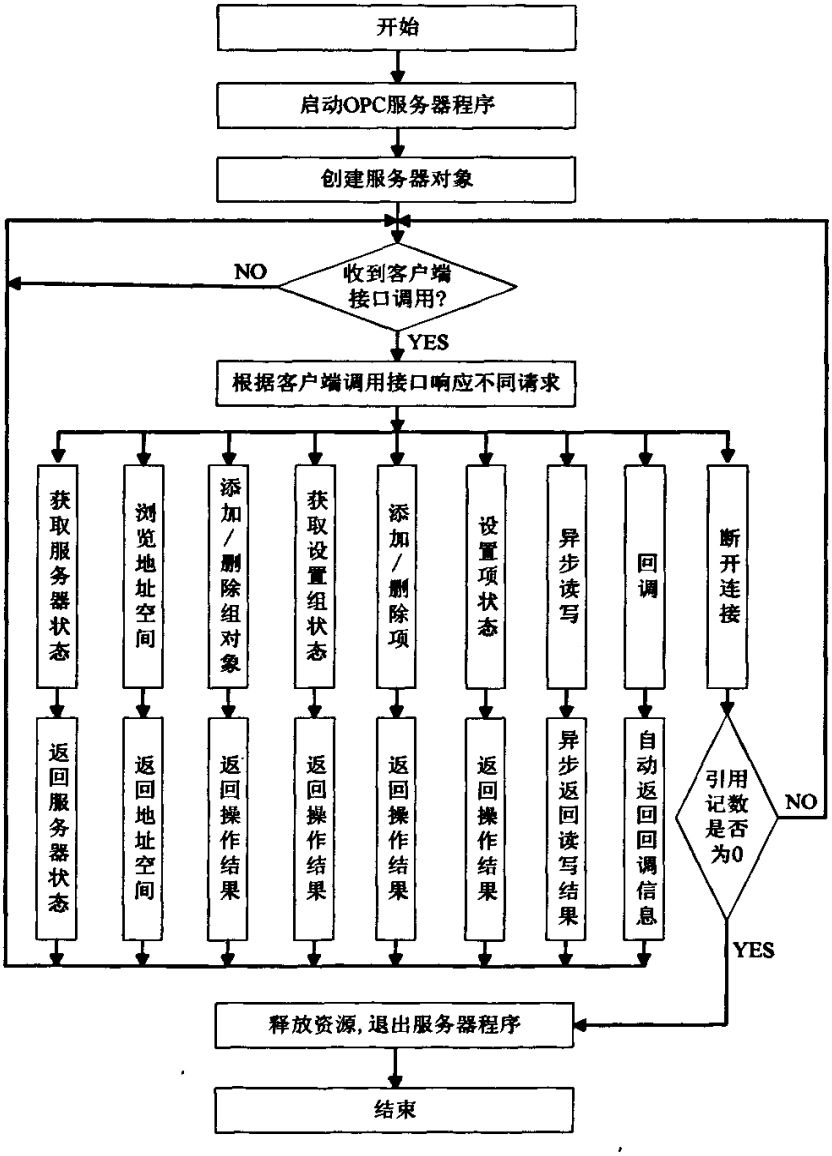

图 3-10 OPC 服务器的整体结构图

3.2.2 基于同步通信的 OPC DA 服务器的编程实现

（1）利用 ATL 应用程序创建 OPCDA 服务器程序

打开 VC++6.0 开发环境，选择 ATL COM AppWizard，同 3.1.2 节利用 ATL 创建组件的方法类似，选择目录 D:\OPCDA 和填写工程名 OPCDA，现在的 IDL 文件如下：

```
//OPCDA.idl: IDL source for OPCDA.dll
//This file will be processed by the MIDL tool to
//produce the type library(OPCDA.tlb) and marshalling code.
import"oaidl.idl";
import"ocidl.idl";
[
uuid (9DB24EAC-C452-477B-8B70-87F51F3330D),
version (1.0),
helpstring("OPCDA 1.0 Type Library")
]
library OPCDALib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");
};
```

打开按照 OPCDA2.05a 规范定义的 IDL 文件，并将该文件关于 OPCDA 接口定义的内容复制到本工程中的 IDL 文件的最前面，然后在工程的最后处对 library OPCDALib 修改如下：

```
library OPCDALib
{
   importlib("stdole32.tlb");
   importlib("stdole2.tlb");
   interface IOPCServer ;
   interface IOPCServerPublicGroups ;
   interface IOPCBrowseServerAddressSpace;
```

```
    interface IOPCGroupStateMgt ;
    interface IOPCPublicGroupStateMgt ;
    interface IOPCSyncIO ;
    interface IOPCAsyncIO ;
    interface IOPCItemMgt;
    interface IEnumOPCItemAttributes ;
    interface IOPCDataCallback ;
    interface IOPCAsyncIO2 ;
    interface IOPCItemProperties ;
};
```

这样修改的目的是为了将 OPC 规范中定义的接口全部引入到我们所创建的工程中，编译后即引入了 OPC 规范中定义的所有接口。

（2）创建 Server 和 Group 对象

创建 Server 和 Group 对象的方法同 3.1.2 节。对象名分别为 TestServer 和 TestGroup，它们都是由标准的 OPC Server 和 Group 对象所派生。

（3）为 TestServer 和 TestGroup 对象添加方法

为对象添加方法的详细步骤同 3.1.2 节。两个对象的所有接口方法都应严格按照 OPC DA 2.05a 规范的定义来实现。下面仅给出 TestServer 对象的 IOPCServer 接口的方法添加后的定义：

```
STDMETHODIMP   AddGroup(
                    LPCWSTR         szName,
                    BOOL            bActive,
                    DWORD           dwRequestedUpdateRate,
                    OPCHANDLE       hClientGroup,
                    LONG            * pTimeBias,
                    FLOAT           * pPercentDeadband,
                    DWORD           dwLCID,
                    OPCHANDLE       * phServerGroup,
                    DWORD           * pRevisedUpdateRate,
                    REFIID            riid,
                    LPUNKNOWN       * ppUnk );
```

```
STDMETHODIMP   GetErrorString(
                HRESULT          dwError,
                LCID             dwLocale,
                LPWSTR           * ppString );
STDMETHODIMP   GetGroupByName(
                LPCWSTR          szName,
                REFIID           riid,
                LPUNKNOWN        * ppUnk );
STDMETHODIMP   GetStatus(
                OPCSERVERSTATUS ** ppServerStatus );
STDMETHODIMP   RemoveGroup(
                OPCHANDLE        hServerGroup,
                BOOL             bForce);
STDMETHODIMP   CreateGroupEnumerator(
                OPCENUMSCOPE     dwScope,
                REFIID           riid,
                LPUNKNOWN        * ppUnk);
```

添加完 TestServer 和 TestGroup 对象的所有方法后，重新编译工程，在 Class Veiw 中将看到 TestServer 和 TestGroup 对象，双击各对象的接口将出现该接口的所有方法。如图 3-11 所示。

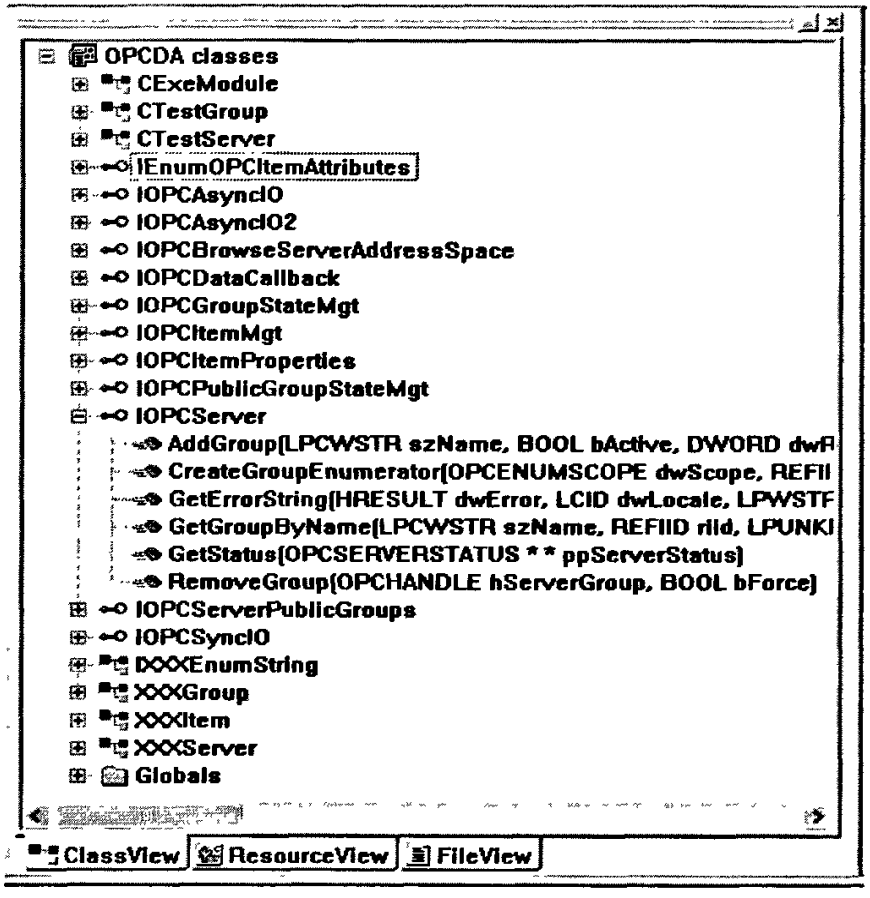

图 3-11 添加完对象的所有方法后的 Class View

3.2.3 OPC DA 服务器类的实现

现在虽然生成了 TestServer 和 TestGroup 对象的框架，但还没有实现 TestServer 对 TestGroup 接口的访问，也没有实现对 TestServer 和 TestGroup 对象的管理。本节将实现这两个功能。

（1）TestServer 对象对 TestGroup 对象接口的聚合

在 OPC DA 服务器程序中，只有一个 CLSID 定义，这表示客户应用程序在访问服务器时只需要创建一个 Server 对象，Group 对象的接口是通过对 Server 对象的创建来实现的。COM 组件的可重用性包括包容和聚合，本文选择聚合来实现 Server 对象对 Group 对象接口的支持，如图 3-12 所示。

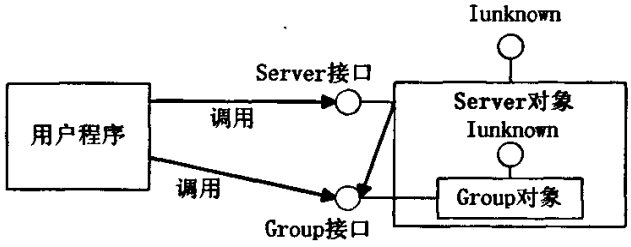

图 3-12 Server 对象对 Group 对象的聚合示意图

在 TestServer.h 文件对 TestServer 类的定义处添加如下代码来定义 Server 对象对 Group 接口的聚合以及定义在创建 Server 对象时创建 Group 对象。

```
BEGIN_COM_MAP(CTestServer)
    COM_INTERFACE_ENTRY(IOPCServer)
    COM_INTERFACE_ENTRY_AGGREGATE(IID_IOPCItemMgt, m_pms)
    COM_INTERFACE_ENTRY_AGGREGATE(IID_IOPCAsyncIO2, m_pms)
    COM_INTERFACE_ENTRY_AGGREGATE(IID_IOPCGroupStateMgt,
m_pms)
END_COM_MAP( )
CComObject<CTestGroup> * m_pms[10];
HRESULT  FinalConstruct( )
    {
        HRESULT hr;
        for(int j=0;j<10;j++)
        {
        hr=CComObject<CTestGroup>::CreateInstance(&m_pms[j]);
        if(FAILED(hr))return hr;
        m_pms[j]->g_seqnum=j;
        m_pms[j]->m_Parent=this;
        }
        return hr;
    }
HRESULT  FinalRelease( )
```

```
{
    for(int j=0;j<10;j++)
    m_pms[j]->Release( );
    return S_OK;
}
```

下面对以上程序做以下说明：

COM_INTERFACE_ENTRY_AGGREGATE(IID, PUNK)是 ATL 的宏，用来定义要聚合的对象接口，IDD 表示 GUID，PUNK 代表 IUknown 指针。该宏共被调用三次，分别实现 TestServer 对象对 TestGroup 对象的三个主要接口：IOPCItemMgt、IOPCAsyncIO2、IOPCGroupStateMgt 的聚合。

CComObject<CTestGroup>* m_pms[10]定义了 10 个 Group 对象指针，同时也表示每个客户最多只能建立 10 个 Group 对象。

FinalConstruct( )函数用来实现创建聚合对象，这个函数在 CTestServer 类初始化时调用，并通过调用 CreateInstance 来创建聚合的对象 Group，由于我们定义了最大十个组对象，因此程序中循环的次数为十次。

FinalRelease( )函数用来实现聚合对象的释放，同时释放 IUknown 指针，该函数在 CTestserver 析构时调用，在次函数中释放了通过 FinalConstruct( )创建的 Group 对象。

（2）新建类来管理对象

按照前面的功能设计，同时为了遵循 OPC DA 规范，下面建立几个类来管理 TestServer、TestGroup 和 Item。设计思路如下：

XXXServer 类作为 TestServer 对象的容器，用来管理 Server 对象，并负责各个 Server 对象数据的刷新。

XXXGroup 类作为 TestGroup 对象的容器，用来管理 Group 对象，并负责各个 Group 对象数据的刷新和对 XXXItem 类的管理；检查异步读写通知；检查数据刷新；检查是否回调等。

XXXItem 类作为 Item 的容器，用来管理 Item 以及 Item 值的仿真实现等。

这三个类分别在 XXXServer.h、XXXGroup.h 和 XXXItem.h 三个文件中定义。三个文件内容和注释详见附录 II、附录 III 和附录 IV。同时三个类所定义的函数分别在 XXXServer.cpp、XXXGroup.cpp 和 XXXItem.cpp 三个文件中实现，所有函数只要严格遵循 OPC DA 2.05a 规范都将很容易实现，限于篇幅，本文将

不列出这些文件。

3.3 OPC DA 服务器的异步通信实现

对于一个全面交互过程来说，同步通信往往不能满足要求。在异步读取数据时，OPC 服务器主动和客户应用程序通信，此时，OPC 服务器提供出接口，这些出接口由客户程序实现，并将接口指针通知 OPC 服务器对象，然后 OPC 服务器对象就利用此接口指针与客户程序进行通信。客户程序实现这些接口的对象被称为接收器(Sink,对 OPC 客户程序来说,其接收器至少要实现 IUnknown 和 IOPCDataCallBack 接口)。对客户程序来说，可以通过常规方式调用 OPC 服务器接口，也可以通过接收器接收 OPC 服务器发送的通知和事件。

异步通信就是通过 COM 机制中的连接点来实现的。可连接对象通过 IConnectionPointContainer 接口管理所有的出接口。对应与每一个出接口，可连接点对象又管理了连接点对象。为了使用连接点（IConnectionPointContainer 和 IConnectionPoint 接口），客户程序必须创建一个对象支持 IUknown 和 IConnectionPoint 接口，客户程序会传递一个指针给 IUknown 接口去激活服务器的连接点，可连接对象的基本结构如图 3-13 所示。

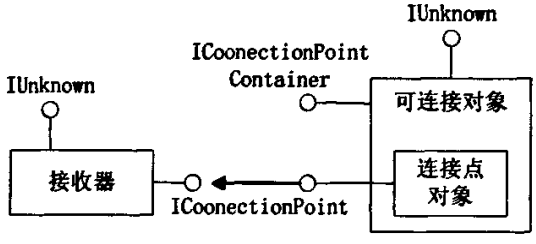

图 3-13 连接对象的基本结构图

IConnectionPoint 接口有两个重要的方法 Advise 和 Unadvise。客户（接收器）可以通过调用连接的点对象的 IConnectionPoint::Advise 来建立连接。通过 Advise 建立连接后，源对象（在此对应于 OPC 组对象）就可以激发事件或者向客户发出请求。对 OPC 组对象来说，客户调用异步读写操作或者组对象项成员的数据发生变化时，组对象就对客户程序发出事件，例如 OnDataChange 等。当客户需要取消连接时，调用带有同样连接标识的 Unadvise 即可。

在OPC DA 2.05a规范中定义了IOPCAsyncIO2、IConnectionPointerContainer和IOPCDataCallBack接口（该接口用于客户端）来支持对OPC服务器数据进行异步读写操作。为了实现异步通信功能，对3.2节OPC服务器同步通信程序的TestServer.h文件中的CTestGroup定义作如下修改：

```
BEGIN_COM_MAP(CTestGroup)
    COM_INTERFACE_ENTRY(IConnectionPointContainer)
    COM_INTERFACE_ENTRY(IOPCItemMgt)
    COM_INTERFACE_ENTRY(IOPCGroupStateMgt)
    COM_INTERFACE_ENTRY(IOPCAsyncIO2)
END_COM_MAP( )
BEGIN_CONNECTION_POINT_MAP(CTestGroup)      //添加的代码
    CONNECTION_POINT_ENTRY(IID_IOPCDataCallback)
END_CONNECTION_POINT_MAP( )
```

添加的代码是一个宏，这个宏用来通知IConnectionPointerContainer：只有一个连接点IID_IOPCDataCallback，也就是IOPCDataCallback接口。如果需要更多的连接点接口，就需要添加更多的CONNECTION_POINT_ENTRY( )宏。

下面为IOPCDataCallback接口添加方法，对3.2节OPC服务器同步通信程序的TestServer.h文件中的CTestGroup定义修改如下：

```
class ATL_NO_VTABLE CTestGroup :
        public CComObjectRootEx<CComSingleThreadModel>,
        public IConnectionPointContainerImpl<CTestGroup>,
        public IConnectionPointImpl<CTestGroup, &IID_IOPCDataCallback,
CComDynamicUnkArray>,
        public IDispatchImpl<IOPCItemMgt, &IID_IOPCItemMgt,
&LIBID_OPCDALib>     //添加的代码
```

同时加入以下宏和函数定义：

```
BEGIN_CONNECTION_POINT_MAP(CTestGroup)
    CONNECTION_POINT_ENTRY(IID_IOPCDataCallback)
END_CONNECTION_POINT_MAP()
STDMETHODIMP OnDataChange(
        DWORD dwTransid,
```

```
    OPCHANDLE hGroup,
    HRESULT hrMasterquality,
    HRESULT hrMastererror,
    DWORD dwCount,
    OPCHANDLE * phClientItems,
    VARIANT * pvValues,
    WORD * pwQualities,
    FILETIME * pftTimeStamps,
    HRESULT * pErrors
    );              //数据发生变化事件函数定义
STDMETHODIMP OnReadComplete(
    DWORD dwTransid,
    OPCHANDLE hGroup,
    HRESULT hrMasterquality,
    HRESULT hrMastererror,
    DWORD dwCount,
    OPCHANDLE * phClientItems,
    VARIANT * pvValues,
    WORD * pwQualities,
    FILETIME * pftTimeStamps,
    HRESULT * pErrors
    );              //读事件完成函数定义
STDMETHODIMP OnWriteComplete(
    DWORD dwTransid,
    OPCHANDLE hGroup,
    HRESULT hrMastererr,
    DWORD dwCount,
    OPCHANDLE * pClienthandles,
    HRESULT * pErrors
    );              //写事件完成函数定义
STDMETHODIMP OnCancelComplete(
```

```
    DWORD dwTransid,
    OPCHANDLE hGroup
    );          //取消事件完成定义
```

限于篇幅，本文不列出这些函数的具体实现。修改完成后的程序即可实现对 OPC DA 服务器的异步读写操作。

3.4 OPC 服务器的注册

根据 OPC 服务器支持的规范版本，OPC 服务器在注册表中的信息也不一样。OPC 基金会规定 OPC DA 2.0 规范的服务器在注册表中的信息为：{63D5F432-CFE4-11D1-B2C8-0060083BA1FB }，在开发 OPC DA 服务器的时候需要将其类别信息注册到系统中。

本文开发的 OPC DA 服务器的注册文件 OPCDA.reg 如下：

[HKEY_CLASSES_ROOT\CLSID\{7C13259A-74FD-4064-818F-C639E4B5811B}\Implemented Categories]

[HKEY_CLASSES_ROOT\CLSID\{7C13259A-74FD-4064-818F-C639E4B5811B}\Implemented Categories\{63D5F432-CFE4-11D1-B2C8-0060083BA1FB}]

其中{7C13259A-74FD-4064-818F-C639E4B5811B}为系统为本服务器的自动生成的 UUID，{63D5F432-CFE4-11D1-B2C8-0060083BA1FB}表示是基于 OPC DA2.0 规范的 OPC DA 服务器。

第 4 章　基于同步通信的 OPC 客户端应用程序设计

OPC 客户端应用程序开发分为两类：基于自动化接口和自定义接口。本章仅研究了基于自动化接口的 OPC 客户端应用程序开发。

4.1 客户端接口

4.1.1 IOPCDataCallback 接口

为支持此连接点，客户必须创建一个既支持 IUnknown 又支持 IOPCDataCallback 的接受器对象。客户向服务器 IConnectionPoint 的 Advise 方法传递 IUnknown 接口建立连接，然后服务器调用其 QueryInterface 方法获得 IOPCDataCallback 接口指针。接口指针可在组对象的数据变化时或 IOPCAsyncIO2 接口被调用时用到。

OnDataChange：当组对象的数据改变时和 Refresh 方法调用时服务器调用此方法通知客户进行数据处理。

OnReadComplete：当 IOPCAsyncIO2 接口异步读完成时服务器调用此方法通知客户进行数据处理。

OnWriteComplete：当 IOPCAsyncIO2 接口异步写完成时服务器调用此方法通知客户进行数据处理。

OnCancelComplete：当 IOPCAsyncIO2 接口异步取消操作完成时服务器调用此方法通知客户进行相关处理。

4.1.2 IOPCShutdown 接口

为支持此连接点，客户必须创建一个既支持 IUnknown 又支持 IOPCShutdown 的接受器对象。服务器获得 IOPCShutdown 接口指针的方法与 IOPCDataCallback 出接口相同。接口的 ShutdownRequest 方法在服务器需要切断连接时调用，客户应该用 UnAdvise 取消所有连接，移除所有组，并释放所有接口。当一个客户程序与多个 OPC 服务器相连时应该保存相对于每个对象的独

立的 ShutdownRequest 回调，这样每个服务器可以独立的切断服务。

4.2 基于同步通信的 OPC 客户端应用程序设计流程

采用自动化接口的客户应用程序相对于自定义接口的用户程序较简单，因此本文仅简要介绍基于自动化接口和同步通信的客户端应用程序开发步骤[28]。

（1）注册 OPC 服务器

利用 ATL 创建的 OPC 服务器，编译通过后，只要运行工程就自动完成了对 OPC 服务器在系统中的注册。于是，只要符合 OPC 标准的客户应用程序都可以访问来自任何生产厂商的 OPC 服务器程序。

（2）安装自动化接口服务

要保证系统目录下有文件：OPCDAAuto.dll，该文件可以从 OPC 基金会官方网站（www.opcfoundation.org）下载。在 VB 环境下可以通过 Project 的菜单 References 来引用"OPC Automation 2.0"，这样就可以使用自动化接口。

（3）连接 OPC 服务器

首先判断 OPC 服务器对象是否创建，如果没有则先创建一个 OPC 服务器对象；然后再判断 OPC 服务器对象是否处于连接状态，如果未连接，则调用 Connect 方法建立连接。与服务器建立好连接后就可以使用其属性如 ServerState、StartTime、CurrentTime、OPCGroups 等来获取服务器的状态、启动时间、当前时间、组对象等信息。

（4）添加 OPC 组和 OPC 项

添加组之前首先判断组聚合是否存在，如果不存在，则创建一个组聚合对象，然后判断该组是否已经存在。在添加项前，先判断项是否已经存在，如果不存在则创建一个项聚合对象，然后就可以调用项聚合的 AddItems 方法添加项[29]。

（5）同步读/写服务器数据

同步方式对 OPC 服务器提出数据访问请求后一直等待服务器的应答，实时性强，因此适合智能楼宇对实时性的苛刻要求。

（6）断开 OPC 服务器

在断开 OPC 服务器前，必须先从下到上进行释放工作，即先释放项集合中的项，然后释放项集合；清空组集合中的组，然后释放组集合[30]；判断 OPC 服

务器对象的状态，如果处于连接状态，则先断开连接，然后释放 OPC 服务器对象[31]。

4.3 基于 VB 的 OPC 客户端同步应用程序设计

本文设计的 OPC 客户端应用程序是以 iFox3.5 数据库为服务器，为方便起见，数据库中仅有两个字段：浮点型的模拟量和日期[32]。程序开发的遵循 4.1 节的流程。首先在 VB6.0 中新建工程，创建如图 4-1 所示的框体：

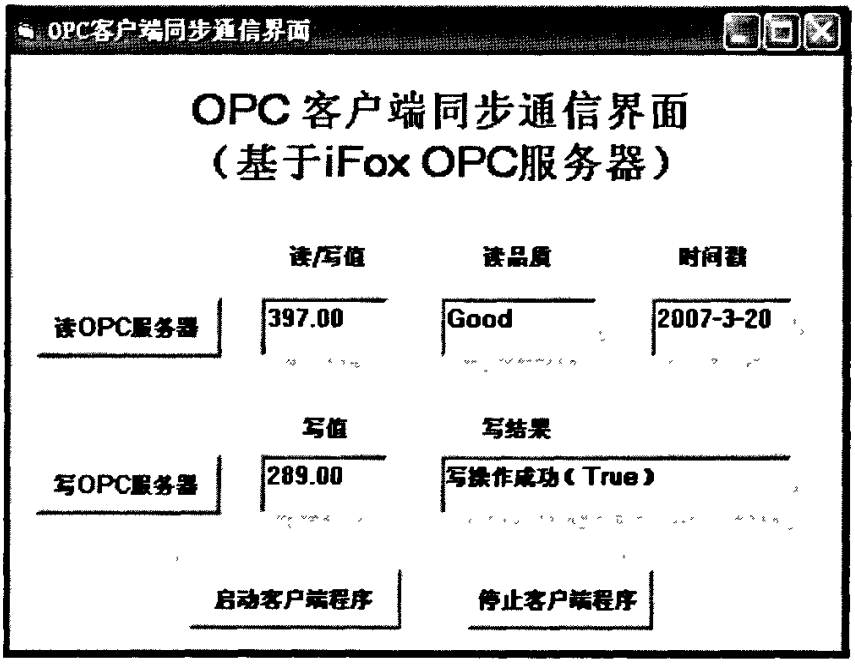

图 4-1 OPC 客户端同步应用程序界面

从 Project 的子菜单 Preferences 中引用 OLE Automation 和 OPC Automation Wrapper 2.0，如图 4-2 所示[33]。

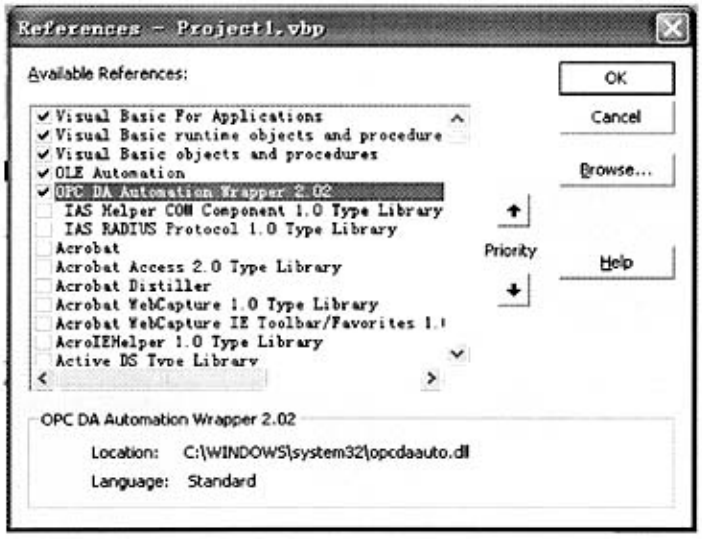

图 4-2 引用 OPC Automation Wrapper 2.0 动态连接库

在程序开始处首先声明 OPC 对象如下：

Dim WithEvent ServerObj As OPCServer

Dim WithEvent GroupObj As OPCGroup

Dim ItemObj As OPCItem

下面仅对按钮触发的事件做详细的介绍。

“启动客户端程序”按钮单击后完成以下功能：创建一个 OPC 服务器对象、连接服务器、添加组对象、为组对象添加项，详细程序如下所示。

“读 OPC 服务器”按钮单击后通过 ItemObj 对象的 Read 方法来获得服务器的数据，详细程序如下所示[34]。

“读 OPC 服务器”按钮单击后 GroupObj 对象的 SyncWrite 方法完成对服务器进行写操作。

“停止客户端程序”按钮单击后，依次完成对 ItemObj 和 GroupObj 对象的释放， 然后通过 ServerObj 对象的 Disconnect 方法断开与服务器的连接，最后释放 ServerObj 对象[35]。

以上四个按钮触发的事件在进行任何一个操作出错时，都会根据错误代码弹出具体的错误提示对话框，详细代码见附录 II。

第 5 章　基于 OPC 技术的智能楼宇监控系统设计

楼宇系统集成工程涉及的子系统主要包括楼控、消防、考勤门禁、紧急广播、保安监控和停车场系统等。本章主要研究了 OPC 服务器在楼宇消防系统中的运用，集成软件采用霍尼韦尔(Honeywell)公司的 EBI(Enterprise Buildings Integrator)，通过 EBI 主控界面来控制消防系统的 CAN 节点设备[36]。

5.1 系统框架结构

现有住宅小区需安装消防设备，在这里将各个烟雾传感器作为 CAN 节点，连接成一个小区内部 CAN 网络，服务器作为 CAN 的另一节点，通过 CAN—USB 网关挂接在 CAN 总线上，可以方便地实现对节点的监控。网络拓扑结构如图 5-1 所示。

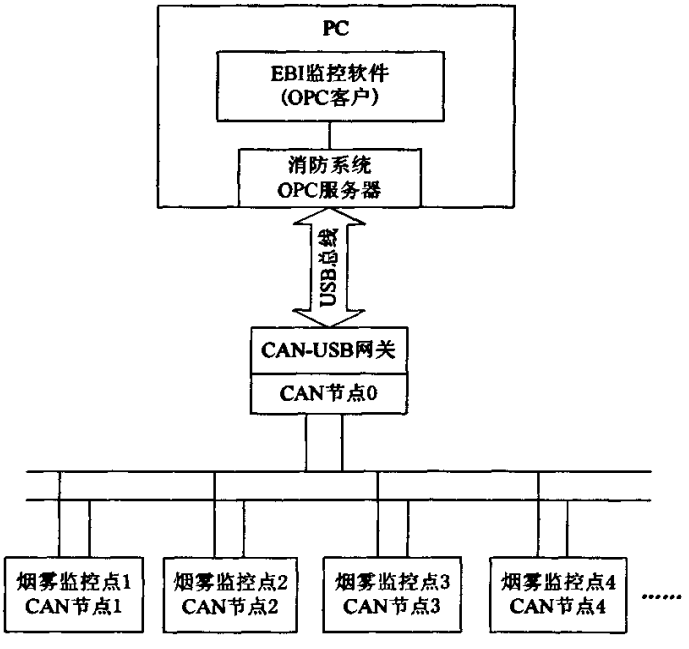

图 5-1 系统结构框图

在服务器节点端，这里采用 OPC 接口，屏蔽了底层的细节，提供统一的通用接口。这样可以提高系统的兼容性，极大地提高软件开发效率。同时利用该服务器的 OPC 接口可以方便的与其他网络进行对接，实现信息的交互，例如可以接入 Internet 来实现网络的远程监控。

5.2 系统硬件实现

（1）监控子节点设计

烟雾传感器采用深圳市易佳杰电子科技有限公司的 NIS-09C，该传感器提供三根接线：VCC、GND 和烟雾报警输出信号。输出信号在正常工作状态为高电平，当烟雾传感器所采集的烟雾浓度超过预定的阈值时，输出信号为低电平，此时触发 C8051F236 的中断，CAN 节点向 PC 监控软件发送烟雾报警信号。

微控制器采用 CYGNAL 公司的 C8051F236，CAN 总线接口使用 Philips 公司的独立 CAN 线控制器 SJA1000，并由光耦 6N137 进行总线隔离， CAN 总线收发器采用 MCP2551。监控子节点的硬件框图如图 5-2 所示。

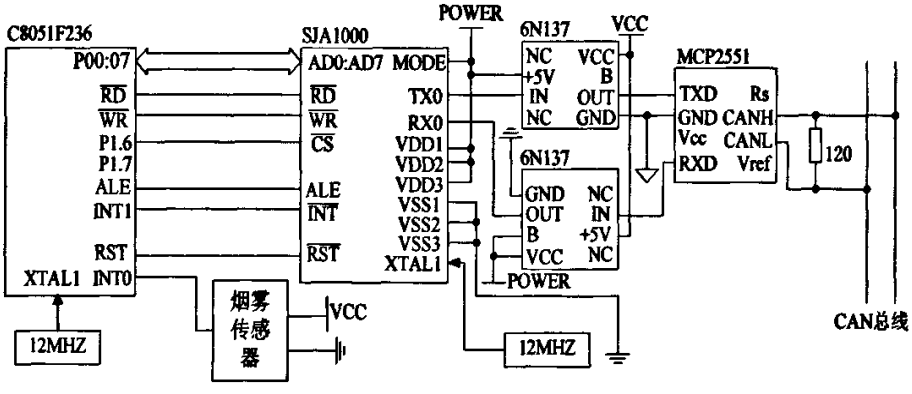

图 5-2 监控子节点硬件框图

（2）CAN-USB 网关设计

CAN-USB 网关的硬件电路和监控子节点大致相同，只增加了 CAN-USB 转换电路，如图 5-3 所示。

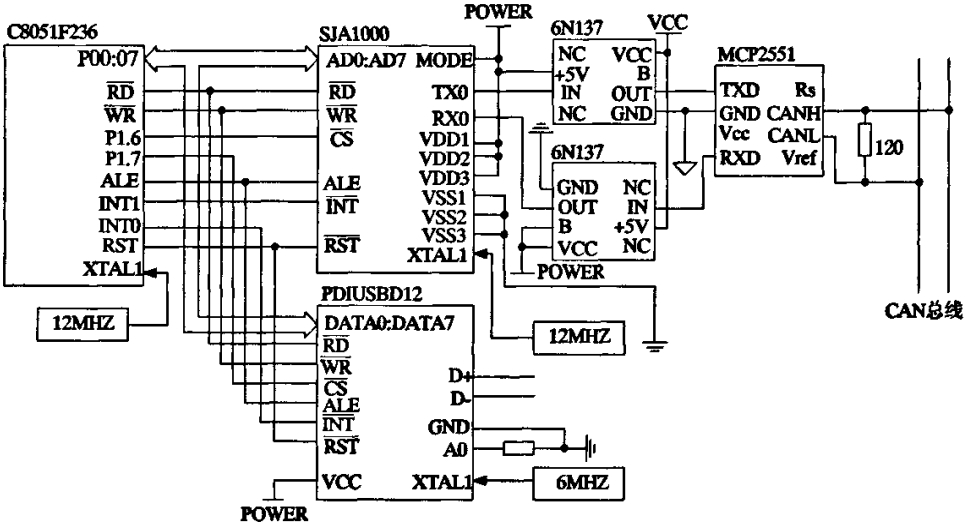

图 5-3 CAN-USB 网关硬件图

USB 接口器件为 PDIUSBD12，它是 Philips 公司的一种性能优化的 USB 器件。它支持本地的 DMA 传输，完全符合 USB1.1 版的规范，同时集成了 SIE (串行接口引擎)、FIFO 存储器、收发器以及电压调整器。其主端点的双缓冲配置增加了数据吞吐量并轻松实现实时数据传输。

5.3 系统软件实现

系统的软件设计分为下位机软件和上位机软件设计。下位机软件设计包括监控节点和 CAN-USB 网关软件；上位机软件包括消防系统的 OPC 服务器开发和 EBI 软件设计。

5.3.1 下位机软件实现

（1）监控节点软件设计

监控节点软件主要包括两个中断服务程序：烟雾量超过预定阈值的中断（INT0）和 CAN 数据接收中断。在 INT0 中断服务程序里向主机发送烟雾超过预定阈值的报警信号。CAN 数据接收中断服务程序流程图如图 5-4 所示。

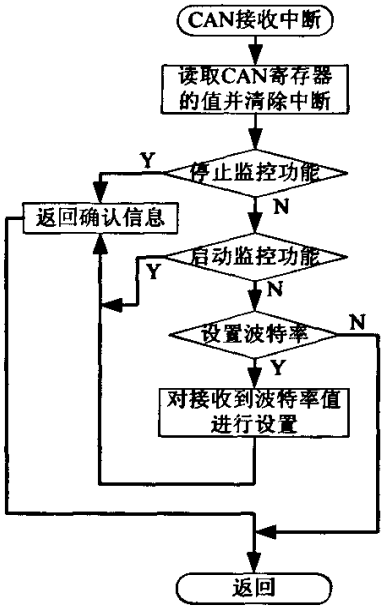

图 5-4 CAN 数据接收中断服务程序流程图

（2）网关软件设计

网关软件设计主要包括PDIUSBD12的中断服务程序和端点1OUT处理程序。

在PDIUSBD12的中断服务程序，用PDIUSBD12的端点1和端点2进行上位计算机与微控制器之间的命令和数据的传输。端点1和端点2设置成模式0 ,其中端点1 进行命令的传输和应答,端点2用于数据的传输。用USB总线进行数据传输时,必须定义一个数据传输的协议,否则主机和设备方就不同步,因为USB设备数据的发送是主机提出请求后再进行的，当主机没有发送IN令牌时,设备是不会发送数据的，而不管设备的数据是否写入PDIUSBD12器件的缓冲区。端点1接收上位机发送过来的命令，这些命令就是主机与设备共同协商好的命令字，程序对命令进行判断，如果是要求发送数据，则向端点2的IN索引写入数据。等待下次端点2的IN到来，所写入的数据即可被上位机接收。PDIUSBD12的中断服务程序流程图如图5-5所示。

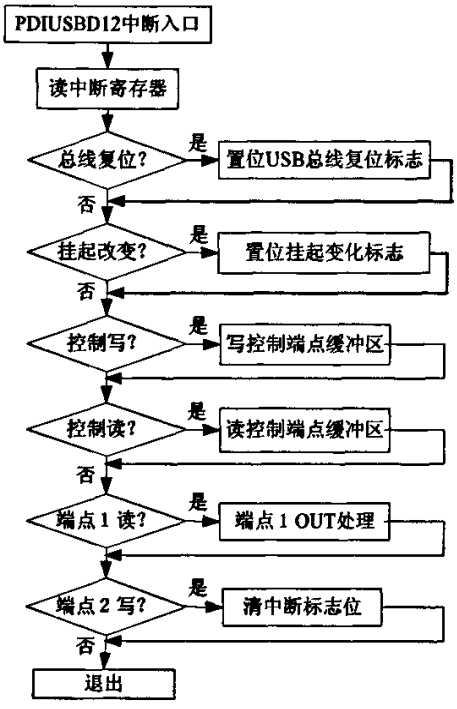

图 5-5 PDIUSBD12 的中断服务程序

端点 1 为 OUT 端点，主要用来配置 CAN 节点控制器 SJA1000 传输的波特率和通过 CAN-USB 网关向监控节点发送开始、停止监控的命令。端点 1 OUT 处理程序流程图如图 5-6 所示。

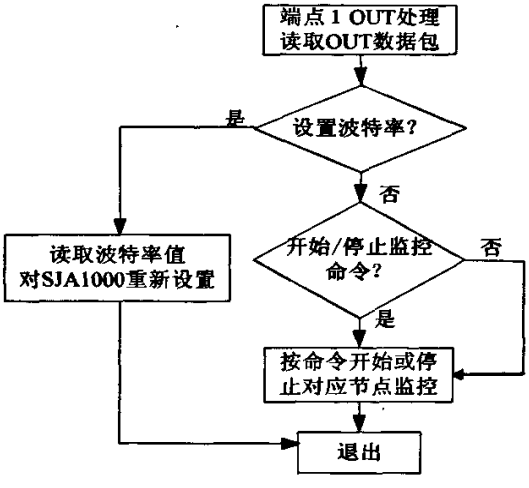

图 5-6 端点 1 OUT 处理流程图

5.3.2 PC 端软件实现

（1）消防控制系统的 OPC DA 服务器开发

通过深入分析消防控制系统的 USB 驱动程序，利用 Microsoft Visual C++6.0 的 ATL 开发了本系统的 OPC DA 服务器。在 IOPCSyncIO 接口的 Read 和 Write 方法实现对 USB 数据的读和写，包括启动、终止各监控节点和读取烟雾报警信号的数据。OPC DA 服务器的具体开发步骤见本文第 3 章。

（2）主机端应用软件 EBI

作为业界领先的企业级楼宇集成管理软件，EBI 为用户提供了一种高效、可靠、灵活的集成软件开发平台。EBI 软件平台主要包括：Quick Builder、Display Builder/HMI Web Display Builder 和 Station。

利用 Quick Builder 进行工程组态是整个开发过程的重点，下面对 EBI 中 OPC Client 接口的通道、控制器和点的配置中的关键问题进行说明[37]。

① 通道组态。定义对应于子系统通道的 OPC DA Server，正确配置各通道的如下参数：

HostName：连接的 OPC Server 所在的计算机名称，若 Server 与 EBI 平台安装在同一机器，则此处为 Local Host。

ProgID：OPC Server 程序在系统中的注册名称，由于 OPCDA 方式是基于 COM /DCOM 技术，所以只有 OPC Server 在操作系统中注册后，OPC 客户端才能够查询并连接到此 OPC 服务器[38]。

② 控制器组态。OPC 类型的控制器在概念上表示一个 OPC 组。根据实际工程的需要可以将服务器中的数据项进行适当的分组，以组的方式管理各个项。

③ 点组态。点的组态是将所需要的 OPC 项与 EBI 数据库中数据点相映射的过程。"Source Address"（源地址）和"DestAddress"（目的地址）的设置是点组态过程中的关键，其格式为：ControlleNameOPCItemName[DataFoanat]，其中，ControlleName 为控制器名称，OPCItemName 为 OPC 服务器中项名称，[DataFormat]为数据格式[39]。

按照上述方法完成组态配置，加载本 OPC 服务器的进程如图 5-7 所示。

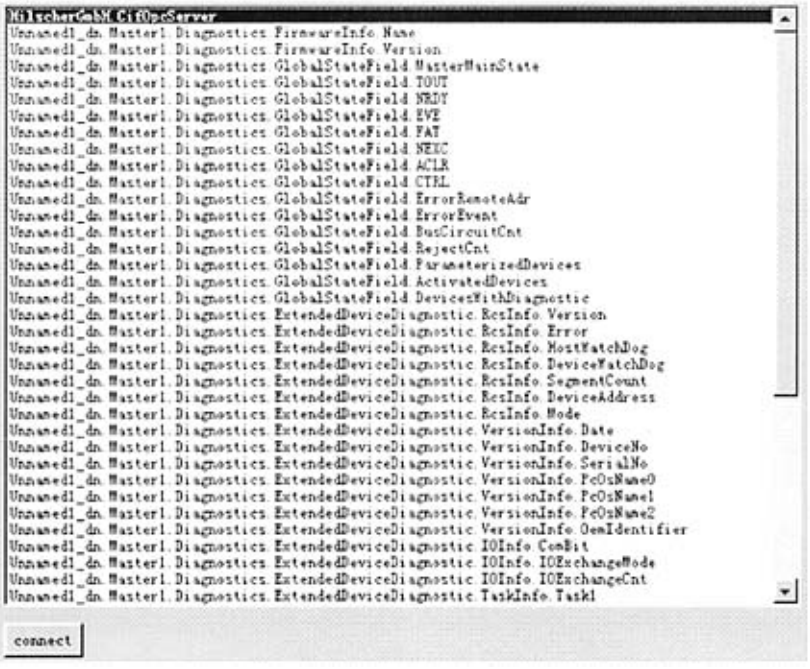

图 5-7 加载消防系统 OPC 服务器的进程图

创建 EBI 主控界面如图 5-8 所示，在 References 中引用已加载的消防系统的 OPC DA 服务器，即可完成对各监控节点，本实例仅监控 4 个节点。

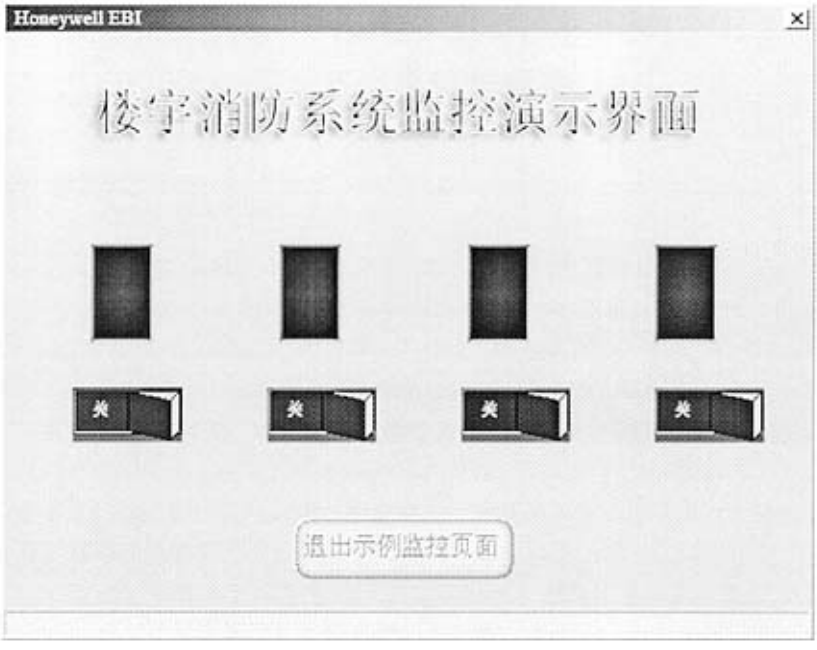

图 5-8 EBI 主控界面

启动监控系统，可以点击按钮独立控制每个监控节点的开关，当烟雾传感器输出的数据超过预定的阈值时，节点对应的按钮上面的黑色将变为红色，并闪烁。

5.4 系统测试结果分析

本系统采用的传感器报警时的烟雾浓度为 30%（误差为 5%），测试时，将监控节点放在一个密封性好的盒子，然后慢慢向盒子里释放烟雾，当烟雾传感器输出低电平时（烟雾浓度超过阈值），监控系统对应的黑色立即变为红色，并不断闪烁以表示警告。

通过测试，由于采用了 OPC 技术和与 OPC 兼容性比较好的 客户端应用软件 EBI，系统的响应时间快，而且在下位机的硬件升级时，主机端的监控软件不用升级，体现了 OPC 技术的优点，本系统达到了预定的目的[40]。

第6章 总结与展望

6.1 总结

本文对 OPC 规范进行了深入的分析，深入研究了接口描述语言，详细研究了 OPC 服务器和客户端应用程序的开发工具，并基于 Visual C++ ATL 和 Visual Basic 成功开发了 OPC DA 服务器以及 OPC 客户端应用程序，最后对 OPC 在智能楼宇消防监控系统的应用进行了研究[41]。

采用 OPC 技术后，由硬件开发商针对产品提供统一的 OPC 接口程序，应用程序端不需要了解硬件的实质和操作过程，直接调用 OPC 接口即可完成与设备之间的数据传输与控制，因此避免了重复开发驱动程序，大大降低了开发周期和开发经费。

本课题的研究成果基本达到了预定的目标：对楼宇子系统开发对应的 OPC 服务器，使各子系统具有较理想的开放性，并为系统集成和升级提供了方便。

6.2 展望

本课题仅研究了 OPC 规范中最重要的 OPC DA 服务器和客户端应用程序的开发，基于其它四个规范的服务器开发还有待研究；同时基于 OPC 服务器的智能楼宇远程监控的实时性和安全性还有待研究。

参考文献

[1] 唐小艳，陈立定，曾明．基于OPC技术的智能建筑系统集成的研究．安徽建筑工业学院学报，2004，4：38-40

[2] 李蕾，戴瑜兴．OPC数据存取服务器的实现．湖南工程学院学报，2005，2：19-22

[3] 王红爱．OPC CLIENT/SERVER开发方向研究:[硕士学位论文]．北京化工大学，2004

[4] 张小军．基于LonWorks技术的楼宇自控系统及其集成研究：[硕士学位论文]．哈尔滨理工大学，2003

[5] 吴由平，马旭东．OPC技术及其在智能楼宇系统集成中的应用．自动化技术，2006，3：90-92

[6] 苗雷，冯济缨．基于现场总线的OPC接口技术的研究．贵州工业大学学报，2004，6：30-32

[7] 刘庚．OPC及其在工业控制中的应用研究：[硕士学位论文]．河海大学，2005

[8] 周鸣，曲凌．基于OPC技术的楼宇自动化系统集成．煤炭工程，2005，11：78-80

[9] 吕勇，李友荣，王志刚等．基于OPC技术的设备远程监测与诊断系统．计算机管理与控制一体化，2005，6：65-67

[10] 李德华.基于LonWorks现场总线的楼宇自动化系统的研究及实现:[硕士学位论文].广东工业大学，2002

[11] 崔丽丽，徐进学．基于OPC技术的客户端数据采集软件包设计．沈阳工业大学学报，2005，5：553-555

[12] Schickhuber, Gerald McCarthy. Distributed fieldbus and control network systems. Oliver Source: Computing & Control Engineering Journal, v8, n1, Feb, 1997, p21-32

[13] 张英．基于LonWorks技术的楼宇自控系统及其集成研究：[硕士学位论文]．重庆大学，2005

[14] 刘快．基于OPC技术的实时控制系统研究与应用：[硕士学位论文]．浙江大学，2004

[15] 姚晓伟，陈在平，尹迅雷．基于OPC技术的现场总线系统集成研究．天津理工大学学报，2005，4：12-14

[16] 侯春生．OPC技术在现场总线控制系统中的应用：[硕士学位论文]．大连铁道学院，2005

[17] 张奇智，曹永灿．基于OPC技术网络控制系统仿真平台．信号与系统，2005,8:142-144

[18] 李冬辉，贾巍．基于 OPC 协议的智能建筑信息集成系统的设计．低压电器，2005,6：22-25

[19] 袁德平．OPC 技术在 PROFIBUS 现场总线中的研究与应用：[硕士学位论文]．西南交通大学，2001

[20] 王海瑞，钟家玉．用 Delphi 开发 OPC 客户端工具的方法研究．微计算机信息，2004,6：28-29

[21] 赵菁晶．智能楼宇监控系统软件平台设计与开发：[硕士毕业论文]．北京工业大学，2003

[22] 李石兵，王大林．智能建筑一体化集成系统中 LonWorks OPC 服务器的开发．智能建筑与城市信息，2005，7：105-107

[23] OPC Foundation. OPC Data Access Custom Interface Specification. Version2.04. 2000,09

[24] 阳宪惠．现场总线技术及其应用．北京：清华大学出版社，2001

[25] 朱耀春．OPC 数据存取服务器的开发与研究：[硕士学位论文]．华北电力大学，2003

[26] 刘国平，柳林林，刘利云．基于 OPC 服务器自动化接口的客户端程序的设计．自动化技术与应用，2005，9：33-35

[27] Holley. OPC unites industrial automation systems. Industrial Communications and Buses, 1997, p11-17

[28] 刘莉．用 VB 编写 OPC 客户端程序的方法．工业控制计算机，2005，5：5-6

[29] 李之明，高玉琢．Delphi 7 组件经典解析．北京：中国铁道出版社，2003

[30] 徐尔贵，丁雷．Visual Basic 教程．北京：清华大学出版社，2003

[31] Neitzel,Lee. OPC unified architecture internals.Technical Conference and Emerging Technologies Conference-Technical Papers Collection, 2004, p1051-1063

[32] Cauffriez,Laurent Conrard. Fieldbuses and their influence on dependability. Conference Record-IEEE Instrumentation and Measurement Technology Conference, v1, 2003, p83-88

[33] Lifang,Wangxiaoquan. Research and application of in-vehicle CAN bus evaluation platform. Chinese High Technology Letters, v15, n1, January, 2005, p58-61

[34] Anon. CAN interface. Elektron, v21, n6, June, 2004, p38-40

[35]Pinho,Luis Migue. Reliable Real-Time Communication in CAN Networks. IEEE Transactions on Computers, v52, n12, December, 2003, p1594-1607

[36] Thomas Nolin,Mikael. Real-time server-based communication with CAN. IEEE Transactions on Industrial Informatics, v 1, n 3, August, 2005, p192-200

[37] 李鑫，吴爱国，何熠．基于 OPC 技术楼宇系统集成的研究与实现．低压电器，2005，8：14-16

[38] Singh,Manjit. Building low-cost intelligent building components with Controller Area Network (CAN) bus. IEEE Region 10 International Conference on Electrical and Electronic Technology, 2001, p466-468

[39] Shengwei Wang,Zhengyuan Xu. Investigation on intelligent building standard communication protocols and application of IT technologies. Automation in Construction, v13, n5 SPEC. ISS., September, 2004, p607-619

[40] Sheble,Nicholas. OPC force decision. Source: InTech, v52, n10, October, 2005, p110

[41] Zhi Wang, Tianran Wang. Research and implementation of field bus interoperability. Proceedings of the World Congress on Intelligent Control and Automation (WCICA), v5, 2000, p3605-3610

致 谢

经过三年的学习，我的学位论文最终得以顺利完成。在三年的学习生活中，我要感谢的人很多，感谢他们在物质和精神上对我的支持。

感谢我的导师黄涛教授，忠心地感谢他在学习和生活中对我的帮助。导师活跃的学术思想，清晰的学术思路，严谨的治学态度，平易近人的学者风范，使我深受启迪和教育。导师对课题研究中的大力支持和悉心指导，使我受益匪浅。

感谢武汉理工大学智能信息系统研究所廖传书和卢珞先老师在项目研究以及论文撰写过程中给予我的关心和指导，以及所有师兄弟们对我的帮助。

感谢宁波诺依克电子有限公司黄金火教授在实习期间对我的指导。

感谢我的父母含辛茹苦地把我培养到现在，特别是在我母亲病重时仍然坚持让我完成学业，他们的养育之恩让我感动终生！感谢学习期间我的弟弟、弟妹以及我的未婚妻陈艳丽对我经济上的资助和精神上的鼓励。

感谢室友钟明、刘建伟、李新军等在我生活窘迫的时候对我的资助和鼓励。

最后谨向所有支持和关心我的人们致以最诚挚的感谢！

攻读硕士学位期间发表的学术论文

[1] 黄涛，王小辉. 2.4G 射频的 CAN 总线汽车故障诊断仪.单片机与嵌入式应用研究，2007 年，第 2 期

[2] 黄涛，王小辉. 基于车载环境的两种改进型 LPC 特征参数识别方法的研究. 计算机应用研究，2007 年 9 月刊

附录 I　遵循 OPCDA2.05a 规范的接口定义程序 OPCDA.idl

```
// OPCDA.IDL
import "oaidl.idl" ;
typedef enum tagOPCDATASOURCE {
                    OPC_DS_CACHE = 1,
                    OPC_DS_DEVICE } OPCDATASOURCE ;
typedef enum tagOPCBROWSETYPE {
                    OPC_BRANCH = 1,
                    OPC_LEAF,
                    OPC_FLAT} OPCBROWSETYPE;
typedef enum tagOPCNAMESPACETYPE {
                    OPC_NS_HIERARCHIAL = 1,
                    OPC_NS_FLAT} OPCNAMESPACETYPE;
typedef enum tagOPCBROWSEDIRECTION {
                    OPC_BROWSE_UP = 1,
                    OPC_BROWSE_DOWN,                OPC_BROWSE_TO}
OPCBROWSEDIRECTION;
// **NOTE** the 1.0 IDL contained an error for ACCESSRIGHTS.
// They should not have been an ENUM.
// They should have been two mask bits as noted here.
cpp_quote("#define OPC_READABLE      1")
cpp_quote("#define OPC_WRITEABLE     2")
typedef enum tagOPCEUTYPE {
                    OPC_NOENUM = 0,
                    OPC_ANALOG,
                    OPC_ENUMERATED } OPCEUTYPE;
typedef enum tagOPCSERVERSTATE {
                    OPC_STATUS_RUNNING = 1,
                    OPC_STATUS_FAILED,
                    OPC_STATUS_NOCONFIG,
                    OPC_STATUS_SUSPENDED,
                    OPC_STATUS_TEST } OPCSERVERSTATE;
typedef enum tagOPCENUMSCOPE { OPC_ENUM_PRIVATE_CONNECTIONS
= 1,
                    OPC_ENUM_PUBLIC_CONNECTIONS,
                    OPC_ENUM_ALL_CONNECTIONS,
                    OPC_ENUM_PRIVATE,
```

```
                OPC_ENUM_PUBLIC,
                OPC_ENUM_ALL } OPCENUMSCOPE;
typedef DWORD OPCHANDLE;
typedef struct tagOPCGROUPHEADER {
                DWORD       dwSize;
                DWORD       dwItemCount;
                OPCHANDLE   hClientGroup;
                DWORD       dwTransactionID;
                HRESULT     hrStatus;
} OPCGROUPHEADER;
typedef struct tagOPCITEMHEADER1 {
                OPCHANDLE   hClient;
                DWORD       dwValueOffset;
                WORD        wQuality;
                WORD        wReserved;
                FILETIME    ftTimeStampItem;
} OPCITEMHEADER1;
typedef struct  tagOPCITEMHEADER2 {
                OPCHANDLE   hClient;
                DWORD       dwValueOffset;
                WORD        wQuality;
                WORD        wReserved;
} OPCITEMHEADER2;
typedef struct  tagOPCGROUPHEADERWRITE {
                DWORD       dwItemCount;
                OPCHANDLE   hClientGroup;
                DWORD       dwTransactionID;
                HRESULT     hrStatus;
} OPCGROUPHEADERWRITE;
typedef struct tagOPCITEMHEADERWRITE {
                OPCHANDLE   hClient;
                HRESULT     dwError;
} OPCITEMHEADERWRITE;
typedef struct tagOPCITEMSTATE{
                OPCHANDLE   hClient;
                FILETIME    ftTimeStamp;
                WORD        wQuality;
                WORD        wReserved;
                VARIANT     vDataValue;
            } OPCITEMSTATE;
```

```
typedef struct tagOPCSERVERSTATUS {
                    FILETIME      ftStartTime;
                    FILETIME      ftCurrentTime;
                    FILETIME      ftLastUpdateTime;
                    OPCSERVERSTATE dwServerState;
                    DWORD         dwGroupCount;
                    DWORD         dwBandWidth;
                    WORD          wMajorVersion;
                    WORD          wMinorVersion;
                    WORD          wBuildNumber;
                    WORD          wReserved;
  [string] LPWSTR szVendorInfo;
} OPCSERVERSTATUS;
typedef struct tagOPCITEMDEF {
  [string]      LPWSTR      szAccessPath;
  [string]      LPWSTR      szItemID;
                BOOL        bActive ;
                OPCHANDLE   hClient;
                DWORD       dwBlobSize;
  [size_is(dwBlobSize)]     BYTE        * pBlob;
                VARTYPE     vtRequestedDataType;
                WORD        wReserved;
} OPCITEMDEF;
typedef struct tagOPCITEMATTRIBUTES {
  [string]      LPWSTR      szAccessPath;
  [string]      LPWSTR      szItemID;
                BOOL        bActive;
                OPCHANDLE   hClient;
                OPCHANDLE   hServer;
                DWORD       dwAccessRights;
                DWORD       dwBlobSize;
  [size_is(dwBlobSize)] BYTE    * pBlob;
                VARTYPE     vtRequestedDataType;
                VARTYPE     vtCanonicalDataType;
                OPCEUTYPE   dwEUType;
                VARIANT     vEUInfo;
} OPCITEMATTRIBUTES;
typedef struct tagOPCITEMRESULT {
                OPCHANDLE   hServer;
                VARTYPE     vtCanonicalDataType;
```

```
                WORD        wReserved;
                DWORD       dwAccessRights;
                DWORD       dwBlobSize;
  [size_is(dwBlobSize)]  BYTE   * pBlob;
} OPCITEMRESULT;
//*****************************************************
// OPC Quality flags
//
// Masks for extracting quality subfields
// (note 'status' mask also includes 'Quality' bits)
cpp_quote("#define     OPC_QUALITY_MASK              0xC0")
cpp_quote("#define     OPC_STATUS_MASK               0xFC")
cpp_quote("#define     OPC_LIMIT_MASK                0x03")
// Values for QUALITY_MASK bit field
cpp_quote("#define     OPC_QUALITY_BAD               0x00")
cpp_quote("#define     OPC_QUALITY_UNCERTAIN         0x40")
cpp_quote("#define     OPC_QUALITY_GOOD              0xC0")
// STATUS_MASK Values for Quality = BAD
cpp_quote("#define     OPC_QUALITY_CONFIG_ERROR      0x04")
cpp_quote("#define     OPC_QUALITY_NOT_CONNECTED     0x08")
cpp_quote("#define     OPC_QUALITY_DEVICE_FAILURE    0x0c")
cpp_quote("#define     OPC_QUALITY_SENSOR_FAILURE    0x10")
cpp_quote("#define     OPC_QUALITY_LAST_KNOWN        0x14")
cpp_quote("#define     OPC_QUALITY_COMM_FAILURE      0x18")
cpp_quote("#define     OPC_QUALITY_OUT_OF_SERVICE    0x1C")
// STATUS_MASK Values for Quality = UNCERTAIN
cpp_quote("#define     OPC_QUALITY_LAST_USABLE       0x44")
cpp_quote("#define     OPC_QUALITY_SENSOR_CAL        0x50")
cpp_quote("#define     OPC_QUALITY_EGU_EXCEEDED      0x54")
cpp_quote("#define     OPC_QUALITY_SUB_NORMAL        0x58")
// STATUS_MASK Values for Quality = GOOD
cpp_quote("#define     OPC_QUALITY_LOCAL_OVERRIDE    0xD8")

// Values for Limit Bitfield
cpp_quote("#define     OPC_LIMIT_OK                  0x00")
cpp_quote("#define     OPC_LIMIT_LOW                 0x01")
cpp_quote("#define     OPC_LIMIT_HIGH                0x02")
cpp_quote("#define     OPC_LIMIT_CONST               0x03")
//*****************************************************
//Interface Definitions
```

```
//*****************************************************
[
  object,
  uuid(39c13a4d-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCServer : IUnknown
{
  HRESULT AddGroup(
    [in, string]          LPCWSTR       szName,
    [in]                  BOOL          bActive,
    [in]                  DWORD         dwRequestedUpdateRate,
    [in]                  OPCHANDLE     hClientGroup,
    [unique, in]          LONG          * pTimeBias,
    [unique, in]          FLOAT         * pPercentDeadband,
    [in]                  DWORD         dwLCID,
    [out]                 OPCHANDLE     * phServerGroup,
    [out]                 DWORD         * pRevisedUpdateRate,
    [in]                  REFIID        riid,
    [out, iid_is(riid)]   LPUNKNOWN     * ppUnk
    );
  HRESULT GetErrorString(
    [in]                  HRESULT       dwError,
    [in]                  LCID          dwLocale,
    [out, string]         LPWSTR        * ppString
    );
  HRESULT GetGroupByName(
    [in, string]          LPCWSTR szName,
    [in]                  REFIID riid,
    [out, iid_is(riid)]   LPUNKNOWN * ppUnk
    );
  HRESULT GetStatus(
    [out] OPCSERVERSTATUS ** ppServerStatus
    );
  HRESULT RemoveGroup(
    [in] OPCHANDLE   hServerGroup,
    [in] BOOL        bForce
    );
  HRESULT CreateGroupEnumerator(
    [in] OPCENUMSCOPE   dwScope,
```

```
    [in] REFIID          riid,
    [out, iid_is(riid)]  LPUNKNOWN  * ppUnk
    );
}
//*****************************************************
[
  object,
  uuid(39c13a4e-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCServerPublicGroups : IUnknown
{
  HRESULT GetPublicGroupByName(
    [in, string]         LPCWSTR      szName,
    [in]                 REFIID       riid,
    [out, iid_is(riid)]  LPUNKNOWN  * ppUnk
    );
  HRESULT RemovePublicGroup(
    [in] OPCHANDLE   hServerGroup,
    [in] BOOL        bForce
    );
}
//*****************************************************
[
  object,
  uuid(39c13a4f-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCBrowseServerAddressSpace: IUnknown
{
  HRESULT QueryOrganization(
    [out]          OPCNAMESPACETYPE  * pNameSpaceType
    );
  HRESULT ChangeBrowsePosition(
    [in]           OPCBROWSEDIRECTION dwBrowseDirection,
    [in, string]   LPCWSTR            szString
    );
  HRESULT BrowseOPCItemIDs(
    [in]           OPCBROWSETYPE     dwBrowseFilterType,
    [in, string]   LPCWSTR           szFilterCriteria,
```

```
    [in]            VARTYPE             vtDataTypeFilter,
    [in]            DWORD               dwAccessRightsFilter,
    [out]           LPENUMSTRING        * pplEnumString
    );
  HRESULT GetItemID(
    [in]            LPWSTR              szItemDataID,
    [out, string]   LPWSTR              * szItemID
    );
  HRESULT BrowseAccessPaths(
    [in, string]    LPCWSTR             szItemID,
    [out]           LPENUMSTRING        * pplEnumString
    );
}
//*****************************************************
[
  object,
  uuid(39c13a50-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCGroupStateMgt : IUnknown
{
  HRESULT GetState(
    [out]           DWORD               * pUpdateRate,
    [out]           BOOL                * pActive,
    [out, string]   LPWSTR              * ppName,
    [out]           LONG                * pTimeBias,
    [out]           FLOAT               * pPercentDeadband,
    [out]           DWORD               * pLCID,
    [out]           OPCHANDLE           * phClientGroup,
    [out]           OPCHANDLE           * phServerGroup
    );
  HRESULT SetState(
    [unique, in]    DWORD               * pRequestedUpdateRate,
    [out]           DWORD               * pRevisedUpdateRate,
    [unique, in]    BOOL                * pActive,
    [unique, in]    LONG                * pTimeBias,
    [unique, in]    FLOAT               * pPercentDeadband,
    [unique, in]    DWORD               * pLCID,
    [unique, in]    OPCHANDLE           * phClientGroup
    );
```

```
    HRESULT SetName(
        [in, string] LPCWSTR szName
        );
    HRESULT CloneGroup(
        [in, string]        LPCWSTR         szName,
        [in]                REFIID          riid,
        [out, iid_is(riid)] LPUNKNOWN   * ppUnk
        );
}
//*****************************************************
[
    object,
    uuid(39c13a51-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCPublicGroupStateMgt : IUnknown
{
    HRESULT GetState(
        [out] BOOL * pPublic
        );
    HRESULT MoveToPublic(
        void
        );
}
//*****************************************************
[
    object,
    uuid(39c13a52-011e-11d0-9675-0020afd8adb3),
    pointer_default(unique)
]
interface IOPCSyncIO : IUnknown
{
    HRESULT Read(
        [in]                        OPCDATASOURCE   dwSource,
        [in]                        DWORD           dwCount,
        [in, size_is(dwCount)]      OPCHANDLE       * phServer,
        [out, size_is(,dwCount)]    OPCITEMSTATE    ** ppItemValues,
        [out, size_is(,dwCount)]    HRESULT         ** ppErrors
        );
    HRESULT Write(
```

```
    [in]                        DWORD           dwCount,
    [in, size_is(dwCount)]      OPCHANDLE       * phServer,
    [in, size_is(dwCount)]      VARIANT         * pItemValues,
    [out, size_is(,dwCount)]    HRESULT         ** ppErrors
    );
}
//*****************************************************
[
  object,
  uuid(39c13a53-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCAsyncIO : IUnknown
{
  HRESULT Read(
    [in]                        DWORD           dwConnection,
    [in]                        OPCDATASOURCE   dwSource,
    [in]                        DWORD           dwCount,
    [in, size_is(dwCount)]      OPCHANDLE       * phServer,
    [out]                       DWORD           * pTransactionID,
    [out, size_is(,dwCount)]    HRESULT         ** ppErrors
    );
  HRESULT Write(
    [in]                        DWORD           dwConnection,
    [in]                        DWORD           dwCount,
    [in, size_is(dwCount)]      OPCHANDLE       * phServer,
    [in, size_is(dwCount)]      VARIANT         * pItemValues,
    [out]                       DWORD           * pTransactionID,
    [out, size_is(,dwCount)]    HRESULT         ** ppErrors
    );
  HRESULT Refresh(
    [in]  DWORD                 dwConnection,
    [in]  OPCDATASOURCE         dwSource,
    [out] DWORD                 * pTransactionID
    );
  HRESULT Cancel(
    [in] DWORD dwTransactionID
    );
}
//*****************************************************
```

```
[
  object,
  uuid(39c13a54-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCItemMgt: IUnknown
{
  HRESULT AddItems(
    [in]                          DWORD              dwCount,
    [in, size_is(dwCount)]        OPCITEMDEF         * pItemArray,
    [out, size_is(,dwCount)]      OPCITEMRESULT      ** ppAddResults,
    [out, size_is(,dwCount)]      HRESULT            ** ppErrors
    );
  HRESULT ValidateItems(
    [in]                          DWORD              dwCount,
    [in, size_is(dwCount)]        OPCITEMDEF         * pItemArray,
    [in]                          BOOL               bBlobUpdate,
    [out, size_is(,dwCount)]      OPCITEMRESULT      ** ppValidationResults,
    [out, size_is(,dwCount)]      HRESULT            ** ppErrors
    );
  HRESULT RemoveItems(
    [in]                          DWORD              dwCount,
    [in, size_is(dwCount)]        OPCHANDLE          * phServer,
    [out, size_is(,dwCount)]      HRESULT            ** ppErrors
    );
  HRESULT SetActiveState(
    [in]                          DWORD              dwCount,
    [in, size_is(dwCount)]        OPCHANDLE          * phServer,
    [in]                          BOOL               bActive,
    [out, size_is(,dwCount)]      HRESULT            ** ppErrors
    );
  HRESULT SetClientHandles(
    [in]                          DWORD              dwCount,
    [in, size_is(dwCount)]        OPCHANDLE          * phServer,
    [in, size_is(dwCount)]        OPCHANDLE          * phClient,
    [out, size_is(,dwCount)]      HRESULT            ** ppErrors
    );
  HRESULT SetDatatypes(
    [in]                          DWORD              dwCount,
    [in, size_is(dwCount)]        OPCHANDLE          * phServer,
```

```
    [in, size_is(dwCount)]          VARTYPE          * pRequestedDatatypes,
    [out, size_is(,dwCount)]        HRESULT          ** ppErrors
    );
  HRESULT CreateEnumerator(
    [in]                            REFIID           riid,
    [out, iid_is(riid)]             LPUNKNOWN        * ppUnk
    );
}
//*****************************************************
[
  object,
  uuid(39c13a55-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IEnumOPCItemAttributes : IUnknown
{
  HRESULT Next(
    [in]  ULONG celt,
    [out, size_is(,*pceltFetched)] OPCITEMATTRIBUTES ** ppItemArray,
    [out] ULONG * pceltFetched
    );
  HRESULT Skip(
    [in] ULONG celt
    );
  HRESULT Reset(
    void
    );
  HRESULT Clone(
    [out] IEnumOPCItemAttributes ** ppEnumItemAttributes
    );
}
// Data Access V2.0 additions
[
  object,
  uuid(39c13a70-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
interface IOPCDataCallback : IUnknown
{
HRESULT OnDataChange(
```

```
    [in]                        DWORD          dwTransid,
    [in]                        OPCHANDLE      hGroup,
    [in]                        HRESULT        hrMasterquality,
    [in]                        HRESULT        hrMastererror,
    [in]                        DWORD          dwCount,
    [in, size_is(dwCount)]      OPCHANDLE      * phClientItems,
    [in, size_is(dwCount)]      VARIANT        * pvValues,
    [in, size_is(dwCount)]      WORD           * pwQualities,
    [in, size_is(dwCount)]      FILETIME       * pftTimeStamps,
    [in, size_is(dwCount)]      HRESULT        * pErrors
);
  HRESULT OnReadComplete(
    [in]                        DWORD          dwTransid,
    [in]                        OPCHANDLE      hGroup,
    [in]                        HRESULT        hrMasterquality,
    [in]                        HRESULT        hrMastererror,
    [in]                        DWORD          dwCount,
    [in, size_is(dwCount)]      OPCHANDLE      * phClientItems,
    [in, size_is(dwCount)]      VARIANT        * pvValues,
    [in, size_is(dwCount)]      WORD           * pwQualities,
    [in, size_is(dwCount)]      FILETIME       * pftTimeStamps,
    [in, size_is(dwCount)]      HRESULT        * pErrors
);
  HRESULT OnWriteComplete(
    [in]                        DWORD          dwTransid,
    [in]                        OPCHANDLE      hGroup,
    [in]                        HRESULT        hrMastererr,
    [in]                        DWORD          dwCount,
    [in, size_is(dwCount)]      OPCHANDLE      * pClienthandles,
    [in, size_is(dwCount)]      HRESULT        * pErrors
);
  HRESULT OnCancelComplete(
    [in]                        DWORD          dwTransid,
    [in]                        OPCHANDLE      hGroup
);
}
//*****************************************************
[
  object,
  uuid(39c13a71-011e-11d0-9675-0020afd8adb3),
```

```
  pointer_default(unique)
]
interface IOPCAsyncIO2 : IUnknown
{
  HRESULT Read(
    [in]                    DWORD          dwCount,
    [in, size_is(dwCount)]  OPCHANDLE      * phServer,
    [in]                    DWORD          dwTransactionID,
    [out]                   DWORD          * pdwCancelID,
    [out, size_is(,dwCount)] HRESULT       ** ppErrors
    );
  HRESULT Write(
    [in]                    DWORD          dwCount,
    [in, size_is(dwCount)]  OPCHANDLE      * phServer,
    [in, size_is(dwCount)]  VARIANT        * pItemValues,
    [in]                    DWORD          dwTransactionID,
    [out]                   DWORD          * pdwCancelID,
    [out, size_is(,dwCount)] HRESULT       ** ppErrors
    );
  HRESULT Refresh2(
    [in]                    OPCDATASOURCE  dwSource,
    [in]                    DWORD          dwTransactionID,
    [out]                   DWORD          * pdwCancelID
    );
  HRESULT Cancel2(
    [in]                    DWORD          dwCancelID
    );
  HRESULT SetEnable(
    [in]                    BOOL           bEnable
    );
  HRESULT GetEnable(
    [out]                   BOOL           *pbEnable
    );
}
//*****************************************************
[
  object,
  uuid(39c13a72-011e-11d0-9675-0020afd8adb3),
  pointer_default(unique)
]
```

```
interface IOPCItemProperties : IUnknown
{
  HRESULT QueryAvailableProperties (
    [in]                          LPWSTR      szItemID,
    [out]                         DWORD       * pdwCount,
    [out, size_is(,*pdwCount)]    DWORD       ** ppPropertyIDs,
    [out, size_is(,*pdwCount)]    LPWSTR      ** ppDescriptions,
    [out, size_is(,*pdwCount)]    VARTYPE     ** ppvtDataTypes
);
  HRESULT GetItemProperties (
    [in]                          LPWSTR      szItemID,
    [in]                          DWORD       dwCount,
    [in, size_is(dwCount)]        DWORD       * pdwPropertyIDs,
    [out, size_is(,dwCount)]      VARIANT     ** ppvData,
    [out, size_is(,dwCount)]      HRESULT     ** ppErrors
);
  HRESULT LookupItemIDs(
    [in]                          LPWSTR      szItemID,
    [in]                          DWORD       dwCount,
    [in, size_is(dwCount)]        DWORD       * pdwPropertyIDs,
    [out, string, size_is(,dwCount)]  LPWSTR  ** ppszNewItemIDs,
    [out, size_is(,dwCount)]      HRESULT     ** ppErrors
);
}
[
    uuid(9DB24EAC-C452-477B-8B70-871F51F3330D),
    version(1.0),
    helpstring("OPCDA 1.0 Type Library")
]
library OPCDALib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");
    interface IOPCServer ;
    interface IOPCServerPublicGroups ;
    interface IOPCBrowseServerAddressSpace;
    interface IOPCGroupStateMgt ;
    interface IOPCPublicGroupStateMgt ;
    interface IOPCSyncIO ;
    interface IOPCAsyncIO ;
```

```
interface IOPCItemMgt;
interface IEnumOPCItemAttributes ;
interface IOPCDataCallback ;
interface IOPCAsyncIO2 ;
interface IOPCItemProperties ;
[
    uuid(7C13259A-74FD-4064-818F-C639E4B5811B),
    helpstring("TestServer Class")
]
coclass TestServer
{
    [default] interface IOPCServer;
};
```

附录 II　基于 VB 的 OPC 客户端同步应用程序

```
Option Explicit
Option Base 1
Dim WithEvents ServerObj As OPCServer          //对象定义
Dim WithEvents GroupObj As OPCGroup
Dim ItemObj As OPCItem
Private Sub Command_Start_Click()
    Dim OutText As String
    On Error GoTo ErrorHandler
    Command_Start.Enabled = False
    Command_Read.Enabled = True
    Command_Write.Enabled = True
    Command_Exit.Enabled = True
    OutText = "连接 OPC 服务器"
    Set ServerObj = New OPCServer
    ServerObj.Connect ("Matrikon.OPC.Simulation")
    OutText = "添加组"
    Set GroupObj = ServerObj.OPCGroups.Add("MyOPCGroup")
    OutText = "为组添加 Item"
    Set ItemObj = GroupObj.OPCItems.AddItem("Random.Real4", 1)
    Exit Sub
ErrorHandler:
    MsgBox Err.Description + Chr(13) + _
        OutText, vbCritical, "ERROR"
End Sub
Private Sub Command_Read_Click()
    Dim OutText As String
    Dim myValue As Variant
    Dim myQuality As Variant
    Dim myTimeStamp As Variant
    On Error GoTo ErrorHandler
    OutText = "读 ITEM 值"
    ItemObj.Read OPCDevice, myValue, myQuality, myTimeStamp
    Edit_ReadVal = myValue
    Edit_ReadQu = GetQualityText(myQuality)
    Edit_ReadTS = myTimeStamp
    Exit Sub
```

```
ErrorHandler:
    MsgBox Err.Description + Chr(13) + _
        OutText, vbCritical, "ERROR"
End Sub
Private Sub Command_Write_Click()
    Dim OutText As String
    Dim Serverhandles(1) As Long
    Dim MyValues(1) As Variant
    Dim MyErrors() As Long
    OutText = "写值"
    On Error GoTo ErrorHandler
    Serverhandles(1) = ItemObj.ServerHandle
    MyValues(1) = Edit_WriteVal
    GroupObj.SyncWrite 1, Serverhandles, MyValues, MyErrors
    Edit_WriteRes = ServerObj.GetErrorString(MyErrors(1))
    Exit Sub
ErrorHandler:
    MsgBox Err.Description + Chr(13) + _
        OutText, vbCritical, "ERROR"
End Sub
Private Sub Command_Exit_Click()
    Dim OutText As String
    On Error GoTo ErrorHandler
    Command_Start.Enabled = True
    Command_Read.Enabled = False
    Command_Write.Enabled = False
    Command_Exit.Enabled = False
    OutText = "删除对象"
    Set ItemObj = Nothing
    ServerObj.OPCGroups.RemoveAll
    Set GroupObj = Nothing
    ServerObj.Disconnect
    Set ServerObj = Nothing
    Exit Sub
ErrorHandler:
    MsgBox Err.Description + Chr(13) + _
        OutText, vbCritical, "ERROR"
End Sub
Private Function GetQualityText(Quality) As String
    Select Case Quality
```

```
        Case 0:      GetQualityText = "BAD"
        Case 64:     GetQualityText = "UNCERTAIN"
        Case 192:    GetQualityText = "GOOD"
        Case 8:      GetQualityText = "NOT_CONNECTED"
        Case 13:     GetQualityText = "DEVICE_FAILURE"
        Case 16:     GetQualityText = "SENSOR_FAILURE"
        Case 20:     GetQualityText = "LAST_KNOWN"
        Case 24:     GetQualityText = "COMM_FAILURE"
        Case 28:     GetQualityText = "OUT_OF_SERVICE"
        Case 132:    GetQualityText = "LAST_USABLE"
        Case 144:    GetQualityText = "SENSOR_CAL"
        Case 148:    GetQualityText = "EGU_EXCEEDED"
        Case 152:    GetQualityText = "SUB_NORMAL"
        Case 216:    GetQualityText = "LOCAL_OVERRIDE"
        Case Else:   GetQualityText = "UNKNOWN ERROR"
    End Select
End Function
```