国内图书分类号：TP311.5　　学校代码：10079
国际图书分类号：004.41　　密级：公开

工学硕士学位论文

OPC UA客户端访问与测试功能研究及开发

硕士研究生：李金亮
导　　师：陆会明 教授
申请学位：工学硕士
学　　科：控制科学与工程
专　　业：模式识别与智能系统
所在学院：控制与计算机工程学院
答辩日期：2011年3月
授予学位单位：华北电力大学

Classified Index: TP311.5
U.D.C: 004.41

Dissertation for the Master's Degree in Engineering

Research and Development for the OPC UA Client's Access and Test Function

Candidate: LiJinliang

Supervisor: LuHuiming

Academic Degree Applied for: Master of Engineering

Speciality: pattern recognition and intelligent systems

School: School of North China Electric Power University

Date of Defence: March, 2011

Degree-Conferring-Institution: North China Electric Power University

华北电力大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《OPC UA 客户端访问与测试功能研究及开发》，是本人在导师指导下，在华北电力大学攻读硕士学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签名：李国亮　　日期：2011 年 03月 11日

华北电力大学硕士学位论文使用授权书

《OPC UA 客户端访问与测试功能研究及开发》系本人在华北电力大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归华北电力大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解华北电力大学关于保存、使用学位论文的规定，同意学校保留并向有关部门送交论文的复印件和电子版本，允许论文被查阅和借阅。本人授权华北电力大学，可以采用影印、缩印或其他复制手段保存论文，可以公布论文的全部或部分内容。

本学位论文属于（请在以上相应方框内打“√”）：

保密□，在　　年解密后适用本授权书

不保密☑

作者签名：李国亮　　日期：2011 年03月 11日

导师签名：[illegible]　　日期：2011 年03 月 11日

摘 要

OPC Unified Architecture 采用集成的地址空间与更加安全的传输机制突破了传统 OPC 的限制，使得基于互联网跨越不同平台的数据传输更加安全可靠。基于 OPC UA 规范的客户端软件的研究与开发对于服务器软件的测试具有很重要的作用。对 OPC UA 规范中客户端的架构与功能进行研究，给出应用程序的结构，在.NET 平台下使用 C#语言开发出了客户端。对客户端与服务器查找、浏览、读/写、订阅的交互及相应的实现代码进行叙述，测试了查找服务器访问地址空间等客户端的功能，结果表明客户端能有效与服务器交互完成服务器软件测试任务。

关键词：OPC 客户端；OPC 统一架构；OPC UA 地址空间；.NET；

Abstract

The OPC Unified Architecture using the integrated address space and more secure transport mechanism break through the limitations of traditional OPC, which make the Internet-based data communication across different platforms more secure. The research and development of Client application based on OPC UA specification for the test of server application is necessary. This paper study client in structure and function for application with OPC UA specification, and develop a client application in the .NET platform structure using c# language. It is describe that the client browse, read and write, subscription and the realization of the corresponding code, and test the client function such as find servers, The result indicate that the client can communicate with the server completed the server application testing tasks.

Keywords: OPC UA Client; OPC Unified Architecture; OPC address space; .NET

目 录

第1章 引言

1.1 OPC UA 技术及其发展现状

随着自动化技术和信息技术的快速发展，工业自动化控制系统也获得了长足的发展。现在大多数分布式系统采用客户/服务器结构[1]，由于没有设备接口的通用标准，设备厂商按照自己的标准接口开发应用程序，从系统的开放性角度看有很多局限性。如何开发标准的自动化软件访问处于不同总线系统和协议中的设备数据是工业控制现场中主要研究任务[2]。另外，工业现场智能设备的应用，如何将工厂底层设备信息采集到过程控制管理层到经营决策层，完成各个层次间现场信息的交互是工业控制现场的自动化的基础。数据交互在控制现场亟待优化的环境下，OPC 技术的出现解决了一部分问题，但是由于 OPC 技术存在技术上的局限性不能满足逐渐发展的工业现场控制要求，OPC 基金会于 2006 年提出 OPC UA 这是新一代理想数据交换技术[3]。

1.1.1 传统 OPC 及其限制

OPC(OLE for Process Control)技术在当今过程控制领域，是一种非常流行的数据交换技术。它是 OPC 基金会在微软 Windows 的 OLE 技术基础上发展的工业自动化数据交换标准，采用组件对象模型/分布式组件对象模型（COM/DCOM）技术，为工业控制系统提供了一种标准的数据访问机制，将现场信号按照统一的标准与监控软件、商业管理软件无缝连接起来，同时将硬件和软件有效地分离开[4]。只要硬件开发商提供带有 OPC 接口的服务器，任何支持 OPC 规范的客户程序均可按照统一的方式对不同硬件供应商的设备进行存取，无须重复开发通信驱动程序。

最初提出 OPC 技术是为了简化传统的 I/O 驱动开发方式。近些年来，OPC 技术的发展超出了仅作为驱动程序的初衷，为企业内部的信息交换提供了一个开放的平台。OPC 技术的出现改变了工业控制现场的系统结构[5]，图 1-1 是传统的过程控制系统结构图,上位机中的监控软件及经营决策层管理软件在访问底层硬件设备数据时需要开发专用的通信驱动程序。基于 OPC 技术过程控制系统结构如图 1-1，在客户与服务器交互的结构中以统一的方式访问现场数据。只要遵循 OPC 接口的约束，双方的数据交互就是透明的。硬件供应商提供符合 OPC 标准的服务器，软件开发商没有必要了解硬件的底层信息。OPC 就是通过这样类似于“软件总线”的

技术，解决了硬件和软件之间数据交换的问题。改变的系统结构对于终端用户来讲意味着系统更加开放，只要用户选择了符合 OPC 规范客户端，就可以任意访问支持 OPC 技术的设备和软件，集成已经存在的不同系统将变得非常容易[6]。

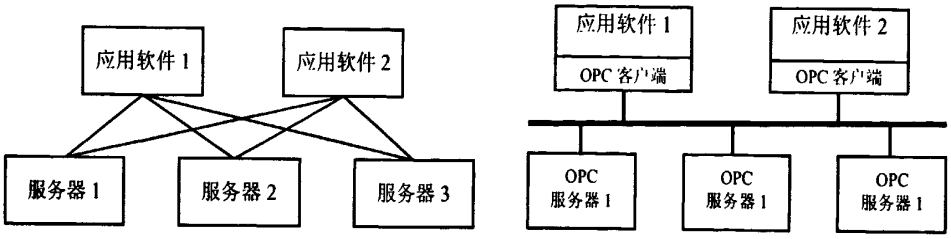

图 1-1 传统过程与基于 OPC 过程控制系统结构

依据工业控制软件中不同要求，OPC 基金会发布了几个不同 OPC 规范包括：OPC DA(Data Access)数据访问、OPC A&E （Alarm&Event）报警与事件、OPC HDA(Historical Data Access)历史数据访问。每种规范对应着一种类型的服务器和客户端程序的开发，区别是它们实现功能的侧重点不同。实时数据交换是工业控制的必要条件，OPC 技术为工业控制现场实时数据交换提供了一种高效的手段。基于 OPC 接口的生产过程控制与调度系统中的 SCADA 软件得到了广泛的应用。通过 OPC 接口上层 DCS 或 SCADA 可以实时采集底层应用程序的数据，也可以通过接口将 DCS 或 SCADA 系统中的数据采集到上位机中的应用程序中。OPC 服务器通过 OPC 接口提供设备的过程数据，OPC 客户端连接到 OPC 服务器后可以访问数据，这种客户/服务器方式在自动化行业得到了广泛的应用。OPC 技术对于推动工业自动化异构系统集成起到极大的作用。

传统 OPC 软件接口基于 Windows 的 COM/DCOM，所以可以集合到微软的操作系统中，COM 和 DCOM 提供进程内通信功能并可以使用 DCOM 跨越计算机通信。在微软操作系统的环境下，OPC 客户端（COM 客户端）可以同 OPC 服务器（COM 服务器）进行信息交互。但是，由于 COM 技术对 Microsoft 平台的依赖性，基于 COM 的 OPC 接口很难被应用到其他系统例如：LINUX 、UNIX 等系统平台中，造成应用程序的移植性不好[7]。另外，DCOM 在网络通信方面存在缺陷。默认情况下，在运行时 DCOM 为每个通信连接动态分配 TCP 或 UDP 端口，而绝大多数防火墙是不允许这种行为的，这使得 DCOM 的远程连接无法通过 Internet 上的防火墙，所以使用 COM 和 DCOM 的 OPC 服务器与客户端只能运行于 Windows 系统和本地网络中，随着控制要求的提高无法满足软件开放性的要求。虽然，OPC 基金会为了实现跨平台并在 Internet 上工作要求发布了 XML-DA 规

范，但是包含相同信息的 XML 报文比 DCOM 报文大很多[7]，在工业控制现场对实时性要求高的苛刻环境下，传输速度无法满足要求。另外，对于跨网络数据通信的安全性方面，也不能满足要求，因此 XML-DA 规范的出现并没有从根本上解决传统 OPC 的限制。除了平台的独立性，OPC 会员更提出对复杂数据与系统的描述要求移除传统 OPC 技术的限制。此外，在客户端开发方面，原有 OPC 规范的 API 是独立的，分别满足不同的要求而独立开发。这种分离虽然简化了 OPC 服务器的开发，却把数据集成的重担交给了通过 OPC 服务器采集数据的客户端软件，增加了客户端开发的工作量[7]。

1.1.2 OPC UA 技术的提出及其优势

为了解决传统 OPC 技术平台独立性、安全机制不足、使用和实现过于复杂等限制并满足系统发展使用要求，OPC 基金会于 2006 年发布了 OPC UA（OPC Unified Architecture）这是新一代组件和系统之间统一数据交互标准，用来取代传统 OPC 技术。OPC UA 规范摒弃了传统 OPC 规范中的限制，有下面几个特点：

（1） 集成传统 OPC DA、A&E、HDA 的功能[8]与信息使用通用接口向外提供信息。

（2） 进程内或网络通信使用开放的独立于平台的协议。

（3） 跨越防火墙的因特网访问与通信。

（4） 协议层与应用程序级集成的访问与安全机制。

（5） 扩展的面向对象模型，对象可以包含变量和方法并可以激发事件。

（6） 传输机制和模型化能力形成其他标准的基础

（7） 可扩展性，使用范围从嵌入式系统到企业资源管理软件。

从上述的几点可以看出 OPC UA 彻底的摒弃了过时的 COM/DCOM 技术，使 OPC UA 不再局限于 Windows 系统，而是跨平台的技术标准，面向未来的网络计算时代，是以 SOA、WebService 为核心的理想的交换技术。OPC UA 技术的发展也改变了传统 OPC(OLE for Process Control)，将其重新定义为 Openness（开放性）、Productivity（生产力）、Collaboration（协作性）[7]。

OPC UA 技术使用消息传递的机制，解决了传统 OPC 在跨平台方面的限制，OPC UA 可以在 Linux 和 Unix 平台中使用。客户端发送服务请求，服务器响应请求并返回客户端请求的结果。OPC UA 通过通信栈来传输消息[8]，通信栈中的传输协议可以采用 SOAP/HTTP 和 TCP/IP 两种方式，传输协议都是与平台独立的，并且能跨越防火墙。OPC UA 使用面向服务的体系结构（SOA）实现了跨 Internet 的

数据交换系统，为构建跨地域的数据采集系统提供了有力的技术支撑。现有 OPC XML-DA 技术虽然实现了在互联网上的数据通信，相同时间内读取的数据量相对于基于 COM/DCOM 的数据量少两个数量级左右。OPC UA 全面超越了 OPC XML-DA，OPC UA 的通信机制包含了 DCOM、Web Services、.NET Remoting、MSMQ、ASMX、WSE 等的优势，能够提供安全的，可靠的通信，保证了性能和良好的可互操作性。另外，OPC UA 技术增强了 OPC UA 技术的可扩展性，使得基于 OPC UA 标准的产品可以更好的实现工厂级的数据采集和管理，从 PLC、DCS 等现场设备、到 HMI、SCADA、再到生产制造系统再到企业上层管理软件都可以通过 OPC UA 标准，统一进行沟通。如图 1-2 所示。

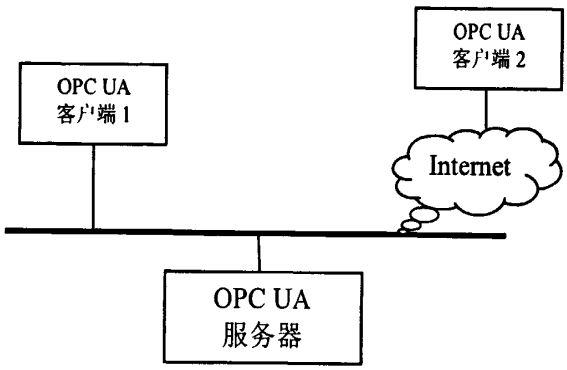

图 1-2 统一访问方式

OPC UA 服务器为客户端提供的对象和相关信息都是与服务器的地址空间有关的，提供一个一致的、完整的地址空间和服务模型。这就允许一个单一的 OPC UA 服务器把数据、报警与事件和历史信息统一到它的地址空间里，并且可以用一套统一的服务为它们向外提供接口，这些服务也包括一个统一的安全模型。传统 OPC 分别定义了 OPC DA、OPC A&E、OPC HDA 规范，它们的地址空间都是彼此完全独立的。通过不同的 API（Application Programming Interface）访问的是不同的地址空间，增加了客户端开发的复杂性。OPC UA 服务器将各自独立的地址空间集成到一个唯一的地址空间中，客户端访问实时数据、历史数据、报警信息等都是访问的一个集成的地址空间，同时实现数据存取、报警与事件、历史数据存取、控制命令、复杂数据的交互通信。

从企业网络和自动化网络及底层应用程序通信，安全性成为首要考虑的问题，任何安全性方面的疏漏都可能造成企业巨大损失。传统 OPC 规范中没有安全性的设计，其安全性完全依赖于 COM/DCOM 的安全性。OPC UA 定义了自己安全模型

规范，其中使用一套完善的安全机制,实现多层安全架构。OPC UA 安全性主要考虑：客户端和服务器的合法性，用户的合法性，客户端和服务器之间通信的一致性和机密性。首先客户与服务器的连接需要建立安全信道，安全信道中绑定了使用的安全协议，建立安全信道使用了证书认证、用户名/密码、WS 安全标识等安全措施，信息交换的过程中使用绑定的安全协议等加密技术保证信息传输的安全[9]。新的安全模型保证了数据从原始设备到上位机软件，从本地机到远程系统通信的可靠性。数据传输的可靠性方面，OPC UA 规范中提出高度可靠性和冗余性的设计[10]，可调试的逾时设置，错误发现和自动纠正等新特性，都使得符合 OPC UA 规范的软件产品可以很自如的处理通信错误和失败。OPC UA 的标准冗余模型也使得来自不同厂商的软件可以同时被采纳并兼容。OPC UA 规范的出现是新时代自动化系统的需要，是 Internet 的重要性在自动化系统中越来越加强的产物，简化易用的数据采集和交换技术，是网络计算 SOA、WebService 等新技术在工厂信息化建设中应用的必然，符合未来自动化系统和信息系统发展的趋势。

1.1.3 国内外研究现状

目前，OPC 基金会的公司会员已经拥有 440 余位，3500 多家致力于开发 OPC 产品的公司超过 22000 中产品,它是由一些世界领先地位的自动化系统和硬件、软件公司与微软（Microsoft）紧密合作而建立的。目前，大多数仪表及控制厂商都支持该标准。迄今为止，OPC 基金会发布的规范共有十三个部分，OPC 基金会网站上也再不断更新规范，其中有的规范还出于草案状态，尚未形成一套最终的规范，所以 OPC UA 的产品研究还是处于发展的阶段。

OPC 简化了系统的结构，使系统具有更长的寿命、更低的价格，使得解决工业控制问题成为可能，所以现在的控制系统还是使用的传统 OPC。OPC UA 是超过 30 多个 OPC 厂商共同开发的，开放、跨平台的体系简历在 OPC 基金会成功的接口基础之上。从 2006 年 OPC 基金会发布 OPC UA 规范以来，经过几年的发展已经取得了很大的进步。作为 OPC UA 先驱厂商的一员，Beckhoff 已经于 2007 年底完成了 OPC UA 产品测试，陆续的其他自动化厂商也初步实现了 OPC UA 客户端与服务器的开发。MatrikonOPC 是 OPC 技术上一直相当领先的供应商，围绕 OPC UA，MatrikonOPC 也研发推出了新的产品。2010 年更出现了 OPC UA 技术应用于现场的案例。2010 年 5 月，德国“阿尔法·文图斯”的海上风电厂并网发电，其 SCADA 系统中使用了 OPC UA 技术。国内对于 OPC UA 的研究主要集中在几大自动化软件商，三维力控在产品中已经考虑到 OPC UA 标准、九思易等许多公司为

了在下一代产品中兼容 OPC UA、北京亚控也一直致力于 OPC 的开发，虽然 OPC UA 还没有成熟的产品面世，但是 OPC UA 技术顺应了自动化行业的发展趋势，采用 OPC UA 将是组态软件发展的必然。

1.2 本论文的主要研究内容

论文中描述了开发所使用的 DOT NET 平台与开发语言 C#，通过 OPC UA 技术原理，结合 OPC UA 客户端的要求，在此基础上介绍 OPC UA 客户端程序的实现过程，OPC UA 采用客户/服务器模型，使用消息传递机制，本文主要研究在 OPC UA 规范下实现符合规范的具有能访问与测试 OPC UA 服务器功能的客户端。对于客户端的研究必然要涉及到读取服务器内容，文中也对服务器地址空间中数据组织结构做简单的介绍。

本文完成的内容主要有下面几部分组成：

(1) 简要介绍了程序的开发平台.NET，其中主要是.NET 平台下的 Web Service 与安全对 OPC UA 的影响。

(2) 介绍开发 OPC UA 客户端所设计到的规范，详细剖析了服务 Part4 部分涉及的 OPC UA 客户端所要实现的功能要求。

(3) 描述了使用 C#语言在 visual studio 2008 开发环境下，具体的开发过程并使用测试服务器对开发完成的客户端做测试，测试客户端各项功能。

第 2 章 OPC UA 客户端开发平台 .NET

随着网络技术的进一步发展，系统规模越来越大，如何开发效率更高的新型软件是组态软件研究的重要研究课题。“以网络取代计算机”观念的提出是软件开发跨越性的发展，.NET 是网络时代所需要的新一代计算平台，它改变了传统的计算机计算模式，取而代之以网络计算模式，突破了“软件运行于计算机”的概念[11]，将软件的运行革命性的扩展到网络范围。OPC UA 开发的理念是跨平台与统一的架构系统，这与.NET 平台面向服务的思想体系、面向网络实现大型和复杂系统应用的思想契合。例如 OPC UA 规范中跨平台使用 XML 编码传输正是利用了.NET 平台使用基于 XML 的网络协议的思想，XML Web service 允许应用程序通过 Internet 进行通讯和共享数据，而不管所采用的是哪种操作系统、设备或编程语言。Microsoft .NET 平台提供创建 XML Web service 并将这些服务集成在一起提供无缝的编程体验。OPC UA 基于 SOA、Web Service[12]是一种新的理想的数据采集技术，可以说 OPC UA 是.NET 作为操作平台的思想在数据采集、交互和共享方面的一种体现，.NET Framework 实现了跨技术边界的无缝通信，并且能支持各种业务流程。在.NET 平台上开发和实现 OPC UA 程序就更为简单。

2.1 Microsoft .NET 平台概述

Microsoft .NET 是微软耗巨资多年悉心研究于 2002 年全球同步推出的新一代网络操作平台。早期的.NET 作为用户选择安装的选件外挂在 Windows（98，2000，XP）系统，从 Windows Vista 开始，在 Windows7 和未来的 Windows 系统中，.NET 系统将内置于 Windows 系统之中，成为 Windows 操作系统的重要构成部分。.NET 是网络时代所需要的新一代计算平台，它改变了传统的计算机计算模式，取而代之以网络计算模式。.NET 技术的核心就是以“网络计算”取代“计算机计算”的概念，将软件的运行革命性的扩展到网络范围，可以说真正的.NET 时代的软件是运行于“计算机网络”的[11]。.NET 功能强大而统一，易于构建和运行具有全新客户体验的各种各样的功能的强大应用。.NET 被构建于以下因特网标准之上：HTTP 因特网应用程序之间的通信协议。XML 扩展标记语言用于在因特网应用程序之间交换数据的格式。SOAP 用于请求 Web Service 的标准格式。UDDI

用于搜索和发现 Web Service 的标准。SOAP 简单对象访问协议，是一种轻量级的独立于平台和语言的通信协议，它允许程序经由标准的因特网 HTTP 进行通信。

.NET Framework 是支持生成和运行下一代应用程序和 XML Web Services 的内部 Windows 组件[11]。.NET Framework 旨在实现下列目标：

(1)提供一个一致的面向对象的编程环境，而无论对象代码是在本地存储和执行还是在本地执行但在 Internet 上分布，或者是在远程执行的。

(2)提供一个将软件部署和版本控制冲突最小化的代码执行环境。

(3)提供一个可消除脚本环境或解释环境的性能问题的代码执行环境。

(4)使开发人员的经验在面对类型大不相同的应用程序（如基于 Windows 的应用程序和基于 Web 的应用程序）时保持一致。

(5)按照工业标准生成所有通信，以确保基于 .NET Framework 的代码可与任何其他代码集成。

(6)提供一个可提高代码（包括由未知的或不完全受信任的第三方创建的代码）执行安全性的代码执行环境。

基于.NET 平台开发的应用软件，具有许多基于 Windows API 的软件所不具有的优越性主要体现在以下方面：网络化的计算平台、跨平台趋势、稳定、可靠、安全、继承与扩展 。.NET 平台下一个主要的技术为 Web Services，它秉承“软件就是服务”的真言，同时顺应分布式计算模式的潮流，组件模式、小巧、单一、对于开发人员来讲，开发成本较低。Web services 不是微软发明的，同样也不属于微软专有。Web Service 是一个开放的标准，和 HTTP、XML、SOAP 一样。

Web Service 远程调用服务过程中使用 XML 描述传递参数和返回结果[13]。XML 的使用可以定义无穷无尽的标记来描述文件中的任何数据元素，从而突破 HTML 固定标记集合的约束，使文件的内容更丰富更复杂并组成一个完整的信息体系。良好的数据存储格式、可扩展性、高度结构化、便于网络传输是 XML 的特点。Web Service 提供一个由通用 XML 语法定义的公共接口，这个接口描述了客户可以使用的所有方法及其签名。接口的定义由 Web Service 描述语言（WSDL）完成。建立 Web Service 后，必须提供各种协议以调用接口中的服务，目前 Web Service 通常使用 SOAP 作为调用协议。SOAP 有助于实现大量异构程序和平台之间的互操作性，从而使应用程序被广泛的用户访问。SOAP 本身只是调用信息的包装，它可以通过多种方式进行传输，包括:HTTP、SMTP 和 FTP，SOAP 容易穿过防火墙，极大地方便 Internet 上的传输。所有的这些技术都通过 W3C 进行标准化，因此建立在这些技术之上的 Web Service 完全解决了平台支持问题。

.NET Framework 具有两个主要组件[14]：公共语言运行库和.NET Framework 类库。公共语言运行库是.NET Framework 的基础。可以将运行库看作一个在执行时管理代码的代理，它提供内存管理、线程管理和远程处理等核心服务，并且还强制实施严格的类型安全以及可提高安全性和可靠性的其他形式的代码准确性。事实上，代码管理的概念是运行库的基本原则。以运行库为目标的代码为托管代码，而不以运行库为目标的代码为非托管代码。.NET Framework 的另一个主要组件是类库，它是一个综合性的面向对象的可重用类型集合，可以开发多种应用程序，这些应用程序的传统的命令行或图形用户界面应用程序，包括基于 ASP.NET 所提供的最新创新的应用程序。.NET Framework 可由非托管组件承载，这些组件将公共语言运行库加载到它们的进程中并启动托管代码的执行，从而创建一个可以同时利用托管和非托管功能的软件环境。.NET Framework 不但提供若干个运行库宿主，而且还支持第三方运行库宿主的开发。

2.1.1 公共语言运行库

公共语言运行库是执行系统的一个重要部分，在.NET 系统中有很多用途，例如：管理内存、线程执行、代码执行、代码安全验证、编译以及其他系统服务[15]。这些功能是在公共语言运行库上运行的托管代码所固有的。至于安全性，取决于包括托管组件的来源（如 Internet、企业网络或本地计算机）在内的一些因素，托管组件被赋予不同程度的信任。这意味着即使用在同一活动应用程序中，托管组件既可能能够执行文件访问操作、注册表访问操作或其他须小心使用的功能，也可能不能够执行这些功能。运行库强制实施代码访问安全。例如，用户可以相信嵌入在网页中的可执行文件能够在屏幕上播放动画或唱歌，但不能访问他们的个人数据、文件系统或网络。这样，运行库的安全性功能就使通过 Internet 部署的合法软件能够具有特别丰富的功能。运行库还通过实现称为通用类型系统 (CTS) 的严格类型验证和代码验证基础结构来加强代码可靠性。CTS 确保所有托管代码都是可以自我描述的[15]。各种 Microsoft 和第三方语言编译器生成符合 CTS 的托管代码。这意味着托管代码可在严格实施类型保真和类型安全的同时使用其他托管类型和实例。

此外，运行库的托管环境还消除了许多常见的软件问题。例如，运行库自动处理对象布局并管理对对象的引用，在不再使用它们时将它们释放。这种自动内存管理解决了两个最常见的应用程序错误：内存泄漏和无效内存引用。运行库还提高了开发人员的工作效率。例如，程序员可以用他们选择的开发语言编写应用程序，却仍能充分利用其他开发人员用其他语言编写的运行库、类库和组件。任何选择以运

行库为目标的编译器供应商都可以这样做。以 .NET Framework 为目标的语言编译器使得用该语言编写的现有代码可以使用 .NET Framework 的功能，这大大减轻了现有应用程序的迁移过程的工作负担。

尽管运行库是为未来的软件设计的，但是它也支持现在和以前的软件。托管和非托管代码之间的互操作性使开发人员能够继续使用所需的 COM 组件和 DLL。运行库旨在增强性能。尽管公共语言运行库提供许多标准运行库服务，但是它从不解释托管代码。一种称为实时 (JIT) 编译的功能使所有托管代码能够以它在其上执行的系统的本机语言运行。同时，内存管理器排除了出现零碎内存的可能性，并增大了内存引用区域以进一步提高性能。

2.1.2 .NET Framework 类库

.NET Framework 类库是生成.NET 应用程序、组件和控件的基础。类库由命名空间组成，每个命名空间都包含可在程序中使用的类型：类、结构、枚举、委托和接口，可加快和优化开发过程。.NET Framework 包括的类型可执行下列功能：表示基础数据类型和异常、封装数据结构、执行 I/O、访问关于加载类型的信息，调用.NET Framework 安全检查，提供数据访问、多客户端 GUI 和服务器控制的客户端 GUI。

.NET Framework 类库是一个与公共语言运行库紧密集成的可重用的类型集合。该类库是面向对象的，并提供托管代码可从中导出功能的类型。这不但使 .NET Framework 类型易于使用，而且还减少了学习 .NET Framework 的新功能所需要的时间。此外，第三方组件可与 .NET Framework 中的类无缝集成。.NET Framework 提供了一组丰富的接口以及抽象类和具体（非抽象）类。可以按原样使用这些具体的类，或者在多数情况下从这些类派生其他的类。若要使用接口的功能，既可以创建实现接口的类，也可以从某个实现接口的 .NET Framework 类中派生类。例如，.NET Framework 集合类实现一组可用于开发集合类的接口。您最为

及提供元数据为语言互操作性提供了必要的基础。通用类型系统定义了如何在运行库中声明、使用和管理类型，同时也是运行库支持跨语言集成的一个重要组成部分。因为所有面向运行库的语言都遵循通用类型系统规则来定义和使用类型，类型的用法在各种语言之间是一致的。元数据通过定义统一的存储和检索类型信息的机制使语言互操作性成为可能。编译器将类型信息存储为元数据，公共语言运行库使用该信息在执行过程中提供服务，因为所有类型信息都以相同的方式存储和检索，而与编写该代码的语言无关，所以运行库可以管理多语言应用程序的执行。所有高级语言都编译为 IL。一旦编译完成，并为对象产生了元数据，从其他语言就能够很容易的访问。这意味着用一种语言所编写的对象能够被另一种语言继承。在 C# 中继承用 VB 编写的一个类是可能的。到目前为止，.NET 架构大约支持 C++、Visual Basic、C#、Perl、Jscript 等[17]。

即便运行库向所有托管代码提供在多语言环境中执行的支持，仍无法保证创建的类型的功能可以被其他编程语言完全利用。因为面向运行库的每种语言编译器都使用类型系统和元数据来支持其自己独有的一组语言功能。在不了解调用代码将用何种语言编写的情况下，不太可能知道调用方是否可以访问组件公开的功能。例如：如果选择的语言支持无符号整数，使用 UIINT32 类型的参数设计方法；但是在不识别无符号整数的语言中，该方法可能无法使用。为了确保使用任何编程语言的开发程序访问托管代码，.NET Framework 提供了公共语言规范（CLS），它描述了一组基本的语言功能并定义如何使用这些功能的规则。

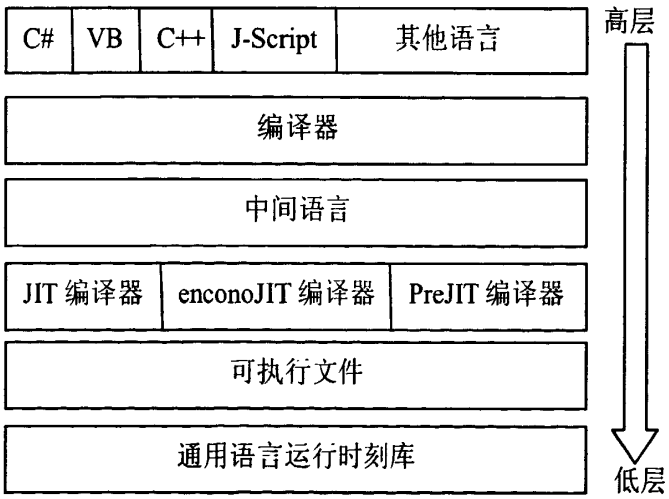

图 2-1 .NET 编译器

通用语言运行时刻库为组件与计算机或机器本地交互提供所有服务。注意到安

装 CLR 的机器并没有标识为一台 PC，并且操作系统也不必是 DOS 或 Windows。Microsoft 有一种明确的策略，将.NET 构架部署到多种平台上，从最小的装置、手持设备、电话和 PDA 直到 UNIX 和 Linux 系统。.NET 提供的可移植性将最终为程序员提供一种开发一次，能够到处使用的程序设计环境。.NET 架构采用 COM 的所有优点。对于远程访问基于.NET 对象的系统来说，这些对象看起来只是象 COM 对象。由于 Web Service 使用基于 XML 的协议和其他系统进行通信，调用网络服务(Web service)的应用程序会始终使用 XML 来传送其请求，并获得 XML 返回的应答，因此这些应用程序不会关心运行于其他计算机上的是何种操作系统或编程语言，实现应用程序的互操作性。

2.3 .NET 安全支持

在软件环境中应用程序的来源很多，它们执行很多任务，。应用程序安全问题非常重要，作为客户端，不想让不确定的程序随意在系统上运行和删除有价值的数据。作为一个服务提供者，不想使用户的错误或攻击能够使整个系统被击垮。CLR 通过用户和代码识别，并结合权限检查来管理安全。例如：对于发布者和代码源，能够了解代码识别，因此能相应授予该资源的使用权限。这种类型的安全称为基于证据的安全，这个.NET 的一个主要特点。.NET 架构还使用 Windows NT 账户和组提供基于角色的安全支持。

应用程序的成功的安全解决方案必须能强化两个安全模型间的平衡。它必须提供对资源的访问，以便以完成有用的工作，它需要对应用程序的安全性作细致的控制以确保代码被识别，检测，并给予合适的安全级别。.NET Framework 就提供了一个这样的安全模型。Microsoft .NET Framework 通过提供相应的支持使组件能够确定授权用户执行什么操作。这些安全性机制使用一个简单而一贯的模型，从而使熟悉代码访问安全性的开发人员可以轻松使用基于角色的安全性，熟悉基于角色的安全性的开发人员也可以轻松使用代码访问安全性。代码访问安全性和基于角色的安全性都是用公共语言运行库提供的一个通用结构实现的。另外，安全策略是一组可配置的规则，公共语言运行库在决定允许代码执行的操作时遵循此规则。安全策略由管理员设置，并由运行库强制。运行库帮助确保代码只能访问或调用安全策略允许的资源或代码。每当发生加载程序集的尝试时，运行库就使用安全策略确定授予程序集的权限。在检查了描述程序集标识的信息（称为证据）后，运行库使用安全策略决定代码的信任程度和由此授予程序集的权限。证据包括但不仅限于代码的

出版商、它的站点以及它的区域。.NET 使用运行库支持、证据、代码识别、权限、代码访问安全性、基于角色的安全性完成对应用程序安全性支持。

2.4 本章小结

本章首先介绍了.NET 平台的基础内容，描述.NET 框架中的公共语言运行库和.NET Framework 类库两部分主要内容，.NET 框架有助于简化服务和应用程序的开发。对于.NET 平台中互操作性与安全方面的描述，是 OPC UA 应用程序跨平台与应用程序安全模型的基础。

第 3 章 OPC UA 规范分析与关键技术

OPC 技术经过多年的发展取得显著的进步，从传统 OPC 技术到 OPC XML，到现在 OPC UA 技术研究的具有跨平台等特点逐步解决了在工业现场中应用的局限性。OPC 基金会于 2006 年发布了 OPC 统一架构规范，规范的发布相对于传统 OPC 增加了安全部分规范，使用不同的映射技术利用 Web Service 与 XML 技术实现网络兼容性，可升级性，独立平台等优点。当前，OPC UA 规范包含核心规范与存取规范两部分内容。

3.1 OPC UA 规范概述

OPC UA规范由十三部分构成[18]，核心规范包括OPC UA Security Model、OPC UA Address Space Model、OPC UA Service、OPC UA Information Model、OPC UA Service Mappings和OPC UA Profiles。存取规范为OPC UA Data Access，OPC UA Alarms and Conditions，OPC UA Historical Access。每种规范对应着一种类型的服务器和客户程序的开发，它们的区别是实现功能的侧重点不同。把OPC UA规范划分成不同的部分是为了把核心设计从底层的运算处理和网络传输分离出来。这使得OPC UA在不改变基础设计的情况下，被运用到未来技术上称为可能。规范中第一部分到第七部分组成技术基础和OPC UA应用程序的实现，第三部分到第五部分是规范的核心，第八部分到第十一部分定义提供传统OPC信息例如实时数据或者警报事件的特定信息模型，第十二部分和第十三部分定义实用的规范。另外，其他一些规范是与其他标准组织合作定义的信息模型，提供相应的信息模型中如何描述信息及如何使用OPC UA描述和传输。

第一部分—概念（Concept）非标准部分主要是OPC UA规范的概要。

第二部分—安全（Security）非标准部分包括OPC UA安全的基本介绍及安全模型。

第三部分—地址空间[19]（Address Space Model）规范中定义OPC UA服务器地址空间基础规则和元素，是第五部分与第八部分规范定义的基础。

第四部分—服务（Service）规范定义了抽象的服务集，服务集中包含一系列方法，这些服务提供访问所有信息模型的通用方式，客户端应用程序使用这些服务函

数完成与服务器的交互，服务器中具体实现服务的功能[20]。

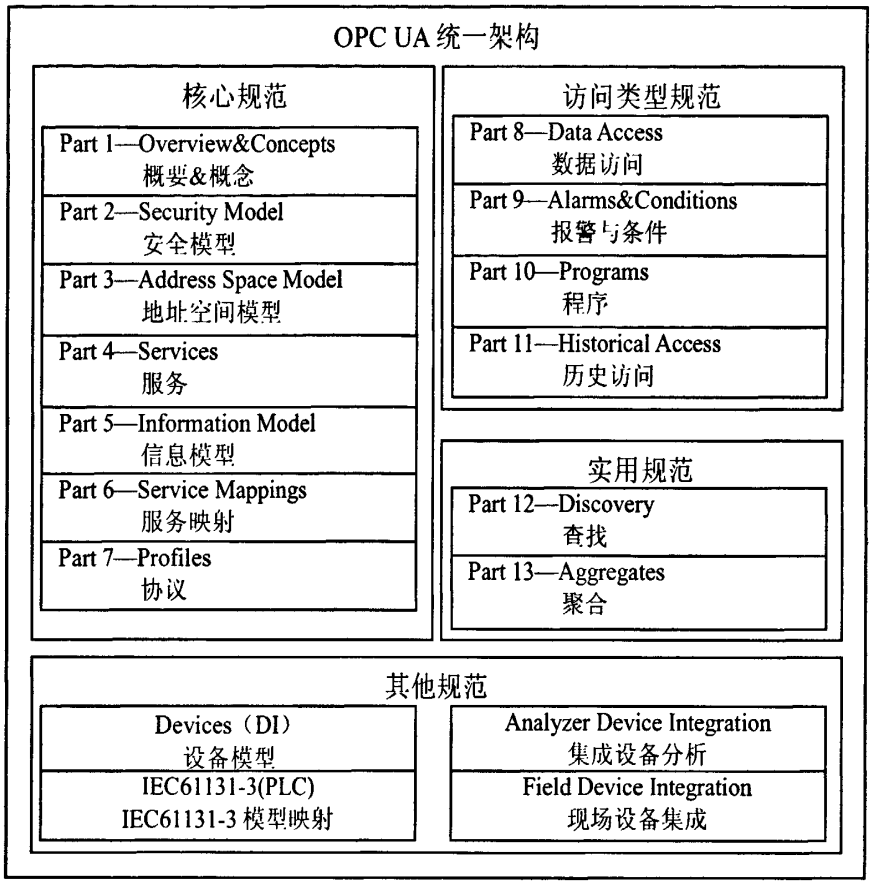

图 3-1 OPC UA 统一架构规范

第五部分—信息模型（Information Model），基本信息模型定义地址空间入口点和基本类型信息[21]，例如数据类型或对象类型，定义了服务器对象的功能和诊断信息。信息模型规范与第三部分第四部分规范组成OPC UA程序开发的核心。

第六部分—映射（Service Mapping）在第四部分中定义的服务都是抽象函数，映射规范确定了客户端与服务器之间信息交互中服务消息的不同数据编码方式、安全机制与传输协议的实现[22]。

第七部分—协议（Profile）协议确定由OPC UA服务器提供的应用程序OPC功能的子集，或是可被OPC UA Client使用的功能，这部分规范为OPC UA定义一组协议。

第八部分— 数据访问（Data Access）这部分为过程数据定义变量类型、属性和状态码等模型，说明OPC UA数据访问的规则。

第九部分—报警与事件（Alarms and Conditions）定义条件和过程报警的模型支持与通过事件发送状态变化。

第十部分—程序（Programs）定义程序执行、操作和监控的状态机模型

第十一部分—历史数据访问（Historical Access）定义访问历史数据存取和历史事件存取两部分。

第十二部分—查找（Discovery）定义如何在网络中查找OPC服务器，以及获取必要的信息建立与服务器的连接

第十三部分—聚合（Aggregates）定义从原始数据计算样本值的方法，主要用于处理实时数据和历史数据。

设备（Devices）为设备的配置和诊断定义通用模型。

IEC 61131-3 定义IEC 61131-3软件模型和OPC UA服务器地址中标准化逻辑控制器的映射。

现场设备集合（Field Device Integration）在电子设备描述语言基础上定义现场设备完整工程单元的模型。

OPC统一体系结构规范是一个不依赖任何平台的标准，借助此标准各种各样的系统和设备能在不同的网络中以客户端/服务器的模式进行通信。OPC统一体系结构通过确认客户端和服务器的身份和自动抵御攻击来支持稳定的、安全可靠的数据通信。通过地址空间与信息模型，信息通过使用标准的和宿主程序定义的数据类型进行表达，服务器定义客户端可识别的对象模型。OPC UA在第四部分定义了一系列服务器所能提供的服务[20]，特定的服务器需要向客户端详细说明它们所支持的服务。通过定义的抽象服务函数，服务器可以提供查看实时数据和历史数据的接口，并且由报警和事件组件来通知客户端重要的变量或事件变化。OPC UA可以被映射到一种通信协议上并且数据可以以不同的形式进行编码来达到传输便捷和高效的目的。通过第三部分地址空间信息模型规范，将被访问的对象以标准信息模型提供给客户端。同时OPC UA规范中定义了二进制结构和XML文档的形式，使OPC UA数据以不同的数据格式传输[23]。另外，数据格式可能被其他标准组织和厂商定义。OPC UA地址空间中使用引用扩充了节点间关联的支持而不是把节点限制在单一层面。OPC UA规范目标是要支持更广泛的服务器，从底层的PLC到企业服务器。对于OPC UA标准的兼容性，规范中规定由传统OPC暴露出来的数据可以通过OPC UA进行映射。厂商可以直接遵循OPC UA标准移植，也可以对先前的产品进行外部封装显示传统OPC数据。

3.2 安全模型

在今天的工业自动化系统中安全变的越来越重要，因为控制系统所处网络不再孤立，安全事故在特定环境中可能造成重大的经济和环境的影响[24]。OPC UA 规范一个很重要的功能就是支持数据在Internet 上远程传输，而如何能在易受攻击的互联网上保证数据传输的安全已经成为首要考虑的问题[27]。而现有OPC 技术的安全性单纯依靠COM 本身的安全机制来保证，而且基于COM 的OPC 技术不能通过Internet 进行数据传输。所以，其安全性无法得到保证，不能适应通过互联网进行远程的实时监控。OPC UA规范的第二部分安全模型[28]保证了在不同的安全要求、威胁和安全策略的环境中运行。数据传输中的安全性、可靠性在传统OPC规范与OPC UA规范的安全对比如表3-1 。

表 3-1 传统OPC与OPC UA安全性对比

	OPC DA规范	OPC UA规范
安全性	完全基于COM/DCOM的安全性无自身的安全设计	有一系列安全机制
可靠性	完全基于DCOM的可靠性	消息序列号、生存期保持
冗余性	无冗余设计	冗余服务器和客户端

OPC UA 安全模型包括客户端和服务器端的认证、用户认证、数据保密性等操作。考虑到在互联网上进行数据传输安全性考虑，OPC UA 服务器或客户端必须要采用一定的安全策略保证数据在互联网环境下系统的安全[25]。OPC UA 采用了以下机制来保证数据采集和传输的可靠性。

（1）OPC UA 定义了一个Getstatus 服务，客户端可以周期性的知道服务器的状态，同时定义了与状态相关的一系列诊断变量。通过这些诊断变量就可以知道服务器各个方面是否正常，此外允许客户端程序订阅服务器状态的变化。

（2）定义了一个生存期保持(keep-alive)的间隔，服务器周期性的发出生存期保持的消息，客户端可以及时的检测到服务器和通信的状态。

（3） 传输的消息都有序列号，客户端程序就可以根据序列号检测数据是否丢失。如果丢失，可以根据序列号重传。

（4）为服务器和客户端设计了冗余机制。

OPC UA定义通用安全架构使用三层安全架构，每一层都有其相应的安全责任与特定的安全机制如图3-2，因此每层都可以使用不同技术实现。

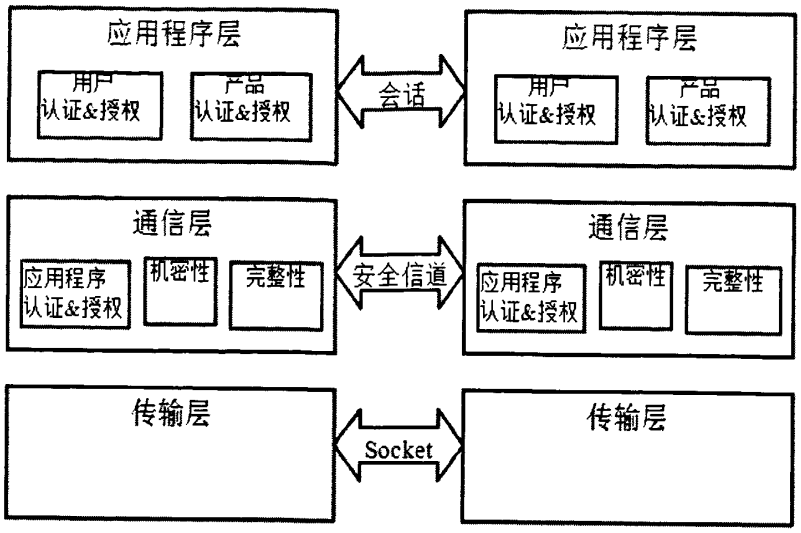

图 3-2 OPC UA三层安全架构

最底层的传输层通过套接字连接负责发送和接收加密的数据，安全信道层通过应用数字签名和加密敏感信息维护交互消息的完整性。另外，OPC UA使用了X509证书认证，引入身份验证和授权的概念。会话层建立在安全信道基础上，在会话中传输现场信息、设置、指示及设备相关的数据，使用身份验证和授权的机制。同时系统定义了错误恢复机制，以便于维护系统可用性。OPC UA的安全模型保证了数据从底层现场设备到MES、ERP系统，从本地到远程的各级自动化和信息化系统的数据可靠传递[29]。

OPC UA程序安全的设计中使用数字证书，证书中包含经第三方确认的不同信息内容。使用数字证书有两个目的：第一，绑定特定信息和公有密钥以便证书的接受者识别证书的所有者。第二，保证公有密钥和相关信息的完整。数字证书有不同的类型和格式，X509v3是OPC UA规范中最常见的证书，包含下表3-2中的内容

表 3-2 X509证书包含字段

Field（字段）	描述
Version	描述证书的版本，在OPC UA中应该是X509v3即版本3
Serial Number（序列号）	必须为正数且由特定的CA颁发的唯一号码
Signature Algorithm	包含由CA用于签署此证书的签名算法
Issuer	标识发行并签署证书的CA,CA的标识符由一个可分辨名称表示
Valid From	证书的有效期开始时的日期
Valid To	证书的有效期结束时的日期
Subject	拥有一个可分辨名称（DN）以表示该证书的实体的标识符。

Public Key	包含公钥类型的主体，以及密钥本身的标识符
<Extension>	使用V3版的X.509证书时提供此扩展。标准化的扩展包括例如密钥用法、证书策略、主题备用名称和CRL分发点
Signature	包含数字证书的颁发者，以及签名

OPC UA应用程序在建立连接中使用三种类型X509证书，第一种是OPC UA应用程序实例证书，使用受信任的私有或公有认证标识主机上OPC UA应用程序的有效性。每个OPC UA产品的安装都需要X.509v3证书。第二种是OPC UA软件证书，这个证书确定OPC UA产品的特定版本，在建立连接过程中通过交换软件证书确定相互通信方式与各自支持的服务。第三种OPC 用户证书用于识别建立连接过程中访问服务器的数据的当前用户。

3.3 地址空间

OPC UA 客户端开发的一个主要任务是服务器地址空间的浏览，需要了解地址空间数据的组织结构，规范第三部分 Address Space Model 对地址空间做详细的介绍，本节简单介绍地址空间方便客户端开发。在传统 OPC 规范中，各个规范使用单独的地址空间与一套相应的服务，所以在使用不同功能时访问不同的地址空间，降低程序运行效率，数据采集后在客户端进行处理增加了客户端开发难度。OPC UA 提供了一个一致的、完整的地址空间，将传统 OPC 服务器数据、报警与事件和历史信息统一到一个地址空间[26]里，用一套统一的服务器向外提供数据与服务。

3.3.1 地址空间结构

为了提高客户端与服务器的互操作性与客户端浏览地址空间效率 OPC UA 地址空间的节点都是以层次结构进行组织的[30]。尽管地址空间里的节点通过分层结构组织容易客户端浏览，但节点之间相互引用，把地址空间组织成一个相互联系的网状结构。节点之间通过引用相互连接，引用描述两个节点之间的关联，含有引用的节点称为源节点，被引用的节点为目标节点。所有引用通过引用类型定义，例如：HasProperty、HasSubtype。引用类型分为层次引用和非层次引用，层次引用用于组织地址空间，层次引用可以用于标识现场设备的层次结构；非层次引用用于联系对象的组件，例如温度传感器的温度值。

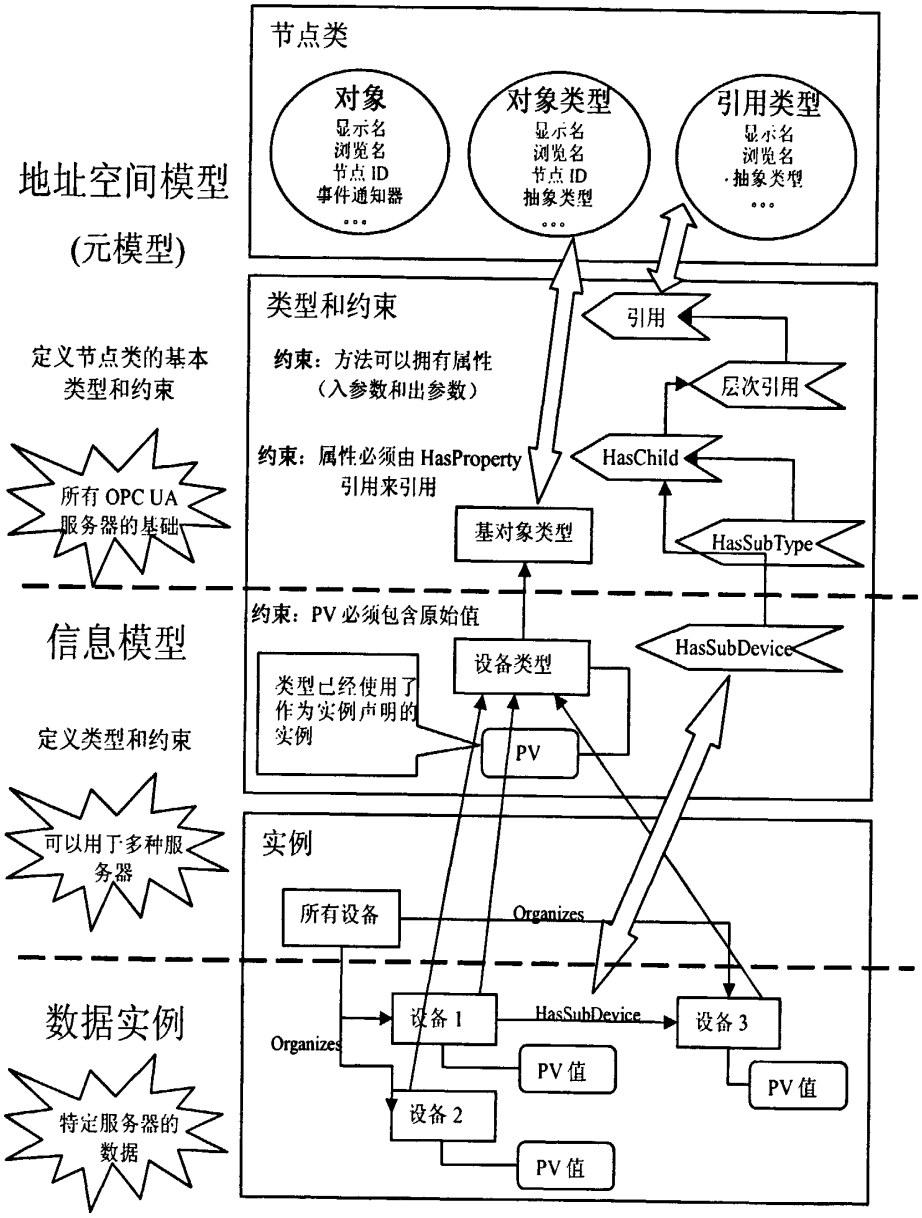

图 3-3 地址空间模型、信息模型与数据

客户通过 OPC 服务来访问地址空间信息，地址空间的基本对象有：变量、对象和引用等描述和管理采集自底层的数据。抽象底层系统的设备为程序中的对象，对象在地址空间中的表现为一个一个节点。图 3-3 描述了从设备数据映射到地址空间信息模型的过程。

通常为了简化客户端访问整个地址空间，OPC UA 使用视点(view)概念。例如：服务器提供维护服务器的配置信息视点，或服务器中所有温度传感器的温度值

视点。由此可见视点是地址空间的一个子集，视点的缺省是整个地址空间。视点跟地址空间一样，组织成层次结构，并包含节点间的引用。视点在地址空间中表现为一个节点，这个节点就是进入视点内容的入口点。视点对客户端可视，客户使用视点节点 NodeId 作为筛选参数，在浏览地址空间的某个视点时，将只看到这个视点内的节点。但是视点仅用于浏览和查询地址空间信息，读写具体的节点时不能使用。服务器可以在地址空间里定义它们自己的视点，或者视点可以通过调用 OPC UA 服务来创建。

3.3.2 地址空间节点及引用

地址空间中的所有节点都由基节点继承而来，基节点是地址空间的元数据模型，是抽象的节点类型不能被实例化，基节点包含了所有节点公有属性如表 3-3.

表 3-3 节点的公有属性

属性	数据类型	描述
NodeId	NodeId	唯一地标识 OPC UA 服务器中的一个节点，并用于定位 OPC UA 服务中的节点
NodeClass	NodeClass	识别节点的枚举类别如对象或方法
BrowseName	QualifiedName	浏览 OPC UA 服务器时，用于标识节点。它不是本地化的文本
DisplayName	LocalizedText	用来在用户界面中显示节点的名称。因此，它本地化的文本
Description	LocalizedText	可选属性，包含该节点的本地化说明文本
WriteMask	UInt32	可选，指定了节点属性是可写的，可以由 OPC UA 客户端修改
UserWriteMask	UInt32	可选，指定了由当前连接到服务器的用户是否可以修改节点的属性

地址空间中的其他节点都是由基节点派生出来的，有对象、对象类型、引用类型、变量、变量类型、数据类型、方法和视点，这些节点都是对基节点的属性进行相应的扩展生成的。地址空间中可用节点类如表 3-4

表 3-4 地址空间节点类

节点类	描述
对象（Object）	对象是变量、方法和事件的容器
变量（Variable）	变量代表对象的数据或属性或是节点的属性
方法（Method）	方法是对象的组件拥有一组输入与输出参数
视域（View）	视域是地址空间的子集，作为浏览操作的过滤器用做访问入口
变量类型（Variable type）	通常是类型实例中可找到的属性

引用类型（Reference Type）	定义节点之间可能的引用类型
数据类型（Data type）	描述在变量中值的内容

地址空间中最重要的功能单位就是对象，对象使用 HasComponent 引用类型将变量、方法等组织在一起，并可以产生事件通知。变量类节点表示对象的数据属性，具有值，工程单位，时间戳和质量等属性。客户端可以读取变量的值、订阅该值的变化和写入值。例如变量可以用于表示由温度传感器测量到的温度值，或一个用来管理控制程序的设置值。变量一般用于提供地址空间的数据，包括用于描述节点的配置数据或元数据。方法类节点代表可以被客户端调用的操作，由服务器执行并向客户端返回结果。每个方法都指定了客户端发送的输入参数和客户端接收的输出参数。使用方法的例子有打开阀门或启动电机，以及更复杂的任务。事件表示系统中发生的有重要意义的事情。其中表示异常情况的事件称为警报，例如当温度传感器的值超过高限后会产生“高报警”；而事件一般用来代表需要注意的系统更改，如用户的操作、设定值的更改等。

为了简化客户访问地址空间，OPC UA 服务器创建了一个视点(view)，视点在地址空间中表现为一个节点。视点就是地址空间的一个子集，缺省值就是整个地址空间。视点就是简化了地址空间的层次结构，其将地址空间分成若干块，其实视点与地址空间一样，也是被组织成为一个层次结构，并包含节点间的引用。视点是一个根节点，视点对客户可视，客户通过浏览节点确定其结构。例如，在工厂中可以把所有温度传感器的温度设定为一个视域，这样对于维修人员来说查看方便，提高了工作效率，而且可以隐藏其他无权查看的内容。

3.4 OPC UA 服务规范

OPC UA 服务规范将划分成了不同的服务集，每个服务集定义了用于访问服务器特定部分的逻辑分组，解决了现有OPC规范在应用时服务重叠问题，包括安全信息服务集、会话服务集、节点管理服务集、视图服务集、属性服务集、方法服务集、监视服务集、订阅服务集、查询服务集等[31]。服务集的定义位于应用程序层，是OPC UA客户端访问服务器提供信息模型的方法。客户端与服务器之间通过接口提供服务，服务的定义是独立于传输协议和编程环境的，这是与传统OPC根本的不同。OPC UA 服务器对客户端提供两个功能，它们允许客户端向服务器发出请求并从服务器接收响应，也允许客户端向服务器发送通知。而服务器使用通知来报告事件，比如报警、数据值变化、事件和程序的执行结果。本节中主要描述这些服务函

数及这些不同函数提供的功能。根据服务集在使用中的不同将其分类如表3-5。

表 3-5 根据实际使用划分服务集

实际使用	服务集或服务
查找服务器	Discovery Service Set
管理客户端与服务器间通信	Secure Channel Service Set Session Service Set
查找地址空间中信息	View Service Set
读/写数据	Read and Write Service
订阅数据改变与事件通知	Subscription Service Set Monitored Item Service Set
调用服务器端的函数	Call Service
访问历史数据和事件	HistoryRead and HistoryUpdate Service
查找整个地址空间信息	Query Service Set
修改服务器地址空间的结构	Node Management Service Set

客户端应用程序的开发测试服务器实现的功能，功能的具体实现需要调用服务器实现的服务即调用服务集中的方法，根据服务器返回的消息在客户端显示。下面介绍在客户端实现中使用的服务集中重要的服务表3-6。

表 3-6 重要服务

Service	Description
Browse	浏览服务器地址空间，客户端定义一个开始节点和过滤
BrowseNext	标准，服务器返回结果
TranslateBrowsePathsToNodeIds	基于对象类型信息获取对象组件的NodeId
Read	读取节点属性包括变量的值
Write	写节点的属性包括变量的值
Publish	从服务器向客户端发送变化的数据或事件
Republish	
Call	调用服务器中的方法
HistoryRead	读取变量的历史值或事件历史
AddNodes	向服务器地址空间中增加节点
AddRdference	增加节点间的引用
DeleteNodes	删除地址空间中的节点
DeleteReference	删除地址空间中节点间的引用

1、查找服务器

查找服务集用于查找可利用的OPC UA服务器，并获取服务器的端点信息，包括连接服务器使用的网络协议和安全配置。

FindServers 调用FindServers服务返回所有注册的服务器列表，如果不存在查找服务器，服务器返回本身

GetEndpoints 服务结果返回服务器支持的端点信息，包括建立连接需要的信息，如服务器的网络地址和安全设置。FindServers与GetEndpoints都不需要建立与服务器的连接可以实现。

2、客户端与服务器之间连接的管理

OPC UA需要建立不同级别的通信信道保证安全、灵活和可靠的数据通讯要求。安全信道服务集定义的服务用于打开安全信道，安全信道是客户与服务器之间

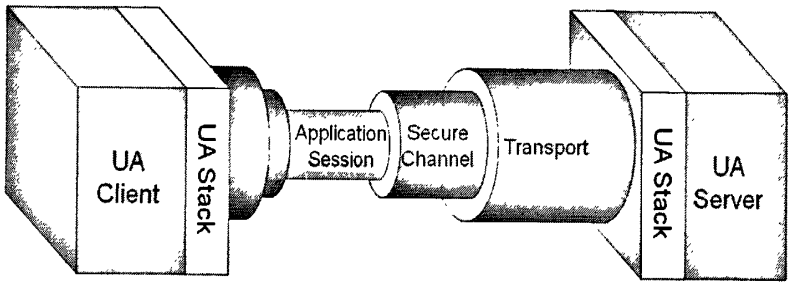

图 3-4不同级别的安全信道

安全的连接，保存一组客户与服务器共享的一组密钥。

OpenSecureChannel 用于打开或更新安全信道保证在会话中传输信息的机密性与完整性，服务需要通信栈当消息发送与接收期间应用不同的安全算法。

CloseSecureChannel 关闭安全信道

会话服务集中定义了建立应用程序层连接需要的服务，会话服务建立在安全信道之上。

CreateSession 用于创建客户与服务器之间会话，服务器返回会话的标识符。如果在确定的超时时间内没有发出服务请求则服务器会使用CloseSession关闭会话。会话到调用ActivateSession服务才能使用，提供软件证书及用户身份标识。服务器验证证书确定是否激活会话。

3、查找地址空间中的信息

OPC UA提供描述与传输信息的能力，如何查找服务器地址空间中不同类型数据。视域服务集包含重要服务是Browse与BrowseNext浏览地址空间，服务用于访问服务器地址空间信息模型。

Browse 用于客户端浏览地址空间，通过客户端确定开始节点与浏览过滤器，服务器返回一组通过引用与开始节点连接起来的节点。如果传递一组开始节点则返回与每个节点相连接的节点，为了建立树形结构客户端开发中使用传递一个开始节

点。

BrowseNext 用于在返回的结果不能在一个调用中返回通过设置继续点继续浏

监视项也一并删除。

ModifySubscription 修改订阅的设置。

Publish 用于客户向服务器发送请求消息，可靠的数据通信的另一个要求是探测丢失信息，通过交换每个通知消息的序列号来实现。如果探测到丢失的消息，客户发送Republish服务获取丢失的通知消息。

6、调用服务器端函数

客户端可以调用服务器暴露出的方法，方法服务集中定义Call服务来实现。方法是对象的组件可以在对象的背景下调用。

Call 服务用于实际调用方法，这个服务允许调用一组方法减少客户与服务器交互次数。

7、修改地址空间的结构

节点管理服务集使OPC UA客户可以在服务器的地址空间中创建和删除节点和引用。当客户端与服务器断开连接后所做修改仍存在与地址空间中。

AddNodes 服务用于向服务器地址空间中增加一个或多个节点

AddReferences 用于在地址空间节点间创建一个或多个引用，确保引用的类型与目标节点的类型匹配。

DeleteNodes 在地址空间中删除节点，如果删除的节点包含其他子节点则子节点也被删除。

DeleteReferences 用于删除节点间的一个或多个引用。

通过上面的叙述看到有两种OPC UA服务，一种用于创建通信环境例如安全信道、会话、订阅和监视项服务集，另一种是用于交换信息例如Browse地址空间的结构信息，Read、Write和Publish访问数据和调用执行方法。与传统OPC相比，服务的数量减少，目的是提供一致的服务。服务以抽象方式定义可是使用不同传输和不同的开发环境完成平台独立、可扩展、高性能、Internet访问和跨防火墙能力。所有的服务都可以以同步或异步方式调用。

3.5 映射技术

为了应用程序的开放性和应用程序之间的互操作性 OPC UA 的服务都是以抽象的服务，规范中第六部分确定了实现时的技术映射。OPC UA 应用程序数据交换有三个任务：数据编码（data encoding、通信加密（securing the communication）和数据传输（transportiing the data）。OPC UA 应用程序一般分为多个功能层如图 3-5 所示

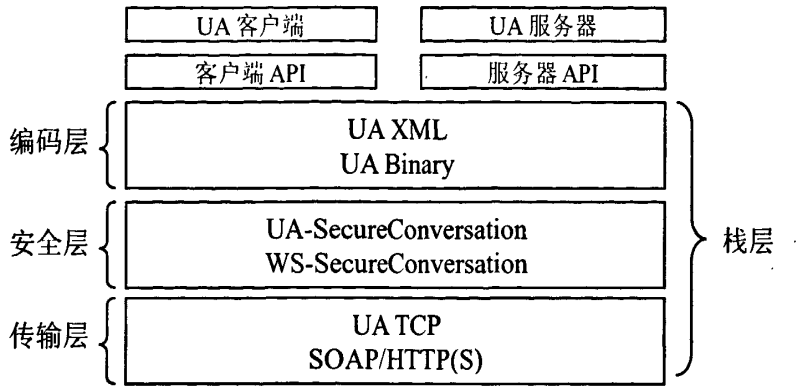

图 3-5 映射

3.5.1 数据编码

数据编码是服务消息的序列化操作，包括输入与输出参数格式化为网络格式。OPC UA 现在确定了两种编码 OPC UA Binary 和 XML。

OPC UA Binary-线上性能与开销对工业系统是至关重要的，例如嵌入到控制器中的应用程序需要更高的运行性能，所以 OPC UA 工作组定义了 OPC UA Binary 编码格式提供快速的编码与解码，具有数量小高效的特性。

XML-有时我们需要不同应用程序、不同平台和用户都能识别的数据交换格式。在这种情况下 XML 文档具有标准化的结构发挥重要作用。在这种情况下在操作级的应用程序例如 MES 与 ERP 层的软件使用 XML 格式交换数据，许多的内置类型依据 XML 规范[W3C04a] [W3C04b]进行编码。

3.5.2 安全协议

为了映射服务规范定义的抽象服务，OPC UA 定义两种安全协议为：WS-SecureConversation 与 UA-SecureConversation，两种协议均基于证书建立连接。

WS-SecureConversation 是 WS-Security 的扩展规范定义了通过 Web Services 进行安全数据交换的概念和技术。WS-SecureConversation 保护 XML 数据的最优方案，因为其使用了 XML 加密及签名对于开发运行在不同操作系统及不同级别自动化领域的 OPC UA 应用程序提供了很好的方式。

UA-SecureConversation 是 OPC UA 工作组自己定义的安全协议，其实它不是一个新的安全协议。WS-SecureConversation 是 XML 文档的交换的高效协议，但是在

考虑性能与开销情况下，程序运行的快速性与高效性是决定性原因，所以很少使用XML。UA-SecureConversation 不是直接映射抽象服务而是使用经编码的服务消息作为基本信息增加其他安全相关的消息头。

3.5.3 传输协议

OPC UA 规范中定义两种传输协议 UA TCP 与 SOAP/HTTP，这两种协议用于在UA客户端与UA服务器之间建立网络层的连接。

UA TCP 使用 UA TCP 的传输层可以获取快速与简单的网络通信，根据不同的要求采用 TCP 通讯协议，第一，发送和接收数据的缓冲区的大小，可以在应用层进行配置。第二，不同的终点的 OPC UA 服务器可以共享一个 IP 地址和端口。最后，对传输层发生的错误快速反应并恢复。

SOAP/HTTP 代表基于 HTTP 的 SOAP 协议，因为其简单且可以穿越防火墙，因此是一种在 Web 服务的环境中被广泛接受的通信方案。因为 HTTP 网络协议的标准端口用于传输数据，在防火墙中并没有打开其他的端口，这意味着 OPC UA 应用程序可以使用类似与网络浏览器与服务器之间的通信加密方式来通过互联网互相加密。

3.6 本章小结

本章主要介绍了 OPC UA 规范，对 OPC UA 规范中关键技术做详细介绍从而说明了 OPC UA 规范所具有的特点。对于客户端的开发重要的是浏览地址空间的内容，对地址空间的相关信息做简单的介绍。服务函数的定义是规范的第四部分内容是客户端与服务器的实现功能的主要构成，对规范中定义的抽象函数说明了其具体的功能。OPC UA 规范体系庞大，对于实现客户端程序需要了解规范中各个部分的内容。对客户端功能的实现需要对第四部分服务规范做主要研究。

第 4 章 OPC UA 客户端实现与功能测试

通过.NET 开发平台与 OPC UA 规范的相关介绍，了解 OPC UA 应用程序开发的依据。本文中客户应用程序开发使用 C#语言，开发工具为 Visual Studio 2008，C#是微软在.NET 平台下首推的语言，具有像 C++、Java、VB 和 Delphi 这些语言的最佳特性，它是完全面向对象的，并且是强制类型的，支持垃圾收集、声明特性、属性和集合等，所以使用 C#语言可以降低开发难度是客户端开发的趋势[34]。OPC UA 客户端应用程序开发的主要任务是通过服务器接口建立与服务器的连接、浏览和操作服务器的地址空间、测试服务器实现功能，简单的客户端就是服务器地址空间浏览器。文中根据 OPC UA 规范使用 OPC 基金会发布 SDK 开发客户端程序，主要介绍通过服务调用将 SDK 层提供的服务器地址空间数据可视化，并将用户的操作转化为 API 的调用，实现对服务器功能测试。

4.1 OPC UA 客户端架构

OPC UA 的体系结构与传统 OPC 一样使用简单的客户端/服务器（Client-Server）模式，服务器提供服务客户端使用服务完成任务，且 OPC UA 服务器与客户端可以互为服务器或客户端。与传统 OPC 的主要不同在于 OPC UA 的通信是基于消息传送机制的，且其内部工作机制也不同。如图 4-1 所示 OPC UA 的客户端结构包括 OPC UA 客户端应用程序、OPC UA 通信栈、OPC UA 客户端 API。

客户端应用程序使用 API 调用服务器提供的服务，通信栈将客户端的 API 调用转换成消息，并通过底层发送给服务器。客户与服务的交互有两种方式：一种是客户端的服务请求，服务请求经底层通信实体发送给 OPC UA 通信栈，通过 OPC UA 服务器接口调用请求/响应服务，请求的任务将在服务器的地址空间中执行，执行完成后返回一个响应消息[32]。另外一种为发送发布请求，请求服务器发布数据或通知消息，发布请求经过底层通信实体发送给 OPC UA 通信栈，通过 OPC UA 服务器接口发送给预定，当预定指定的监测项探测到数据变化或者事件/报警发生时，监视项生成一个通知发送给预定并由预定发送给客户。

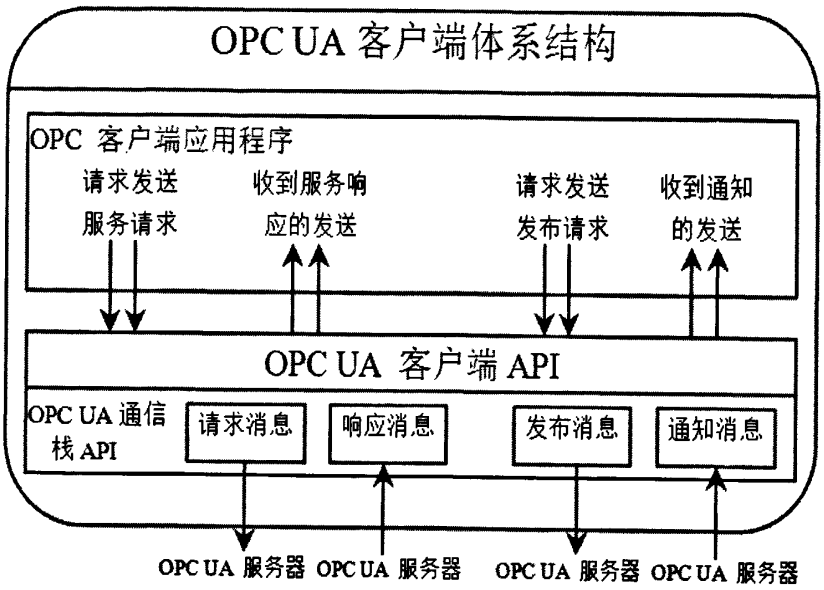

图 4-1 客户端体系结构

4.2 OPC UA 客户端实现基础

通过 OPC UA 规范确定了 OPC UA 客户端应该实现的基本功能，将单独的功能分别在客户端界面控件中实现，分别确定了查找服务器控件、连接服务器控件浏览地址空间控件、节点属性显示控件与节点监视控件五个部分组成。每个部分都承担不同的功能，将用户对功能的操作通过客户端 API 转化为服务消息。在图 4-2 中显示了程序设计的框架图，第一部分为实现客户端功能控件[33]包括：查找服务器部分（Discovery Control）、浏览地址空间部分（Browse Control）、属性视图部分（Attribure List Control）与监视操作部分（MonitoredItems）。在此之下是客户端应用程序接口（ClientAPI）部分提供了可用于查找服务器、建立会话与订阅处理类，这些类是基于 C#实现是 SDK 暴露出来的接口。为了发送与接收服务消息，客户端与服务器都需要使用通信栈，通信栈包含了编码层、安全层与传输层三层机构。

本文中开发客户端应用程序的过程中需要添加 OPC UA 类库文件，添加对两个文件的引用

```
using  Opc.Ua.Core
using  Opc.Ua.Client
```

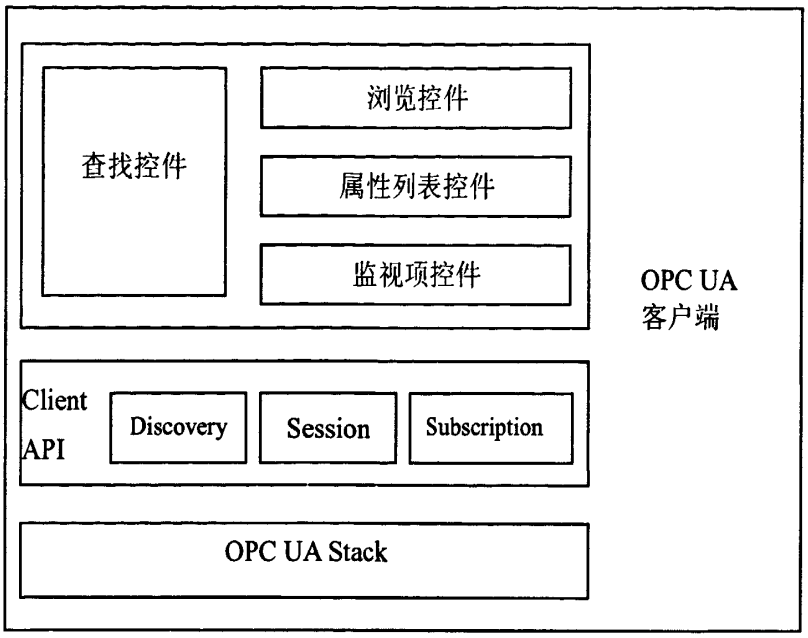

图 4-2 客户端程序结构

4.3 OPC UA 客户端功能实现

OPC UA 客户端功能的实现主要是使用客户端 API 实现客户端访问与测试功能，在程序中需要包括查找服务器、客户端与服务器的连接管理、浏览地址空间、订阅数据变化或事件几部分内容，下面通过规范中对服务函数的描述到客户端程序的实现分别进行叙述。

4.3.1 查找服务器

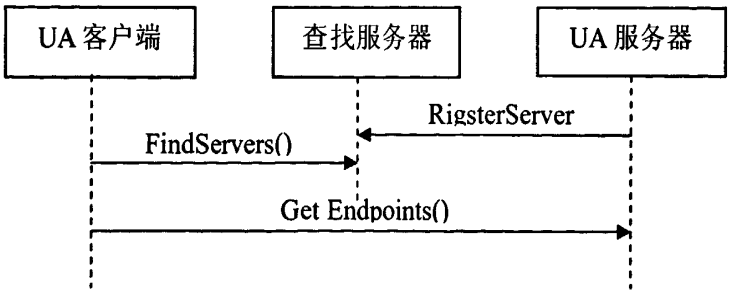

图 4-3 查找服务器过程

在工业控制系统中，底层含有多个智能设备即可能存在多个 OPC 服务器程序。客户端应用程序需要查找所有运行于系统中的服务器程序。规范中有 Discovery 规范定义查找的过程，OPC 基金会发布了本地查找服务器程序，可以在本机运行查找服务器获取已经注册的服务器程序，图 4-3 中说明了查找服务器的过程。

客户端中实现查找过程使用查找功能块，使用下拉列表控件 ComboBox 的 DropDown 事件实现查找过程。点击下拉列表触发事件响应函数，在程序中设置超时时间用于查找过程，

```
EndpointConfiguration configuration =
        EndpointConfiguration.Create(m_configuration);
configuration.OperationTimeout = 5000;
```

查找过程主要使用查找服务集中的 Findservers 与 GetEndpoints 服务，图 4-4 说明在程序查找过程的实现。如本机中运行本地查找服务器，则提供了在本地网络节点中可用的一组 OPC UA 服务器。默认的情况下，本地查找服务器使用端口 4840，因此本地查找服务器以 opc.tcp://<Node>:4840 定位，在本机上运行的服务器

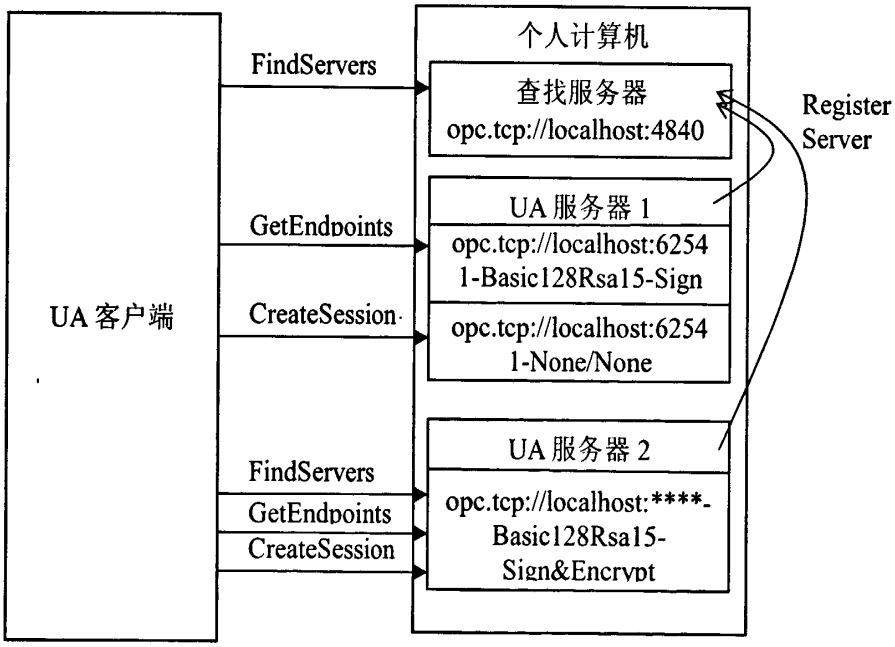

图 4-4 客户与服务器不同连接

需要使用 RegisterServer 服务注册。在图 4-2 中使用 Findservers 服务获取已经注册的服务器及其查找端点，不需要建立安全的连接。使用 discovery URL 与服务器建

立无安全的连接并使用 GetEndpoints 服务可获取服务器的端点与端点的安全设置信息。

在下拉列表中储存查找到的服务器端点，选择好端点与安全设置可以使用 CreateSession 建立与服务器的连接。图 4-2 说明了在没有本地查找服务器运行的情况，如果在系统中仅有一个服务器，服务器可以运行在标准接口，由于所有服务器都要实现 Findservers，所以客户端调用 FindServers 服务会返回服务器本身。应用程序中的主要实现为 UrlCB_DropDown 事件如下：

```
private void UrlCB_DropDown(object sender, EventArgs e)
{
    ... ...
        // 连接到本地的查找服务器并查找可用的服务器.
        using (DiscoveryClient client = DiscoveryClient.Create(new
                    Uri("opc.tcp://localhost:4840"), configuration))
        {
          ApplicationDescriptionCollection servers = client.FindServers(null);

          // 使用可用的服务器地址填充下拉列表
          for (int ii = 0; ii < servers.Count; ii++)
          {
            for (int jj = 0; jj < servers[ii].DiscoveryUrls.Count; jj++)
            {
              string discoveryUrl = servers[ii].DiscoveryUrls[jj];


              if (discoveryUrl.EndsWith("/discovery"))
              {
                discoveryUrl = discoveryUrl.Substring(0, discoveryUrl.Length -
                            "/discovery".Length);
              }

              int index = UrlCB.FindStringExact(discoveryUrl);

              if (index < 0)
              {
                UrlCB.Items.Add(discoveryUrl);
              }
```

```
            }
        }
    ... ...
}
```

4.3.2 客户端与服务器连接管理

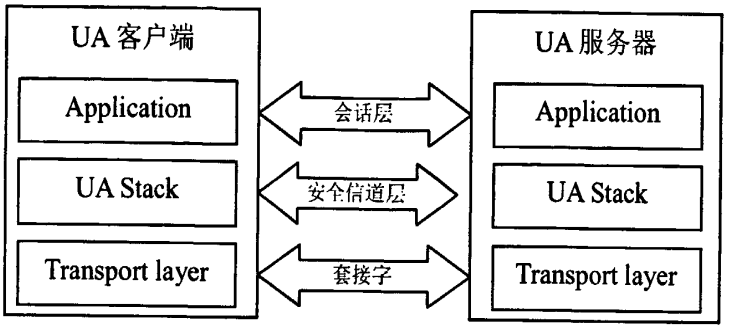

图 4-5 客户端与服务器之间连接

工业数据传输过程中控制系统不再孤立，数据的安全性是控制的必要条件。OPC UA 规范要求建立不同级别的通信渠道保证安全、灵活和可靠的数据通信。低级网络传输信道是由 UA Stack 实现，其中的安全方面处理都是有 OPC 基金会提供。用于认证用户的应用程序级的会话由客户和服务器处理，客户端 SDK 隐藏了所有连接管理只公布了 API，所以客户端调用服务就可以与服务器建立安全连接。

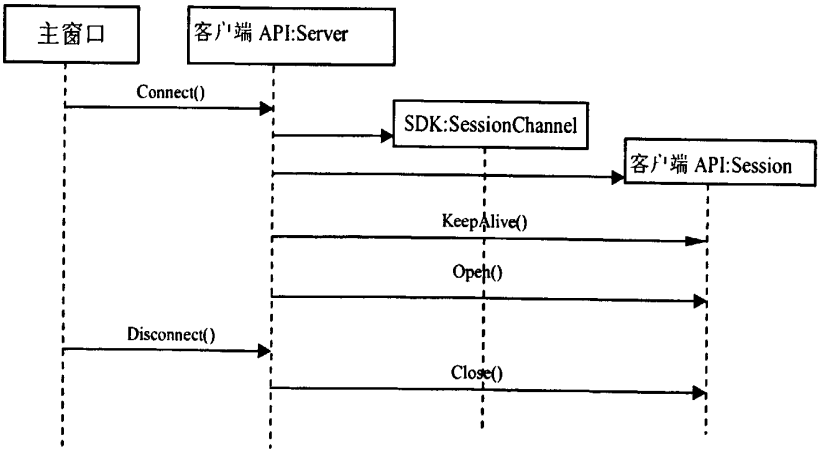

图 4-6 建立连接过程

客户端与服务器建立连接包括建立安全信道、创建会话、激活会话。安全信道的管理是在通信栈层实现，如图 4-5 在安全信道层实现应用程序间的安全认证与基于消息的安全通信。每个传输消息经过签名与加密保证消息完整性与私密性，客户

端创建安全信道需要调用 SDK 的函数。客户与服务器之间的连接主要使用安全信道服务集中 OpenSecureChannel 与会话服务集中 CreateSession、ActivateSession 与 CloseSession 服务函数。客户端界面中使用 Connect 按钮控件的 Click 事件实现连接操作，应用程序的实现定义 Server_ConnectMI_Click 函数，函数中调用 SDK 中 SessionChannel.Create（）服务建立客户与服务器的连接。图 4-6 显示了客户端与服务器通过服务函数建立连接与断开连接的过程，下面使用程序代码说明了具体的连接过程。

```
private void Server_ConnectMI_Click(object sender, EventArgs e)
{
    … …

        // 创建连接到服务器的安全信道.
        ITransportChannel channel = SessionChannel.Create(
            m_configuration,
            endpointDescription,
            endpointConfiguration,
            m_clientCertificate,
            m_configuration.CreateMessageContext());

        ConfiguredEndpoint endpoint = new ConfiguredEndpoint(null,
                            endpointDescription, endpointConfiguration);

        // 创建会话对象.
        m_session = new Session(channel, m_configuration, endpoint,
                                m_clientCertificate);

        // 设置保持会话活跃的回调
        m_session.KeepAlive += new KeepAliveEventHandler(Session_KeepAlive);

        // 创建会话
        m_session.Open("UA Client", null);

        // 填充浏览树.
        PopulateBranch(ObjectIds.ObjectsFolder, BrowseNodesTV.Nodes);
        … …
}
```

4.3.3 浏览地址空间

现场控制系统中，需要采集底层设备的数据，将设备数据可视化或写入数据库。OPC UA 客户端模拟了数据的流动，客户端与服务器建立连接后，将地址空间中的内容可视化，这是客户端开发中最重要的工作。规范中定义了 Browse、Read、ReadValues 与 WriteVaules 等服务函数完成数据采集工作。通过调用规范中定义的浏览及读/写抽象服务将地址空间结构与数据可视化。节点的概念在 OPC

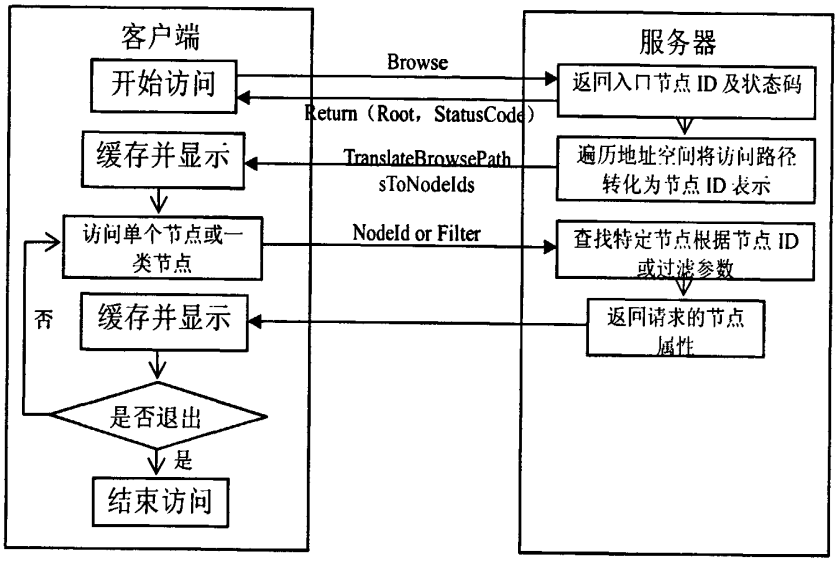

图 4-7 客户端浏览服务器地址空间

UA 中非常重要用于构成地址空间并提供属性值。在 OPC UA 中对象数据更加丰富。服务器地址空间中以引用的方式连接，程序开发中使用树形结构图来显示地址空间结构。

客户端程序界面浏览功能块实现浏览地址空间，浏览功能使用 TreeView 控件，开始浏览操作时输入参数传递一个开始节点，具体的流程如图 4-7 客户端浏览服务器的交互流程图。客户端传递开始节点，服务器就返回与开始节点通过引用的其他节点。浏览功能的实现使用规范中定义 Browse 服务函数，在程序实现文件中添加 FormUtils.cs 文件，其中定义了在程序应用中需要经常使用的函数，浏览服务函数在该类中实现代码如下：

```
public static ReferenceDescriptionCollection Browse(Session session,
                    BrowseDescriptionCollection nodesToBrowse, bool throwOnError)
{
        ... ...
        ReferenceDescriptionCollection references = new
```

```
                                    ReferenceDescriptionCollection();
        BrowseDescriptionCollection unprocessedOperations = new
                                    BrowseDescriptionCollection();
        ... ...
          session.Browse(
            null,
            null,
            0,
            nodesToBrowse,
            out results,
            out diagnosticInfos);
         ... ...
          for (int ii = 0; ii < nodesToBrowse.Count; ii++)
          {
            // 检查是否发生错误
            if (StatusCode.IsBad(results[ii].StatusCode))
            {

              if (results[ii].StatusCode == StatusCodes.BadNoContinuationPoints)
              {
                unprocessedOperations.Add(nodesToBrowse[ii]);
              }

              continue;
            }
            // 确认是否返回所有引用信息
            if (results[ii].References.Count == 0)
            {
              continue;
            }
            // 保存结果
            references.AddRange(results[ii].References);
          }
            // 检查连续点如果存在连续点使用session.BrowseNext( )处理
        ... ...
}
```

首先定义 references 变量用于储存查找到的节点间引用，unprocessedOperations

用于储存未完成的继续点信息。使用 session.Browse 函数浏览地址空间，将返回的信息储存到输出参数 results 中，结果返回后需要判断结果的状态妈信息，使用 for 循环将返回的结果分类，状态码为含有连续点则将连续点储存到 unprocessedOperations 中，如果状态码显示结果状态良好则使用 AddRange（）将引用储存到 reference 中。使用 Browse 服务返回值是与输入参数节点相连接的节点的所有引用。在树型结构视图使用 PopulateBranch 来填充树形空间以显示地址空间结构，具体功能的代码实现为：

```
private void PopulateBranch(NodeId sourceId, TreeNodeCollection nodes)
{
        … …
        // 查找节点的所有组件.
        BrowseDescription nodeToBrowse1 = new BrowseDescription();

        nodeToBrowse1.NodeId = sourceId;
        nodeToBrowse1.BrowseDirection = BrowseDirection.Forward;
        nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates;
        … …
        // 查找根节点下包含的节点（无语义引用）.
        BrowseDescription nodeToBrowse2 = new BrowseDescription();

        nodeToBrowse2.NodeId = sourceId;
        nodeToBrowse2.BrowseDirection = BrowseDirection.Forward;
        nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes; //
        … …
        BrowseDescriptionCollection nodesToBrowse = new
                                            BrowseDescriptionCollection();
        nodesToBrowse.Add(nodeToBrowse1);
        nodesToBrowse.Add(nodeToBrowse2);

        // 取回服务器中所有的引用.
        ReferenceDescriptionCollection references = FormUtils.Browse(m_session,
nodesToBrowse, false);

        // 处理返回结果.
        for (int ii = 0; ii < references.Count; ii++)
        {
```

```
            ReferenceDescription target = references[ii];

            // 增加节点.
            TreeNode child = new TreeNode(Utils.Format("{0}", target));
            child.Tag = target;
            child.Nodes.Add(new TreeNode());
            nodes.Add(child);
        }

        // 显示源节点的属性.
        DisplayAttributes(sourceId);
    ... ...
}
```

PopulateBranch 函数的输入参数为根节点 sourceId，将结果填充到 sourceId 节点中的子节点中是 PopulateBranch 函数的作用。函数中定义两种引用类型的浏览操作，Aggregates 与 Organizes 两种引用类型的主要区别是语义与无语义类型。Aggregate 类型将返回根节点下的子节点信息，Organize 返回节点的数据等信息。关于这两种形象描述可以参考地址空间中图 3-3。返回的引用信息使用 child.Nodes.Add(new TreeNode())增加节点显示到浏览控件 BrowseNodesTV 中。

使用 Browse 服务函数返回的结果仅用于填充浏览树，节点的数据如 Name、Value 与 DataType 等信息需要使用 Read 服务函数。在地址空间控件中中选择节点显示该节点的信息，如果节点包含子节点需要将包含的子节点填充到浏览树中。客户端界面中使用 AfterSelect 事件完成填充，定义 BrowseNodesTV-AfterSelect 函数实现。

```
private void BrowseNodesTV_AfterSelect(object sender, TreeViewEventArgs e)
{
    ... ...
    // 填充节点
    PopulateBranch((NodeId)reference.NodeId, e.Node.Nodes);
    ... ...
}
```

PopulateBranch（）函数中调用 DisplayAttributes(sourceId)显示节点的信息。在客户端程序界面中添加属性视图功能块，使用 ListView 控件显示节点属性信息控件定义为 AttributesLV，在属性视图控件中调用读服务获取引用节点的信息，使用读服务的过程如图 4-8。

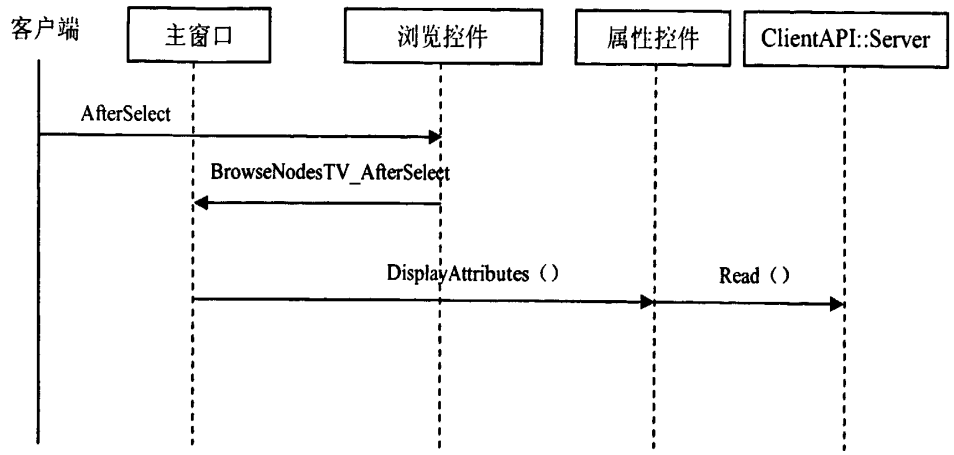

图 4-8 读服务过程

```
private void DisplayAttributes(NodeId sourceId)
{
    … …
    AttributesLV.Items.Clear();

    ReadValueIdCollection nodesToRead = new ReadValueIdCollection();
    … …

    // 获取服务器属性引用.
    ReferenceDescriptionCollection references = FormUtils.Browse(m_session,
nodesToBrowse, false);
    … …
    // 读取所有的值.
    m_session.Read(
      null,
      0,
      TimestampsToReturn.Neither,
      nodesToRead,
      out results,
      out diagnosticInfos);
      … …
    // 处理返回结果.
    for (int ii = 0; ii < results.Count; ii++)
```

```
        {
            ... ...
            // 获取属性名.
            name = Attributes.GetBrowseName(nodesToRead[ii].AttributeId);
            //获取数据类型
            datatype = typeInfo.BuiltInType.ToString();
           //获取值
            value = Utils.Format("{0}", results[ii].Value);
            ... ...
          }

          // 在列表视图中增加名字/值.
          ListViewItem item = new ListViewItem(name);
          item.SubItems.Add(datatype);
          item.SubItems.Add(value);
          AttributesLV.Items.Add(item);
      ... ...
    }
```

使用 AttributesLV.Items.Clear()对属性控件清空上次显示的信息，将所有需要读取的节点储存到 nodesToRead 中。nodesToRead 作为 Read 函数输入参数传递到服务器，服务器返回节点数据储存到 results，分别使用 GetBrowseName、typeInfo.BuiltInType.ToString、Utils.Format 将节点的包含的字段显示的属性视图控件中。

工业控制过程中需要对设备参数进行设置，这是需要向地址空间中数据源执行写入操作。写入操作是对地址空间节点属性字段更改操作，主要使用属性服务集中的 Write 服务。在客户端程序界面地址空间浏览控件区域，当右键点击时显示对节点的输入操作快捷菜单，在程序中添加点击菜单项的事件响应函数 Browse_WriteMI_Click，函数显示输入操作对话框，程序如下：

```
private void Browse_WriteMI_Click(object sender, EventArgs e)
{
    ... ...
        // 确认存在会话与已经选中节点
        if (m_session == null || BrowseNodesTV.SelectedNode == null)
        {
          return;
```

```
        }
        new WriteValueDlg().ShowDialog(m_session, (NodeId)reference.NodeId,
                                        Attributes.Value);
    ... ...
}
```

写入操作如图 4-9 的流程图

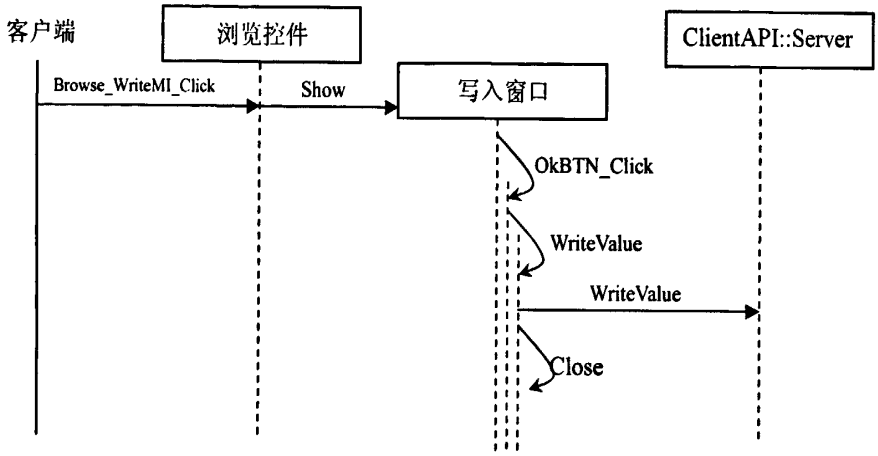

图 4-9 写入流程

```
private void OkBTN_Click(object sender, EventArgs e)
{
    ... ...
    WriteValue valueToWrite = new WriteValue();

    ... ...
    WriteValueCollection valuesToWrite = new WriteValueCollection();
    valuesToWrite.Add(valueToWrite);

    // 写入当前值
    m_session.Write(
      null,
      valuesToWrite,
      out results,
      out diagnosticInfos);
    ... ...
}
```

4.4.4 订阅数据变化或事件

客户端定义监视项订阅数据与事件，每个监视项确定监视数据项订阅则用于发送通知，且可以监视任何节点的属性。通知项是数据结构描述数据的变化及事件的发生图。监视的操作在客户端界面中使用监视功能模块，监视项的各项参数在ListView控件中显示，使用规范中定义的CreateMonitoredItems、

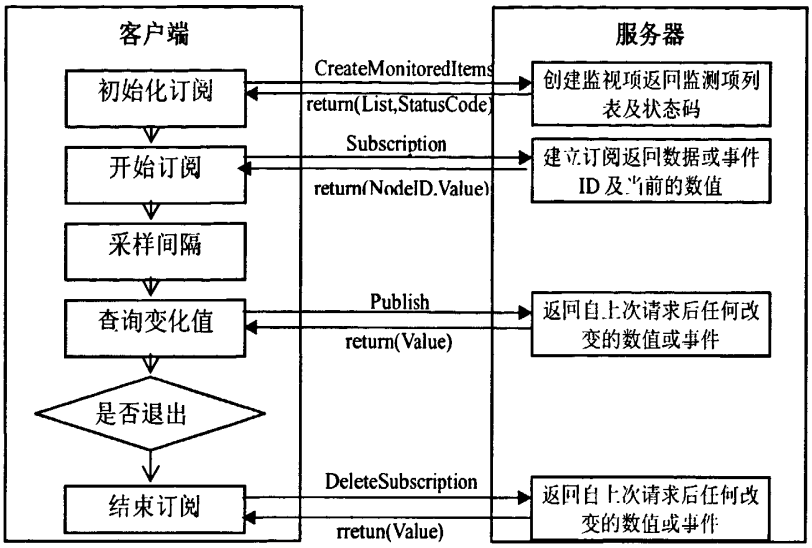

图4-10 客户端订阅流程

ModifyMonitoredItems、SetMonitoringMode、CreateSubscription、ModifySubscription、SetPublishMode等服务函数。图4-10给出了在服务器创建数据订阅的流程。

应用程序的实现定义Browse_MonitorMI_Click函数，调用m_subscription.Create()与m_subscription.AddItem(monitoredItem)实现。程序代码：

```
private void Browse_MonitorMI_Click(object sender, EventArgs e)
{
    …… 
    // 如果订阅不存在则创建订阅
    if (m_subscription == null)
    {
      m_subscription = new Subscription(m_session.DefaultSubscription);
    //设置订阅参数
      m_subscription.PublishingEnabled = true;
```

```
    m_subscription.PublishingInterval = 1000;
    …… ……

    m_session.AddSubscription(m_subscription);

    // 在服务器端创建订阅
    m_subscription.Create();
  }

  // 增加监视项
  MonitoredItem monitoredItem = new
                        MonitoredItem(m_subscription.DefaultItem);
  //设置监视项参数
  monitoredItem.StartNodeId = (NodeId)reference.NodeId;
  monitoredItem.AttributeId = Attributes.Value;
  …… ……
  monitoredItem.Notification += m_MonitoredItem_Notification;

  m_subscription.AddItem(monitoredItem);
  m_subscription.ApplyChanges();

  // 在列表视图中增加属性值
  ListViewItem item = new
            ListViewItem(monitoredItem.ClientHandle.ToString());
  monitoredItem.Handle = item;
  …… ……
  }
```

此外，在监视功能模块中添加了右键快捷菜单，完成对监视项的修改，包括删除监视项、修改监视模式、修改采样间隔与死区特性等功能。

删除监视项的将从监视功能模块中删除一条添加的监视项，功能实现代码如下：

```
private void Monitoring_DeleteMI_Click(object sender, EventArgs e)
{
    …… ……
    // 收集要删除的项目.
    List<ListViewItem> itemsToDelete = new List<ListViewItem>();
```

```
        for (int ii = 0; ii < MonitoredItemsLV.SelectedItems.Count; ii++)
        {
          MonitoredItem monitoredItem = MonitoredItemsLV.SelectedItems[ii].Tag
as MonitoredItem;

          if (monitoredItem != null)
          {
            monitoredItem.Notification -= m_MonitoredItem_Notification;
            m_subscription.RemoveItem(monitoredItem);
            itemsToDelete.Add(MonitoredItemsLV.SelectedItems[ii]);
          }
        }
        // 更新服务器.
        m_subscription.ApplyChanges();
        // 检查状态.
        for (int ii = 0; ii < itemsToDelete.Count; ii++)
        {
          MonitoredItem monitoredItem = itemsToDelete[ii].Tag as MonitoredItem;

          if (ServiceResult.IsBad(monitoredItem.Status.Error))
          {
            itemsToDelete[ii].SubItems[8].Text =
monitoredItem.Status.Error.StatusCode.ToString();
            continue;
          }

          itemsToDelete[ii].Remove();
        }
        ... ...
    }
```

修改监视模式、采样间隔、死区特性等参数分别将调用下面三个函数

```
m_subscription.SetMonitoringMode
monitoredItem.SamplingInterval
monitoredItem.Filter
```

关于具体代码就不赘述。

4.4 OPC UA 客户端测试

通过上一节中实现的代码实现了基本客户端功能，检查客户端功能需要连接服务器，在本机安装查找服务器与 OPC 基金会提供的测试服务器。下面对客户端功能进行测试，首先启动客户端软件如图 4-11

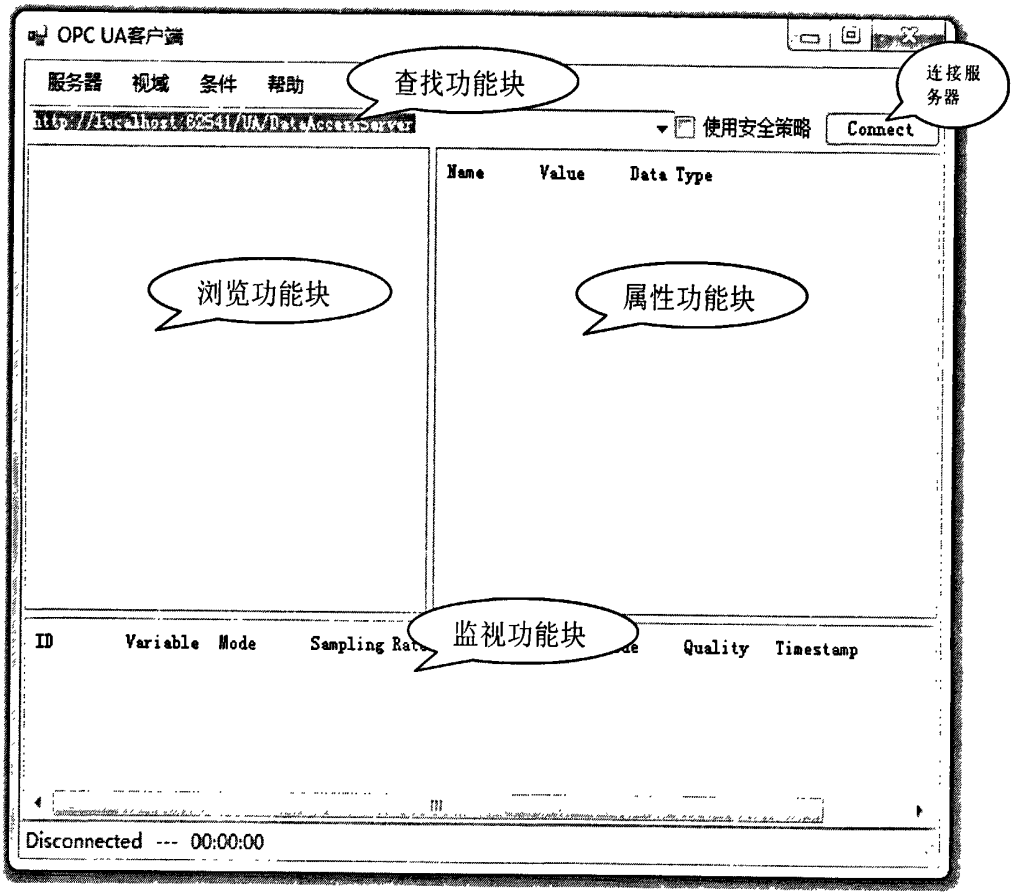

图 4-11 客户端界面

查找功能块可以查找及显示运行于本机的服务器，也可以通过手动输入地址连接非本机服务器，当点击下拉列表时如图 4-12 显示查找到的服务。

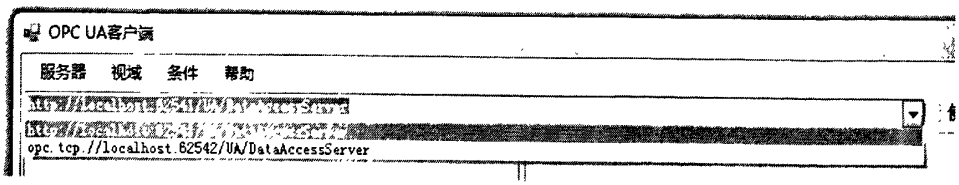

图 4-12 查找服务器实现

选择任意端点，点击 Connect 创建与该服务的连接，如果连接建立成功如图 4-13 在窗口浏览功能块显示了地址空间内容，选择地址空间中的节点将在属性功能块中显示该节点的属性。

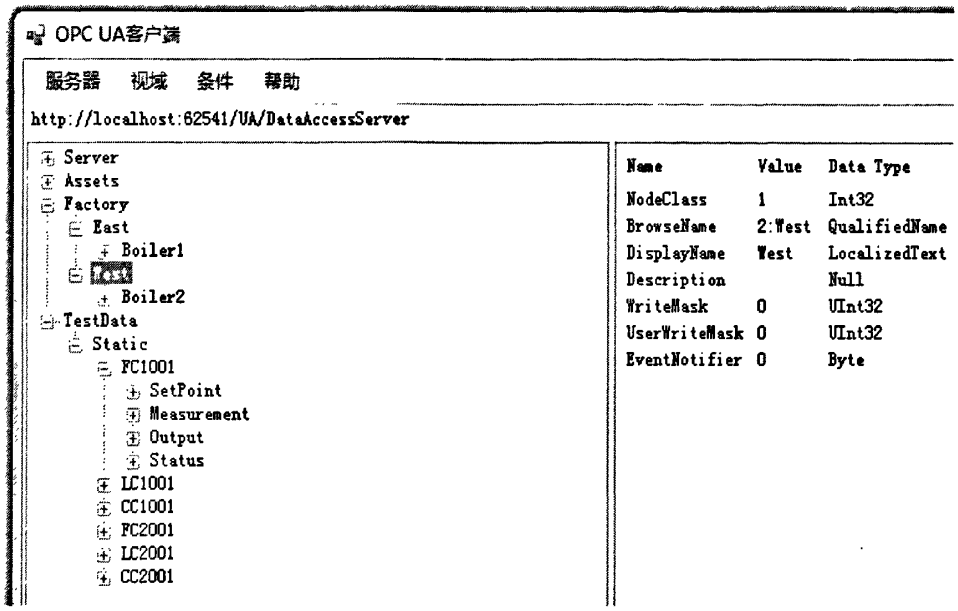

图 4-13 浏览与读操作

在浏览功能块中点击右键出现快捷菜单，可以执行添加监视项与写入操作。任意选择两项将添加到监视功能块如图 4-14

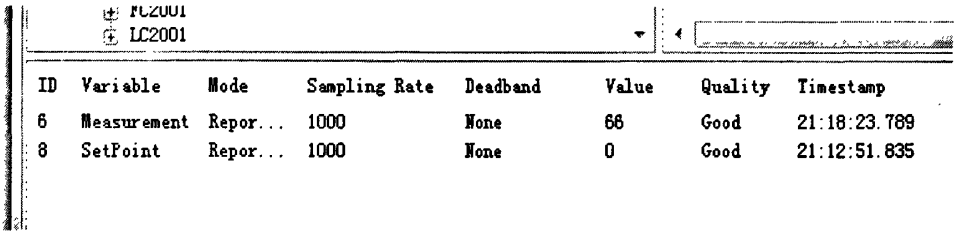

图 4-14 监视操作

写入操作如图 4-15 对显示名为 SetPoint 执行写入 22，在图 4-16 中显示更改后的值 SetPoint 的值更改为 22.

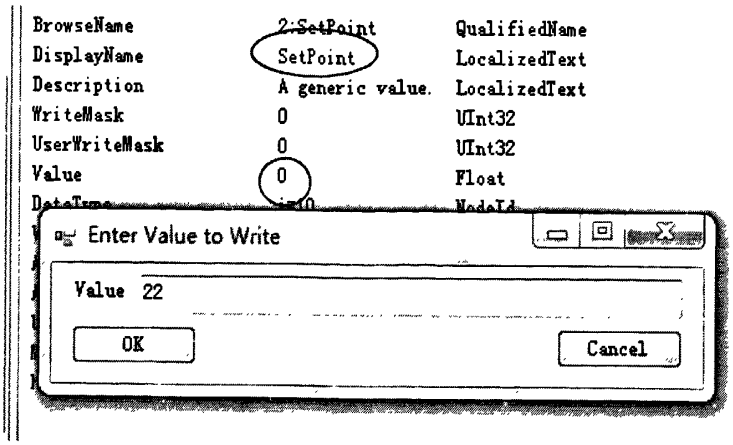

图 4-15 写入操作前

Name	Value	Data Type
NodeClass	2	Int32
BrowseName	2:SetPoint	QualifiedName
DisplayName	SetPoint	LocalizedText
Description	A generic value.	LocalizedText
WriteMask	0	UInt32
UserWriteMask	0	UInt32
Value	22	Float
DataType	i=10	NodeId
ValueRank	-1	Int32

图 4-16 写入操作后

在监视功能块中点击右键出现快捷菜单可执行删除及更改监视项参数操作，如图 4-17 及图 4-18 操作前与操作后对比。

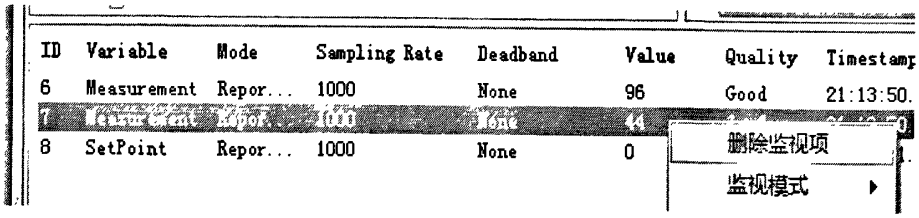

图 4-17 监视块操作

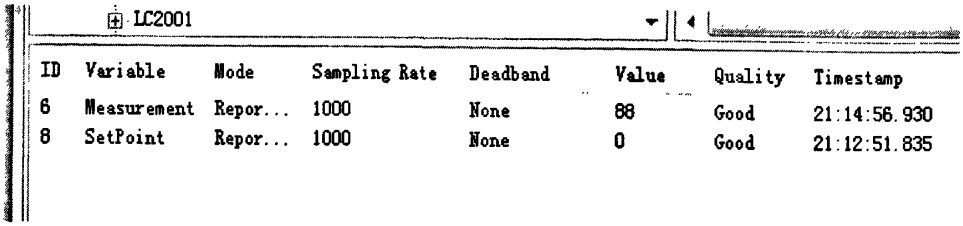

图 4-18 监视块操作后

4.5 本章小结

本章中详细介绍整个客户端程序具体实现过程，通过对各个部分功能块中实现的流程图，给出了各部分实现时主要的代码。通过连接测试服务器，检查客户端应用程序执行功能的有效性。

第5章 结束语

5.1 本文的主要工作

本文针对 OPC UA 规范的客户端做研究，对实现客户端实现需要的服务函数做简单介绍，描述了程序实现基于的.NET 架构的基础支持。主要完成的论文内容如下所述：

1、阐述了 OPC 技术的发展，到 OPC UA 技术出现所带来的变革，对 OPC UA 技术和.NET 平台思想的共通性做简单描述，通过 OPC UA 规范第四部分服务函数的描述，确定 OPC UA 客户端应该实现功能。

2、通过技术映射规范确定了客户端程序的结构，使用 Visual Studio 2008 进行客户端应用程序的开发。完成了查找服务器、连接服务器、浏览地址空间、地址空间节点数据读/写、订阅地址空间数据相应过程及功能描述，给出基于 C#语言的具体实现代码，最后通过测试服务器完成对客户端实现功能的测试。

5.2 未来与展望

随着 OPC UA 规范越来越完善必将取代基于 COM 的 OPC 规范。由于它的平台独立性和 Web 服务的使用，OPC UA 将会在更广泛的范围中应用，跨越从底层设备到 DCS、MES 和 ERP 系统。OPC UA 客户端实现对于完成工业现场数据的流动发挥很重要的作用，本文中实现了客户端的基本功能，对于规范中提到的地址空间的管理部分由于时间关系尚未实现，对于课题后续的研究工作还很多。对于客户端来说，需要与数据库进行交互，这部分内容是 OPC UA 客户端开发的下一部分内容 OPC Unified Architecture 规范在数据传输的可靠性、安全性、平台可移植性等方面具有巨大优势，越来越多的软件开发商开始关注这一领域。国内的亚控等企业也在开发基于 OPC UA 的产品。相信在不远的将来 OPC UA 将成为工控领域新的通信标准，会有越来越多支持 OPC UA 的自动化产品出现。

参考文献

[1] 李光宇, 李延新, 袁爱进.基于C/S模型的OPC客户端实现远程监控[J].微计算机信息.2007(23):25-26

[2] 王华, 刘枫. 基于FDT与OPC UA的设备集成研究[J]. 微计算机信息. 2009(25):19-21

[3] 孙发, 刘枫. 基于OPC Unified Architecture的服务器研究[J]. 仪器仪表标准化与计量. 2006, 05(05):14-17

[4] 杨明极, 郭剑虹, 沈强. OPC技术在工业控制领域的研究[J]. 哈尔滨理工大学学报. 2008(8):29-31

[5] 纪强军, 唐秀昆, 曹长修. 基于Visual C++的OPC客户端实现[J]. 自动化技术与应用. 2007, 26(4): 70-73

[6] 蔡思文, 祁耀斌. OPC客户端设计及其在监控系统的应用[J]. 微计算机信息. 2007, 23(5): 106-108

[7] 赵宴辉, 聂亚杰, 王永丽. OPC UA技术综述[J]. 舰船防化. 2010(2):33-37

[8] 张力展, 靳其兵, 赵大力. 基于OPC UA的管控一体化研究[J]. 工业控制计算机. 2008, 21(9):26-27

[9] OPC Foundation. OPC Unified Architecture Specification Part 1 [S]: Overview and Concepts 1.01. 2008:1-2

[10]OPC Foundation. OPC Unified Architecture Specification Part 4 [S]: Services 1.01. 2008:100-103

[11]张红.基于OPC UA和.NET平台的组态软件新时代 [J].PLC&FA, 2009（10）：84-86

[12]Jeffrey C.Jackson. Web 技术[M]. 清华大学出版社. 2007:10-25

[13]薛思源, 刘枫. 基于WEB服务器的控制网络集成[J]. 仪器仪表标准化与计量. 2005(2):29, 48

[14]罗江华, 朱永光. .NET Web高级开发[M]. 电子工业出版社. 2008:325-327

[15]杨传颖, 黄德先.基于Microsoft .NET Framework的OPC客户端的研究与应用[J].微计算机信息, 2006-22（06）：1-3

[16]潘爱民. COM原理与应用[M]. 北京：清华大学出版社, 1999；302-330

[17]陈在平, 彭登峰. 基于Visual Studio .NET的OPC客户端的研究与实现[J]. 制造业自动化. 2008(12):22-24

[18]杨传颖, 刘志鸿. 基于Microsoft .NET Framework的OPC客户端开发方式的研究[J]. 工业控制计算机.2007(1):31-32.

[19]OPC Foundation. OPC Unified Architecture Specification Part 1 [S]: Overview and Concepts 1.01. 2008:1-2

[20]OPC Foundation. OPC Unified Architecture Specification Part 3 [S]: Address Space Model 1.01. 2008:5-6

[21]OPC Foundation. OPC Unified Architecture Specification Part 4 [S]: Services 1.01.

2008:18-19
[22]OPC Foundation. OPC Unified Architecture Specification Part 5 [S]: Information Model 1.01. 2008:35-38
[23]OPC Foundation. OPC Unified Architecture Specification Part 6 [S]: Mappings RC 1.00.08. 2008:35-38
[24]Wolfgang Mahnke, Stefan-Helmut Leitner, Matthias Damm. OPC Unified Architecture [M]. SpringerPress, Ladenburg Germany, 2009:203-209
[25]王华, 刘枫. OPC UA技术及应用[J]. 工业控制计算机. 2008, 21(12):38-39, 42
[26]陆会明, 阎志峰.OPC UA服务器地址空间关键技术研究与开发[J].电力自动化设备, 2010-30（06）：109-113
[27]Thomas Hadlich. Providing device integration with OPC UA[C]. 2006 IEEE International Conference on Industrial Informatics. Singapore. 2006:263-268
[28]OPC Foundation. OPC Unified Architecture Specification Part 2 [S]: Security Model 1.01. 2008:1-7
[29]Annerose Braune, Stefan Hennig, Sebastian Hegler. Evaluation of OPC UA Secure Communication in Web Browser Applications [C]. The IEEE International Conference on Industrial Informatics. Korea, 2008:1660-1665
[30]Tom Hannelius. Roadmap to adopting OPC UA [C].Proceedings of IEEE Conference on Industrial Informatics. Daejeon,South Korea, 2008:756-761.
[31]Wolfgang Mahnke, Stefan-Helmut Leitner, Matthias Damm. OPC Unified Architecture[M]. Springer Press,Ladenburg Germany， 2009: 10-16.
[32]Daniel Grossmann, Prof.Klaus Bender, Benjamin Danzer. OPC UA based Field Device Integration[C]. SICE Annual Conference 2008. Japan. 2008:933-938
[33]周靖. Visual C# 2008从入门到精通[M].北京：清华大学出版社. 2009
[34]朱立军, 安娜，陈未如. 基于Visual C#的客户端实现 [J]. 现代电子技术. 2009 (2):171- 173.

攻读硕士学位期间发表的论文及其它成果

（一）发表的学术论文

[1] 李金亮. OPC UA 关键技术及其客户端开发[C]. 华北电力大学第八届研究生学术交流年会. 2010，已收录

致谢

本文是在导师陆会明教授的悉心指导下完成的。导师渊博的知识以及豁达的人生态度给我很大的教育，导师给予我不仅是严谨的治学态度，更有正确的人生观使我终生受益。在课题的研究期间，导师倾注了大量的心血，在此向辛勤培育我的导师致以崇高的敬意和衷心的祝福！课题完成期间周围的同学与其他老师也给予我很多帮助，在此谢谢给我支持与鼓励的老师与同学。

由于本人知识水平有限，文中难免出现不完善与错误之处，敬请读者提出批评和指正。

OPC UA客户端访问与测试功能研究及开发

作者：李金亮
学位授予单位：华北电力大学(北京)