目录

摘要 ..... - 3 -
ABSTRACT ..... - 4 -
第一章　绪论 ..... - 5 -
1.1 课题来源 ..... - 5 -
1.2 电子商务的现状和研究 ..... - 5 -
1.2.1 电子商务在我国的发展 ..... - 6 -
1.2.2 电子商务的问题及解决方法 ..... - 7 -
1.3 项目的目的与意义 ..... - 11 -
1.4 本文的组织结构 ..... - 11 -
第二章 相关技术综述 ..... - 12 -
2.1 技术选用理由 ..... - 12 -
2.2.1 页面编程语言的选择 ..... - 12 -
2.2.2 后台数据库的选择 ..... - 12 -
2.2.3 服务器软件和平台的选择 ..... - 13 -
2.2 ASP.NET 介绍 ..... - 13 -
2.3 C＃.NET 简介 ..... - 14 -
2.4 ADO.NET 简介 ..... - 16 -
2.5 SQL SEVER 2000 简介 ..... - 17 -
第三章 系统详细设计 ..... - 19 -
3.1 需求分析 ..... - 19 -
3.2 系统分析 ..... - 19 -
3.3 系统结构图 ..... - 20 -
3.4 网站工作流程 ..... - 21 -
3.4.1 网站图示： ..... - 21 -
3.4.2 网站模块划分： ..... - 22 -
3.4.3 网站操作及具体功能： ..... - 23 -
3.5 网站设计 ..... - 24 -
3.5.1　两层应用程序结构 ..... - 25 -
3.6 两层结构设计 ..... - 26 -
3.6.1 前台业务层设计： ..... - 26 -
3.6.2 数据存储层设计： ..... - 29 -
第四章 系统的实现 ..... - 38 -
4.1 软硬件环境 ..... - 38 -
4.2 网站的实现及其相应关键代码（图文介绍） ..... - 38 -
4.2.1 页面判断的实现 ..... - 39 -
4.2.2 存储过程的实现 ..... - 41 -
4.2.3 前台对后台的访问的实现 ..... - 42 -
4.4.4 页面间参数传递的实现 ..... - 44 -
第五章 工作总结 ..... - 47 -
5.1 设计总结 ..... - 47 -

5.2 不足之处 - 48 -
5.3 未来研究方向 - 48 -
致谢 - 51 -
参考文献 - 52 -
附录 - 53 -
英文翻译 - 53 -
中文译文 - 60 -

摘要

从上个世纪末开始，电子商务迅速崛起，在商业领域中占据越来越重要的位置，并且向传统商业模式发起了极大的冲击。随着互联网的日益推广使用，电子商务这一新兴的商务模式正不断渗透到我们的日常生活中，它的发展标志着一个高度发达的电子信息化世界正在形成。

本文论述了基于 Internet 的电子产品交易网站的设计与实现，采用具有竞争力的开发工具开发，充分体现了这种电子商务模式的优势和特点，在理论上和实践上为这种新模式的发展做了一次有意义的探索。这次设计的目的是实现一个具有完整购物流程的商务网站，该网站能提供多人在线购物，它的意义在于能紧密结合现代经济发展的潮流，使经济模式趋向于更方便快捷，也使得现代的计算机技术能融入到传统的商务中，使传统商务焕然一新，具备更强的竞争力，发展的道路更宽广。

网站采用 B/S 架构即浏览器/服务器模式开发，前台页面采用 ASP.NET 等动态网页脚本语言编写，重要交易与数据库操作模块采用微软的 Visual Studio2003 的 C#.NET 编写，后台数据库由 SQL SEVER 2000 开发，利用 ADO.NET 方式连接和调用后台数据库。在 WINXP 平台下进行该系统的开发,WEB 服务器为 IIS5.1。系统由七个模块构成,分别为前台页面模块,客户管理模块,商品订购模块,网上付款模块,物流仓储模块,商务信函模块,商品评论模块等,实现了一些简单的交易及物流功能。

论文还对本次设计做出了总结，提出一些不足和对未来研究方向的探讨。

关键词：电子商务，B/S，ASP.NET，数据库，C#.NET。

ABSTRACT

From beginning at the end of last century, e-commerce emerges rapidly, occupy the more and more important position in the commercial field, and has initiated the great impact to the traditional commercial mode. With lift network use widely day by day ,e-commerce this new developing commercial mode is permeating through our daily life constantly, its development indicates information-based world of a highly developed electron is taking shape.

This text has described the design based on electronic product trade website of Internet and realized, adopt the development of the competitive developing instrument , has fully reflected the advantage and characteristic of this kind of e-commerce mode, do one meaningful exploration for such new development of mode at the practice in theory. The purpose to design this time is to realize one has intact commercial websites that do shopping procedure, this website can offer many people to do shopping online , its meaning lies in combining the trend of modern economic development closely, make the economic mode trend to be more convenient and more swift, enable modern computer technology to incorporate in the traditional commercial affair too, make the traditional commercial affair take on an entirely new look, possess stronger competitiveness , the road of development is more broad .

Website adopt B/S it builds up to be browser / server mode development, front desk page adopts ASP.NET,etc. dynamic webpage script language write, important trade and database operate module adopt C of Visual Studio2003 of Microsoft. NET is written, backstage supporter's database is developed by SQL SEVER 2000,utilizes ADO.NET way to join and transfer backstage supporter's database . Carrying on the development of this system under WINXP platform, WEB server is IIS5. 1. The system is formed by seven pieces of module , is the page module of front desk respectively, customer manage module, goods order module, pay the bill the module on the net, logistics storage module , commercial correspondence module , the module of comment on commodity,etc., have realized some simple trade and logistics function.

The thesis has been also summarized to this design, propose some is insufficient and studying the discussion of the direction to the future.

Keywords: Electronic commerce, B/S, ASP.NET, Database, C#.NET.

第一章 绪论

1.1 课题来源

电子商务极大地改变了商务模式，带动了经济结构的变革。电子商务的发展，可以降低企业运作成本，充分把握市场信息，提供个性化的客户服务，从而使企业减少开支、提升服务水平，在竞争中获得优势。本课题要求实现一个基于 SQL Server 数据库后台的电子产品交易的商务系统，实现用户注册、商品订购、网上付款、物流管理的整个交易过程，且可以通过商务信函和论坛等方式实现与用户的长期或即时的沟通。

本课题为自拟。课题通过上述过程使得学生可以深入了解数据库开发和 B/S 结构，加强电子商务及其应用软件的构架的认识，培养自身的动手能力。

1.2 电子商务的现状和研究

电子商务是指两方或多方通过计算机和某种形式的计算机网络(直接连接的网络或 Internet 等)进行商务活动的过程。它包括企业和企业之间的商务活动, 网上的零售业和金融企业的数字化处理过程。一些专家甚至认为电子商务包括商业活动中的所有内容，从广告发布到打印发票和客户服务。电子商务主要可以概括为 E2Commerce 和 E2Business 两种，两者之间有着许多的差异。其中, E2Commerce 通常是指通过网络销售东西, 包括 B2B, B2C, C2C 的很多应用, 如网上商店等；而 E2Business 是用网络来管理业务, 主要是内部的, 比如 ERP,CRM 等电子商务的典型应用。

电子商务起源于 20 世纪 70 年代，当时一些大公司通过建立自己的计算机网络实现各个机构之间、商业伙伴之间的信息共享，这个过程被称为 EDI。(电子数据交换)。EDI 通过传递标准的数据流可以避免人为的失误，降低成本，提高效率，据估计在世界一千个最大的企业中，95%以上的在使用这一技术。它过去是、现在也仍是电子商务的基础。今天,Internet 为电子商务带来了飞速的增长。例如 EDI 技术已经摆脱了以前的旧式的昂贵的公司独立网络，而融于 Internet；而更多的企业和企业之间的商务活动则干脆直接采用 Web 技术来进行。目前电子商务概念主要涉及表现“B2C 网上购物”和 B2B 企业级电子商务。“B2C 网上购物”:通过 Web 技术将产品、服务和信息销售给顾客。“网上购物”起源于 1995 年，它的先驱是那些不进行传统零售业的 Internet 公司，如 AMA2ZON,但今天像 WAL2MART 这样的超市也建立了自己的网上商店。

B2B 企业级电子商务：电子商务更核心的是市场潜力比零售业大一个量级的企业级电子商务过程。企业级(BusinesstoBusiness)电子商务一般被简称为 B2B 的电子商务过程,它是一个将买方、卖方及服务于他们的中间商(如金融机构)之间的信息交换和交易行为集成在一起的电子运作方式。而这种技术的使用会从根本上改变企业的计划、生产、销售及运行模式，甚至改变整个产业社会的基本生存方

式。

1.2.1 电子商务在我国的发展

我国电子商务的发展始于 20 世纪 90 年代初，1997 逐渐成为一个热门话题。在短短几年的时间内，我国电子商务已从启蒙阶段迅速进入实施阶段,新的电子商务网站如网上商店，商城，专卖店，拍卖店，网上订票，旅游，教育，医疗以及各种资讯和交易站点等,如雨后春笋般不断涌现出来.发展区域也从北京，上海，广州等少数城市向沿海和内地各大城市扩展，许多企事业单位也已开始登上电子商务的舞台。但总体来说，我国电子商务发展的总体水平还比较低，发展中还存在着诸多问题。

电子商务在全球范围内正以出乎人们预料的速度迅猛发展。根据西方一些调查机构的统计，发达国家有 80%以上的企业利用互联网宣传推销自己的产品。1999 年 8848 网上超市出现后，我国的电子商务开始步入快速发展，2000 年掀起了以.com 公司为代表的第一次发展高潮，在经历了 2001 年的调整以后，从 2002 年开始进入与传统产业相结合的务实发展阶段。

（1）我国政府对发展电子商务给予了高度的重视。“十五”计划纲要明确提出要“加快认证体系、现代支付系统和信用制度建设，大力发展电子商务”。

（2）北京、上海、广东三省市电子商务发展速度最快。国内一半以上的电子商务网站聚集在这三个省市。上海市的电子商务在各个方面都在全国前列，一方面投入了大量的资金用于配套设施建设，另一方面，从税收政策、安全保障、知识产权保护、电子技术标准等方面，抓紧研究制定必要的法规规章，为电子商务的发展创造了一个良好的外部环境。政府投资 2 亿多元建设了全国最大最好的 CA 认证中心工业企业已经建立了数个大型行业平台。

（3）河北省电子商务工作步入快车道。河北省委、省政府对电子商务工作十分重视，2001 年即成立了电子商务工作协调小组，并成立了河北省电子商务研究会作为全省电子商务工作的研究咨询机构。根据抽样调查的结果，截止到 2001 年底，全省有 80%的大企业和 30%的中小企业以各种方式接入了互联网，有 54%的大企业和 21%的中小企业建立了企业网站。

（4）电子商务对传统行业的影响。电子商务是在商务活动的全过程中，通过人与电子通讯方式的结合，极大地提高商务活动的效率，减少不必要的中间环节。传统的制造业籍此进入小批量、多品种的时代，“零库存”成为可能；传统的零售业和批发业开创了“无店铺”“网上营销”的新模式；各种线上服务为传统服务业提供了全新的服务方式。

（5）电子商务对金融业的影响。随着电子商务在电子交易环节上的突破，网上银行、银行电子支付系统以及网上服务、电子支票、电子现金等服务，将传统的金融业带入一个全新的领域。1995 年 10 月，全球第一家网上银行“安全第一网络银行”（Security First Network Bank）在美国诞生，这家银行没有建筑物，没有地址，营业厅就是首页画面，员工只有 10 人，与总资产超过 2000 亿美元的美国花旗银行相比，“安全第一网络银行”简直是微不足道，但与花旗银行不同的是，该银行所有交易都通过互联网进行。

1.2.2 电子商务的问题及解决方法

电子商务的问题：

电子商务的运作,涉及多方面的安全问题,资金安全，信息安全，货物安全，商业秘密等等″安全问题如果不能妥善解决,电子商务的实现就是一句空话。许多用户不愿进行网上交易,也是因为对网上交易的安全性,可靠性持怀疑态度。电子商务的安全问题,不仅涉及技术问题,同时也涉及管理问题和法律问题″我国目前尚不能批量生产自己的网络防火墙,许多银行电子储蓄系统的密码仅有 4 位数字,技术防范措施显然不能适应大规模电子交易的要求。电子商务交易的管理标准还没有系统制定,法律对电子商务违法交易行为的认定尚处于摸索阶段,这些问题都需要组织专门力量迅速加以解决。

在开展电子商务中,信息流的重要性毋庸质疑但仅仅停留在虚拟空间是远远不够的，还要落实到现实空间中来，即要有发达!完善的物流配送体系作为支撑。在中国,电子商务概念先于电子商务实践电子商务技术正在艰难地寻找商务需求,电子商务应用基础和环境还远远不够完善，尤其是物流配送体系并不发达，这就要求我们在开展电子商务过程中，不仅要注重”网上”信息资源开发，更要重视”网下”物流建设问题。

电子商务的进行包括资金的支付与结算,将来要在网络上直接进行交易,就需要通过银行的信用卡等各种方式来完成交易,以及在国际贸易中通过与金融网络的连接来支付″而目前我国专业银行网络选用的通信平台不统一,不利于各银行间跨行业务的互联!互通和中央银行的金融监管。

我国前一段实行的电信体制也限制了 Internet 的发展″集行政管理与经营于一身的垄断体制,严重打击了其他 ISP 的业务,服务质量和水平不能保证,价格又降不下来,把大量的潜在的用户拒之于外″普通市话的成本只有每分钟 0.03-0.05 元,而实际收费却高出数倍″按绝对货币价格计算,中国人要比美国人高 20 倍的价格才能买回同样大小的信息流量,而我们的人均收入不过美国的 1/20。这种体制不改变,电子商务的普及就很难实现。

供需双方的网上交易,相互信任是成交的根本保证。不能排除一些人看到网上广告立即通过网上付款购买。但对大多数顾客来说,购买前的认真判断是必须进行的程序″面对信用程度较低，三角债情况严重，假冒伪劣商品泛滥的现实商业环境，对电子商务望而却步是极为自然的。

电子商务的一个重要技术特征是利用 IT 技术来传输和处理商业信息。因此，电子商务安全从整体上可分为两大部分：计算机网络安全和商务交易安全。

- 计算机网络安全的内容包括：计算机网络设备安全、计算机网络系统安全、数据库安全等。其特征是针对计算机网络本身可能存在的安全问题，实施网络安全增强方案，以保证计算机网络自身的安全性为目标。
- 商务交易安全则紧紧围绕传统商务在互联网络上应用时产生的各种安全问题，在计算机网络安全的基础上，如何保障电子商务过程的顺利进行。即实现电子商务的保密性、完整性、可鉴别性、不可伪造性和不可抵赖性。

计算机网络安全与商务交易安全实际上是密不可分的，两者相辅相成，缺一不可。没有计算机网络安全作为基础，商务交易安全就犹如空中楼阁，无从谈起。没有商务交易安全保障，即使计算机网络本身再安全，仍然无法达到电子商务所特有的安全要求。

计算机网络安全体系：

一个全方位的计算机网络安全体系结构包含网络的物理安全、访问控制安全、系统安全、用户安全、信息加密、安全传输和管理安全等。充分利用各种先进的主机安全技术、身份认证技术、访问控制技术、密码技术、防火墙技术、安全审计技术、安全管理技术、系统漏洞检测技术、黑客跟踪技术，在攻击者和受保护的资源间建立多道严密的安全防线，极大地增加了恶意攻击的难度，并增加了审核信息的数量，利用这些审核信息可以跟踪入侵者。

在实施网络安全防范措施时：

- 首先要加强主机本身的安全，做好安全配置，及时安装安全补丁程序，减少漏洞；
- 其次要用各种系统漏洞检测软件定期对网络系统进行扫描分析，找出可能存在的安全隐患，并及时加以修补；
- 从路由器到用户各级建立完善的访问控制措施，安装防火墙，加强授权管理和认证；
- 利用 RAID5 等数据存储技术加强数据备份和恢复措施；
- 对敏感的设备和数据要建立必要的物理或逻辑隔离措施；
- 对在公共网络上传输的敏感信息要进行强度的数据加密；
- 安装防病毒软件，加强内部网的整体防病毒措施；
- 建立详细的安全审计日志，以便检测并跟踪入侵攻击等。

网络安全技术是伴随着网络的诞生而出现的，但直到 80 年代末才引起关注，90 年代在国外获得了飞速的发展。近几年频繁出现的安全事故引起了各国计算机安全界的高度重视，计算机网络安全技术也因此出现了日新月异的变化。安全核心系统、VPN 安全隧道、身份认证、网络底层数据加密和网络入侵主动监测等越来越高深复杂的安全技术极大地从不同层次加强了计算机网络的整体安全性。安全核心系统在实现一个完整或较完整的安全体系的同时也能与传统网络协议保持一致。它以密码核心系统为基础，支持不同类型的安全硬件产品，屏蔽安全硬件以变化对上层应用的影响，实现多种网络安全协议，并在此之上提供各种安全的计算机网络应用。

电子商务交易中的安全措施：

在早期的电子交易中，曾采用过一些简易的安全措施，包括：

- 部分告知（Partial Order）：即在网上交易中将最关键的数据如信用卡号码及成交数额等略去，然后再用电话告之，以防泄密。
- 另行确认（Order Confirmation）：即当在网上传输交易信息后，再用电子邮件对交易做确认，才认为有效。

此外还有其它一些方法，这些方法均有一定的局限性，且操作麻烦，不能实现真正的安全可靠性。

近年来，针对电子交易安全的要求，IT 业界与金融行业一起，推出不少有效的安全交易标准和技术。

主要的协议标准有：

- 安全超文本传输协议（S－HTTP）：依靠密钥对的加密，保障 Web 站点间的交易信息传输的安全性。
- 安全套接层协议（SSL）：由 Netscape 公司提出的安全交易协议，提供加密、认证服务和报文的完整性。SSL 被用于 Netscape Communicator 和 Microsoft IE 浏览器，以完成需要的安全交易操作。

- 安全交易技术协议(STT,Secure Transaction Technology):由 Microsoft 公司提出，STT 将认证和解密在浏览器中分离开，用以提高安全控制能力。Microsoft 在 Internet Explorer 中采用这一技术。
- 安全电子交易协议（SET，Secure Electronic Transaction）

1996 年 6 月，由 IBM、MasterCard International、Visa International、Microsoft、Netscape、GTE、VeriSign、SAIC、Terisa 就共同制定的标准 SET 发布公告，并于 1997 年 5 月底发布了 SET Specification Version 1.0，它涵盖了信用卡在电子商务交易中的交易协定、信息保密、资料完整及数据认证、数据签名等。

SET 2.0 预计今年发布，它增加了一些附加的交易要求。这个版本是向后兼容的，因此符合 SET 1.0 的软件并不必要跟着升级，除非它需要新的交易要求。SET 规范明确的主要目标是保障付款安全，确定应用之互通性，并使全球市场接受。

所有这些安全交易标准中，SET 标准以推广利用信用卡支付网上交易，而广受各界瞩目，它将成为网上交易安全通信协议的工业标准，有望进一步推动 Internet 电子商务市场。

主要的安全技术有：

虚拟专用网（VPN）

这是用于 Internet 交易的一种专用网络，它可以在两个系统之间建立安全的信道（或隧道），用于电子数据交换（EDI）。它与信用卡交易和客户发送订单交易不同，因为在 VPN 中，双方的数据通信量要大得多，而且通信的双方彼此都很熟悉。这意味着可以使用复杂的专用加密和认证技术，只要通信的双方默认即可，没有必要为所有的 VPN 进行统一的加密和认证。现有的或正在开发的数据隧道系统可以进一步增加 VPN 的安全性，因而能够保证数据的保密性和可用性。

数字认证

数字认证可用电子方式证明信息发送者和接收者的身份、文件的完整性（如一个发票未被修改过），甚至数据媒体的有效性（如录音、照片等）。随着商家在电子商务中越来越多地使用加密技术，人们都希望有一个可信的第三方，以便对有关数据进行数字认证。

目前，数字认证一般都通过单向 Hash 函数来实现，它可以验证交易双方数据的完整性，Java JDK1.1 也能够支持几种单向 Hash 算法。另外，S/MIME 协议已经有了很大的进展，可以被集成到产品中，以便用户能够对通过 E　mail 发送的信息进行签名和认证。同时，商家也可以使用 PGP（Pretty Good Privacy）技术，它允许利用可信的第三方对密钥进行控制。可见，数字认证技术将具有广阔的应用前景，它将直接影响电子商务的发展。

加密技术

保证电子商务安全的最重要的一点就是使用加密技术对敏感的信息进行加密。现在，一些专用密钥加密（如 3DES、IDEA、RC4 和 RC5）和公钥加密（如 RSA、SEEK、PGP 和 EU）可用来保证电子商务的保密性、完整性、真实性和非否认服务。然而，这些技术的广泛使用却不是一件容易的事情。

密码学界有一句名言：加密技术本身都很优秀，但是它们实现起来却往往很不理想。现在虽然有多种加密标准，但人们真正需要的是针对企业环境开发的标准加密系统。加密技术的多样化为人们提供了更多的选择余地，但也同时带来了一个兼容性问题，不同的商家可能会采用不同的标准。另外，加密技术向来是由

国家控制的，例如 SSL 的出口受到美国国家安全局（NSA）的限制。目前，美国的商家一般都可以使用 128 位的 SSL，但美国只允许加密密钥为 40 位以下的算法出口。虽然 40 位的 SSL 也具有一定的加密强度，但它的安全系数显然比 128 位的 SSL 要低得多。据报载，最近美国加州已经有人成功地破译了 40 位的 SSL，这已引起了人们的广泛关注。美国以外的国家很难真正在电子商务中充分利用 SSL，这不能不说是一种遗憾。上海市电子商务安全证书管理中心推出 128 位 SSL 的算法，弥补国内的空缺，并采用数字签名等技术确保电子商务的安全。

电子商务认证中心（CA，Certificate Authority）

实行网上安全支付是顺利开展电子商务的前提，建立安全的认证中心（CA）则是电子商务的中心环节。建立 CA 的目的是加强数字证书和密钥的管理工作，增强网上交易各方的相互信任，提高网上购物和网上交易的安全，控制交易的风险，从而推动电子商务的发展。

为了推动电子商务的发展，首先是要确定网上参与交易的各方（例如持卡消费户、商户、收单银行的支付网关等）的身份，相应的数字证书（DC：Digital Certificate）就是代表他们身份的，数字证书是由权威的、公正的认证机构管理的。各级认证机构按照根认证中心（Root CA）、品牌认证中心（Brand CA）以及持卡人、商户或收单银行(Acquirer)的支付网关认证中心(Holder Card CA，Merchant CA 或 Payment Gateway CA）由上而下按层次结构建立的。

电子商务安全认证中心(CA)的基本功能是：

- 生成和保管符合安全认证协议要求的公共和私有密钥、数字证书及其数字签名。
- 对数字证书和数字签名进行验证。
- 对数字证书进行管理，重点是证书的撤消管理，同时追求实施自动管理(非手工管理）。
- 建立应用接口，特别是支付接口。CA 是否具有支付接口是能否支持电子商务的关键。

第一代 CA 是由 SETCO 公司（由 Visa ＆ MasterCard 组建）建立的，以 SET 协议为基础，服务于 B　C 电子商务模式的层次性结构。

由于 B　B 电子商务模式的发展，要求 CA 的支付接口能够兼容支持 B　B 与 B　C 的模式，即同时支持网上购物、网上银行、网上交易与供应链管理等职能，要求安全认证协议透明、简单、成熟（即标准化），这样就产生了以公钥基础设施（PKI）为技术基础的平面与层次结构混合型的第二代 CA 体系。

近年来，PKI 技术无论在理论上还是应用上以及开发各种配套产品上，都已经走向成熟，以 PKI 技术为基础的一系列相应的安全标准已经由 Internet 特别工作组（IETF）、国际标准化组织（ISO）和国际电信联盟（ITU）等国际权威机构批准颁发实施。

建立在 PKI 技术基础上的第二代安全认证体系与支付应用接口所使用的主要标准有：

由 Internet 特别工作组颁发的标准：LDAP（轻型目录访问协议）、S/MIME（安全电子邮件协议）、TLC（传输层安全套接层传输协议)、CAT（通用认证技术，Common Authentication Technology）和 GSS－API（通用安全服务接口）等。

由国际标准化组织（ISO）或国际电信联盟（ITU）批准颁发的标准为 9594－8/X.509（数字证书格式标准）。

加强电子商务安全技术的研究和标准的制订：

要增强人们对电子商务的信任度，除了法律的制订外，还必须加强计算机的安全技术。有关部门应组织一支精干的安全技术研究队伍，集中力量尽快解决电子商务的安全技术问题。攻击等分中心，通过各种安全控制分中心的协调用，将电子商务交易风险降低到最小限度。

1.3 项目的目的与意义

该项目的目的是实现一个基于 Internet 的电子商务网站，它可以为顾客提供注册，购物，付款，查询商品等多方面的服务。可以让顾客享受一个快速的方便的购物流程。项目的意义在于对现代化科技注入传统商业模式的一次尝试，体验两者结合所带来的巨大受益。在这种网站的实际应用中，电子商务能为企业带来如下益处：降低成本，增加销售；提高工作效率；扩展市场范围；与客户良好沟通；提供全天候的服务；为顾客提供个性化服务。

1.4 本文的组织结构

第一章绪论主要论述了本课题的来源，背景，意义以及发展现状和发展中遇到的一些问题，并对问题进行了分类和分析，接着是列举了本文的组织结构；

第二章主要是对设计中所用到的相关技术进行一些介绍和论述，同时也从中表明了这些技术的优势及选用它们的原因；

第三章主要是需求分析和网站的工作流程（用图加文字说明），网站的模块划分，各个模块的功能说明及其实现的方法，以及对后台数据库的设计说明；

第四章主要讲述了网站实现的软硬件环境（软硬件配置），与网站各种功能相关的实现代码；

第五章主要是对此次设计的工作总结，论述一些没有实现的技术以及探讨一下该设计的未来研究方向。

第二章 相关技术综述

2.1 技术选用理由

在这次电子产品交易网的设计中，我采用了 ASP.NET 搭建前台页面，用 C＃.NET 编写后台代码，以及使用 SQL SEVER 2000 制作后台数据库。使用 ADO.NET 技术页面和数据库进行连接，并对其进行操作。

2.2.1 页面编程语言的选择

本次设计采用 asp.net 进行开发而非 asp 和 jsp，原因是在开发效率上 ASP.NET 觉得比 JSP 快，而在运行效率两者没有太大差距。对于 asp 来说，它属于一种解释型的编程框架，它的核心是 vbs 和 js，受这两种脚本语言的限制，决定了 asp 先天不足，它无法进行象传统编程语言那样的底层操作，所以如果你需要进行一些诸如 socket、文件等的操作时不得不借助于用其他传统编程语言如 C++、VB 、JAVA 等编写的组件，并且由于它是解释执行的，所以在运行效率上大打折扣。对于 ASP.NET 来说，它是一种编译型的编程框架，它的核心是 NGWS runtime ，除了和 asp 一样可以采用 vbs 和 js 作为编程语言外，还可以用 VB 和 C# 来编写，这就决定了它功能的强大，可以进行很多低层操作而不必借助于其他编程语言。而且它的执行效率也高于 asp。

C#和 JAVA 都是功能强大的新一代开发语言，而 Windows 是目前占垄断地位的平台，C#在 Windows 平台上的相对于 Java 要有优势，而 Java 更适合于在 Linux 和 Unix 上开发。C＃的优势就在于开发 B/S 模式的软件。C＃是像 VB 一样简单，像 C++一样强大的新语言。相对于 C++，用 C# 开发应用软件可以大大缩短开发周期。C#是第一流的面向组件的语言，是开发强壮和可重用的软件，所有的.NET Framework 中的基类库（Base Class Library）都由 C# 编写。C# 具有但 Visual Basic 不具有的特性 指针, 移位操作符, 内嵌的文档(XML)重载操作符。而且比起 VB 来，C＃更接近于 C＋＋。

2.2.2 后台数据库的选择

后台数据库有 MS Access,MySQL,Oracle,SQL sever 可供 xuanze。最终选用 SQL SEVER 2000 作为后台数据库的原因是 SQL Server 可以同时服务于许多用户，如果你希望你的站点有较高的访问率，MS Access 是不能胜任的。对于企业级用户来说 MySQL 没有子查询。对于复杂的查询，MySQL 用户必须执行两次或更多的系列查询，每一次都需要在应用和数据库间进行进程间通信或网络通信。这显著地降低了 MySQL 的速度优势。MySQL 没有存储过程，没有触发器或外键约束，只有表级锁定，所以它不适合本次设计。而 Oracle 相对 SQL Server 将占用更多的资源，而对于此次设计，SQL Server 2000 足够完成所需的工作，而且节省了系统资源。

2.2.3 服务器软件和平台的选择

在平台的选择上，我选择了相对 Linux 和 Unix 来说使用更广泛的 Windows 平台。在外界的测试研究下 Windows XP + IIS + ASP.NET + SQL Server 2000 这个组合在开发成本上应该是最低的，因为它拥有最为强大的桌面集成环境，而且这套软件微软打包后价格不贵，支持的功能又特别多，特别适合中小型企业构造企业网站、办公网、电子商务平台等。这个组合的最大风险来自 Windows 的病毒引起的维护成本，因为攻击 Windows 的病毒实在太多，Windows 系统的用户必须相当小心才能使自己的平台不受攻击。

2.2 ASP.NET 介绍

ASP.NET 是目前流行的一种动态网页开发技术。它是 ASP 的升级版本.NET 框架的一部分。在 ASP.NET 框架下，可以采用 VB.NET,C#，JAVASCRIPT.NET 等语言。ASP.NET 第二次访问页面时间比 ASP 快，效率增强。ASP.NET 增添了许多过去没有的功能强大的 WEB 控件，使得程序开发更简单。ASP.NET 将程序代码与 HTML 代码分开，程序结构清晰。

Asp.net 提供了几个超越以前 web 开发模式的优点：

• 增强的性能。Asp.net 是运行在服务器端的编译后的 CLR 代码，而不是像 ASP 那样解释执行。Asp.net 利用提前绑定，即时编译，本地优化和缓存服务来提高性能。所有这一切，性能远远大于你以往写的每一行代码。

• 世界级水平的开发工具支持。在 Visual Studio .net 的集成开发环境（IDE）中，Asp.net 框架由丰富的工具箱和设计器组成。所见即所得的（WYSIWYG）编辑方式、拖放服务器控件、以及自动部署，仅仅是这一强大工具所提供的一少部分功能。

• 强大而富有弹性。由于 asp.net 是基于(CLR)的，因此整个.net 平台的强大和富有弹性，同样可以应用于 web 应用程序开发者。.net 框架的类库、消息以及数据访问解决方案，都可以无缝集成到 web。Asp.net 也是语言中立的，因此你可以选择你最熟悉的语言，或者通过几种语言来共同完成一个应用。而且，CLR 的互用性可以保证你升级到 asp.net 的时候，现存的基于 COM 的开发投资依然保留。

• 简单。Asp.net 使执行常用的工作变得很容易，比如从简单的表单提交、客户端验证，到部署和站点配置。例如，asp.net 允许你建立用户接口，实现页面和逻辑代码的分离，同时，就像 vb 的表单执行模式那样来处理事件（也就是说，由页面驱动模式变成了事件驱动模式）。此外，CLR 简化了部署，用来管理代码服务，例如自动参照和垃圾回收。

• 易于管理。Asp.net 使用一个基于文本的、分层次的配置系统，它简化了服务器端环境和 web 应用程序的设置。由于配置信息用纯文本格式保存，新的设置不需要本地管理工具的支持。这种“零本地支持”的理念也同样应用到了部署 asp.net 应用程序。Asp.net 应用程序部署到服务器，简化为复制必要的文件到服务器。在部署，甚至替换正在运行的变异代码的时候，也不需要重新启动服务

器。

• 可伸缩性和有效利用性。Asp.net 被设计成可伸缩的，能为集群和多处理器环境设计提高性能。而且，asp.net 运行时刻密切监视和管理进程，这样，如果发生了错误的行为，例如漏洞和死锁，新的进程会在当前位置建立，来帮助你的应用程序继续处理事件请求。

• 可订制和扩展。Asp.net 提供一种良好的扩充结构，允许开发者在适当的级别“插入”他们的代码。事实上，使用自己的编写的组件可以扩展或者替换 asp.net 运行时刻的任何子组件。执行自定义验证或状态服务变得前所未有的容易。

本次开发是使用到了 ASP.NET 中的 Web services 技术。Web service 提供了在不同体系机构下构建的网站之间相互提供应用接口服务、数据的一种方案。它采用通用的 SOAP、HTTP 以及 XML，就可以把原本互不相干的站点服务形成一整套分布的、自动化和智能化的网络应用，大大减轻了程序员的开发工作量，充分地利用了已经拥有的网络资源和开发资源。在 asp.net 中，web service 文件后缀名采用 .asmx，开始应使用 <%@ WebService …%> 申明，接着引用 System.Web.Services 命名空间，然后定义一个公用类继承自 WebService 基类，最后实现自己的类，其中向网络开放的功能，应在其方法前面加上 WebMethod 属性。

2.3 C＃.NET 简介

C#是一种先进，面向对象的语言，通过 C#可以让开发人员快速的建立大范围的基于 MS 网络平台的应用，并且提供大量的开发工具和服务帮助开发人员开发基于计算和通信的各种应用。

由于 C#是一种面向对象的开发语言，所以 C#可以大范围的适用于高层商业应用和底层系统的开发。即使是通过简单的 C#构造也可以各种组件方便的转变为基于 WEB 的应用，并且能够通过 Internet 被各种系统或是其他开发语言所开发的应用调用。

即使抛开上面所提到的优点，C#也可以为 C/C++开发人员提供快速的开发手段而不需要牺牲任何 C/C++语言的特点/优点。从继承角度来看，C#在更高层次上重新实现了 C/C++，熟悉 C/C++开发的人员可以很快的转变为 C#开发人员。

开发效率与安全性

目前的各种基于 WEB 应用的软件开发向传统的商业应用软件开发提出了挑战，开发者被组织起来开发具有更短开发周期的各种应用，并且需要能够提供更好的可修正性，而不是建立一个可以长久使用的软件系统。

C#的设计正是充分考虑了这些因素。C#会帮助开发者通过更少的代码完成相同的功能，并且能够更好的避免错误发生。

与 WEB 开发相结合

新的开发模式意味着需要更好的利用现有的各种 WEB 标准，例如 HTML，XML，SOAP（简单对象存取协议）。现存的开发工具是在 Internet 出现前或是未得到充分应用前出现的，所以都不能很好的适应目前 WEB 技术的开发需要。

C#开发者可以方便的在 MS 网络平台上扩展自己的应用。C#可以将任何组件转变为 WEB 服务，并且可以被运行于 Internet 上的任何平台的任何应用调用，

重要的是 C#对这一特性提供了内置的支持。

更重要的一点，WEB 服务框架可以让任何 WEB 服务都看起来类似于 C#的内置对象，所以可以让开发人员在开发过程中继续使用他们已经具备的面向对象的开发方法和技巧。

此外 C#还拥有许多其他特性使自己成为最出色的 Internet 开发工具。例如，XML 目前已经成为网络中数据结构传送的标准，为了提高效率 C#将允许直接将 XML 数据映射成为结构。这样的话可以有效的处理各种数据。

减小开发中的错误

即使是优秀的 C/C++开发人员都难于避免在编码过程出现一些常见错误，比如错误的初始化一个变量，而这种错误将有可能导致各种不可以预知的错误，并且难于被发现。如果一旦错误在发现前被投入生产环境，排除这些错误将会付出昂贵的代价。而 C#的先进设计思想可以消除 C/C++开发中的许多常见错误，比如：

垃圾收集机制将减轻开发人员对内存的管理负担。

C#中的变量将自动根据环境被初始化。

变量是类型安全的。

使用 C#将会使开发人员更加轻易的开发和维护各种商业应用。

提供内置的版本支持来减少开发费用

更新软件系统中的组件（模块）将会是一种容易产生错误的工作，在代码修改过程中可能对现存的软件产生影响。为了帮助开发人员处理这些问题，C#在语言中内置了版本控制功能。例如：函数重载必须被显式的声明（这种情况在 C++和 JAVA 中时常发生），这可以防止代码级错误和保留版本化的特性。另一个相关的特性是接口和接口继承的支持。这些特性可以保证复杂的软件可以被方便的开发和升级。

总结起来，这些特性可以帮助开发更强壮的软件后继版本和减轻开发费用。

功能强，易于表现，灵活

更好的结合商业应用中的流程与软件实现

为了更好实现公司的各种商业计划，在软件系统中必须在商业流程和软件实现间有紧密的联系。但是大多数的开发语言都不能轻易的将各种应用逻辑与代码相联系。例如，开发人员会使用各种注释来标明各种类所代表抽象商业对象。C#允许使用在任何对象上使用预定义数据或是经过扩展的元数据。在系统结构中可以使用区域属性（译者：类似 NT 的网络域结构），并且将这些属性添加到类，接口或者其他元素上。开发者可以独立的测试各种元素上的属性。这将会使得一些如同收集区域中对象属性，或是编写自动工具来保证的区域中的类，接口是否被正确定义的类似工作变得简单。

可扩展的协作能力

虽然管理性强，透明型好，类型安全的开发环境对大多的商业应用都适合，但现实的经验告诉我们一些应用出于执行效率或是与现存的应用接口 API 相结合的原因需要使用原有的开发方式来进行编码。也正是如此，许多 C/C++开发人员宁愿放弃使用一些可以提高开发效率的开发工具。C#通过下面的方法来解决这些问题：

内置支持 COM 模型和 Windows 平台 API。

允许有限制的使用指针。

在 C#中任何对象都会自动成为 COM 对象，开发者不再需要显式的实现

IUnknown 和其他一些 COM 接口，同时也可以方便而自然的使用现存的 COM 对象，而不需要关心这些 COM 对象是否使用 C#开发。

对于使用 C#的开发人员来讲，C#允许开发人员调用 OS 所提供的 API。在经过标记的代码区域内使用指针并手工管理内存分配。这可以让 C/C++开发人员更快的熟悉和转向 C#和并且不需要放弃在以前开发中所形成的开发习惯，而且以前的 C/C++代码依然可以被重用。无论是对于 COM 的支持还是对于 API 调用的支持都是为了为开发人员提供足够的开发控制能力。

在本次设计中我用到了 C#.NET 的 WEB 开发技术，以及它对于数据库的一些操作技术。

2.4 ADO.NET 简介

ADO（ActiveX Data Object）对象是继 ODBC（Open Database Connectivity，开放数据库连接架构。微软所制定的架构，可以让透过这种架构和数据库连结。）之后微软主推存取数据的最新技术，ADO 对象是程序开发平台用来和 OLE DB 沟通的媒介，ADO 目前的最新版本为 ADO.NET。ADO.NET 不像以前的 ADO 版本是站在为了存取数据库的观点而设计的，ADO.NET 是为了因应广泛的数据控制而设计，所以使用起来比以前的 ADO 更灵活有弹性，也提供了更多的功能。ADO.NET 的出现并不是要来取代 ADO，而是要提供更有效率的数据存取。微软透过最新的.NET 技术提供了可以满足众多需求的架构，这个架构就是.NET 共享对象类别库。这个共享对象类别库不但涵盖了 Windows API（Windows Application Programming Interface，Windows 应用程序设计界面。提供许多撰写 Windows 程序所需要使用的对象以及基本函式等。）的所有功能，并且还提供更多的功能及技术；另外它还将以前放在不同 COM 组件上，我们常常使用的对象及功能一并含括进来。除此之外 ADO.NET 还将 XML 整合进来，这样一来数据的交换就变的非常轻松容易了。所以 ADO.NET 的架构及新功能是为了能满足广泛的数据交换需求所产生出来的新技术，这个就是 ADO.NET。

ADO.NET 对象可以让我们快速简单的来存取各种数据。传统的主从式应用程序在执行时，都会保持和数据源的联机。但是在某些状况下和数据库一直保持联机是不需要的，而且一直保持和数据源的联机会浪费系统资源。有些时候我们只需要很单纯的将数据取回，这时候就不需要保持对数据源的联机。ADO.NET 被设计成对于数据处理不一直保持联机的架构，应用程序只有在要取得数据或是更新数据的时候才对数据源进行联机的工作，所以应用程序所要管理的连结减少；数据源就不用一直和应用程序保持联机，负载减轻了效能自然也就提升。不过我们的应用程序也有些情况需要和数据源一直保持联机，例如在线订位系统；此时我们还是可以使用 ADO 对象和数据源随时保持联机的状态。

ADO.NET 对象模型中有五个主要的组件，分别是 Connection 对象、Command 对象、DataSetCommand、DataSet 以及 DataReader。这些组件中负责建立联机和数据操作的部分我们称为数据操作组件（Managed Providers），分别由 Connection 对象、Command 对象、DataSetCommand 对象以及 DataReader 对象所组成。数据操作组件最主要是当作 DataSet 对象以及数据源之间的桥梁，负责将数据源中的数据取出后植入 DataSet 对象中，以及将数据存回数据源的工作。

在本次设计中我大量运用了 ADO.NET 中的 SqlCoammand 命令，以及数据与

DataGrid 控件的绑定。

2.5 SQL SEVER 2000 简介

SQL Server 2000 是创建大型商业应用的最佳的核心引擎数据库之一。2003 年发布的 64 位的 SQL Server 企业版运行在惠普安腾服务器上，达到每分钟单机 TPC-C 基准测试记录——每分钟 658,277 笔事务交易，是当前世界上最快的交易处理平台。这一 TPC-C 测试结果已经由独立的 TPC 组织的审计人员认可（TPC-C 基准测试是行业中公认的权威的和最为复杂的在线事务处理基准测试）。

SQL Server 2000 是一个具备完全 Web 支持的数据库产品，提供了对可扩展标记语言 (XML) 的核心支持以及在 Internet 上和防火墙外进行查询的能力。

SQL Server 2000 提供了以 Web 标准为基础的扩展数据库编程功能。丰富的 XML 和 Internet 标准支持允许您使用内置的存储过程以 XML 格式轻松存储和检索数据。您还可以使用 XML 更新程序容易地插入、更新和删除数据。

通过 Web 轻松访问数据。有了 SQL Server 2000，您可以使用 HTTP 来向数据库发送查询、对数据库中存储的文档执行全文搜索、以及通过 Web 进行自然语言查询。

强大而灵活的基于 Web 的分析。SQL Server 2000 分析服务功能被扩展到了 Internet。您可以通过 Web 浏览器来访问和控制多维数据。

使用 SQL Server 2000 可以获得非凡的可伸缩性和可靠性。通过向上伸缩和向外扩展的能力，SQL Server 满足了苛刻的电子商务和企业应用程序要求。

向上伸缩。SQL Server 2000 利用了对称多处理器 (SMP) 系统。SQL Server Enterprise Edition 最多可以使用 32 个处理器和 64 GB RAM。
向外扩展。向外扩展可以将数据库和数据负载分配给多台服务器。
可用性。通过增强的故障转移群集、日志传送和新增的备份策略，SQL Server 2000 达到了最大的可用性。

SQL Server 2000 是 Microsoft .NET Enterprise Server 的数据管理与分析中枢。SQL Server 2000 包括加速从概念到最后交付开发过程的工具。

集成和可扩展的分析服务。有了 SQL Server 2000，您可以建立带有集成工具的端到端分析解决方案，从数据创造价值。此外，还可以根据分析结果自动驱动商业过程以及从最复杂的计算灵活地检索自定义结果集。
快速开发、调试和数据转换。SQL Server 2000 带有交互式调节和调试查询、从任何数据源快速移动和转化数据、以及按 Transact-SQL 方式定义和使用函数等功能。您可以从任意 Visual Studio 工具以可视化方式设计和编写数据库应用程序。

简化的管理和调节。使用 SQL Server 2000，您可以很容易地在企业资源旁边集中管理数据库。可以在保持联机的同时轻松地在计算机间或实例间移动和复制数据库。　　SQL Server 2000 为用户提供了大规模联机事务处理 (OLTP)、数据仓库和电子商务应用程序所需的最新的出色数据库平台。本文简要概述了 SQL Server 2000 对 SQL Server 7.0 版本的改进。SQL Server 2000 为用户提供了完全集成的可扩展标记语言 (XML) 环境、在分析服务中添加了新的数据挖掘功能、用元数据服务增强了知识库技术。

Microsoft SQL Server 2000 提供的分析服务显著增强了 SQL Server version 7.0 引入的联机分析处理 (OLAP) 服务组件的功能。分析服务引入了数据挖掘功能，可以用来在 OLAP 多维数据集和关系数据库中发现信息。请了解 SQL Server 2000 是如何改进安全控制、增强客户连通性以及实现实时数据分析的。

Microsoft SQL Server 2000 元数据服务扩展并重命名了以前称为 Microsoft 知识库的知识库组件。请了解元数据服务是如何通过引入新的知识库数据浏览器、新的 XML 交换支持和新的知识库引擎功能来扩展知识库技术的。

本次设计中使用了 SQL SEVER 中的存储过程技术，以提高系统的执行效率和修改的方便度。

第三章 系统详细设计

3.1 需求分析

网上购物由于有广大的销售群体，又有展示新产品、新工艺的网络平台，所以通过电子商务可以迅速地对顾客的需求构成导向。同时，企业也可以售、购买形式和购物动态；以及客户对产品的意见，商家通过这些统计数据来获知客户对产品的满意度。本次设计的电子商务网站主要面向商家的是一些中小型的企业用户，而面向的客户以个人客户为主，企业级用户为辅。

(1)系统要求能形成一个完整的购物流程，实现对客户和商品的管理以及对仓库的物流操作。

(2)系统高度集成和模块化:进入系统的数据要能根据事先的设定以及管理工作的内在规律传递到相关的功能模块中 ,达到数据高度共享和系统的高度集成;软件系统在设计和开发过程中要保证各模块的功能。

(3)系统要面向电子商务:新一代的管理软件应当支持 Internet 上的信息获取及网上交易的实现。

(4)个性化界面:即是用户设计自己的 Web 界面，而不是简单的模仿。

(5)可靠性和安全性:大规模的系统、分布式应用、广泛的网络连接需要系统具有一定的可靠性和安全控制机制。

(6)具备可扩展的业务框架：有一个易于扩展的框架结构。使得发商今后对软件的维护和扩展变得更为容易 ,也使应用系统的客户化和二次开发变得简单。

(7)系统需支持客户与企业的交互，实现商务信函功能。

(8)系统需实现客户之间的交流，即实现小论坛功能。

(9)同时网站要求能运行于一般配置的客户机上，对系统资源的节省也要作为设计的考虑环节。

(10)系统要求在机器环境的允许下能支持上百人同时在线购物,并维护其安全性。

3.2 系统分析

- 网站主体采用 ASP.NET 等动态网页脚本语言编写，重要交易与数据库操作模块采用 C#.NET 编写，利用 ADO.NET 方式调用后台数据库。
- 网站硬软件系统应安全可靠，作为前期试验方案，由于硬件条件的限制，人数限制为支持百人同时交易。
- 页面总体风格一致，具有艺术感和创新性，符合实际工作需要。
- 实现用户注册、商品订购、网上付款、物流管理的整个交易过程
- 实现商务信函和论坛交互模块。
- 人机界面友好，工作性能可靠，便于维护。

总体上网站概括性的分为用户管理，商品管理，物流管理，交易管

理等四个主要模块。

用户管理：

用户管理模块主要实现用户的注册，登陆登出，修改资料，注销信息等人事方面的功能；

商品管理：

商品管理模块主要实现商品的查看，商品信息的修改更新，新商品的搜索等功能了；

物流管理：

物流管理模块主要实现库存商品的查看，商品的进货出货，以及对这些活动的记录和查看，此外，它还包括对新商品的引进；

交易管理：

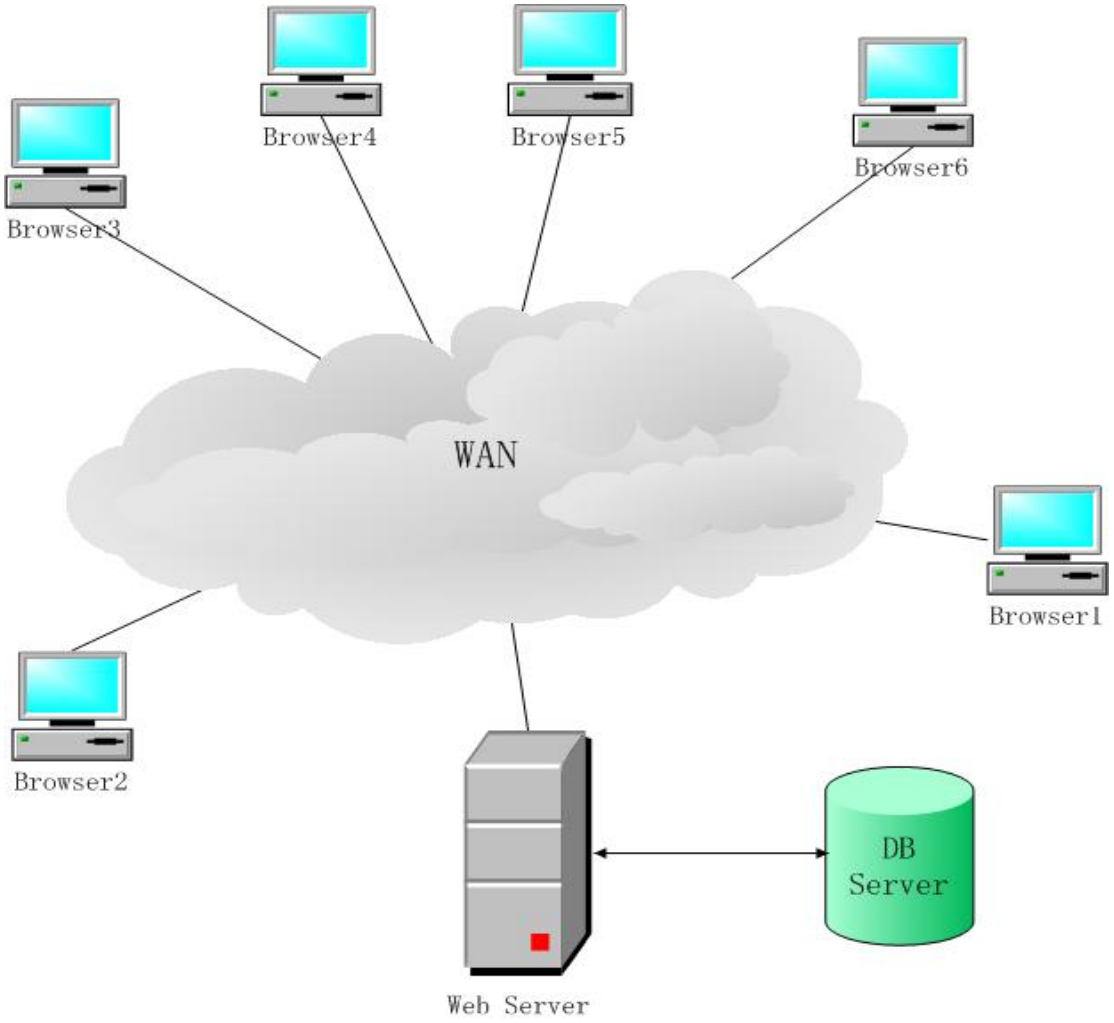
交易管理模块主要实现商品的订购和网上付款的确认，以及对交易进行的记录。

3.3 系统结构图

网站基于 B/S 即浏览器/服务器结构进行开发。

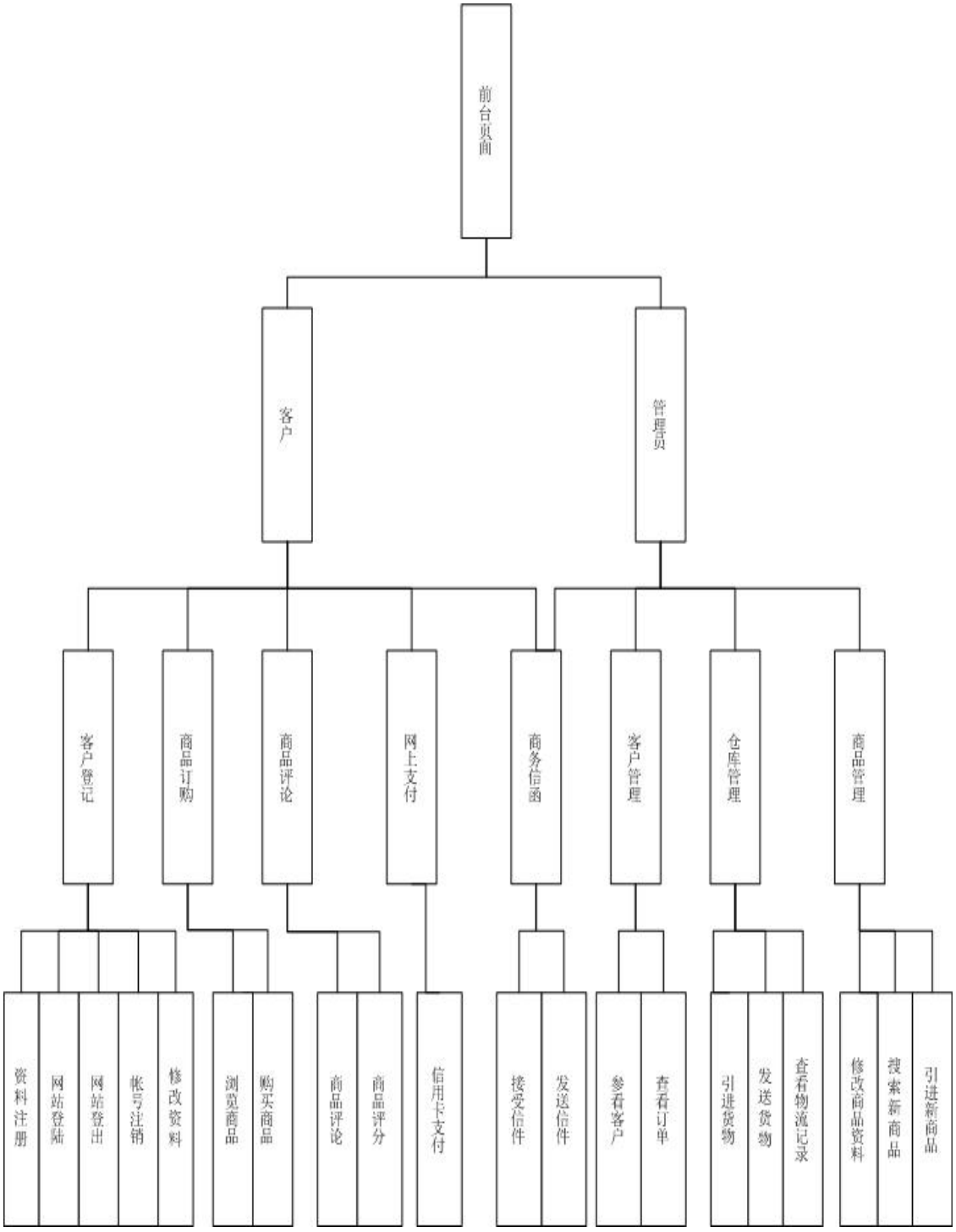
B/S 结构，即 Browser/Server（浏览器/服务器）结构，是随着 Internet 技术的兴起，对 C/S 结构的一种变化或者改进的结构。在这种结构下，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现，形成所谓 3-tier 结构。B/S 结构，主要是利用了不断成熟的 WWW 浏览器技术，结合浏览器的多种 Script 语言（VBScript、JavaScript…）和 ActiveX 技术，用通用浏览器就实现了原来需要复杂专用软件才能实现的强大功能，并节约了开发成本，是一种全新的软件系统构造技术。随着 Windows 98/Windows 2000 将浏览器技术植入操作系统内部，这种结构更成为当今应用软件的首选体系结构。显然 B/S 结构应用程序相对于传统的 C/S 结构应用程序将是巨大的进步。

图示如下：



3.4 网站工作流程

3.4.1 网站图示：



3.4.2 网站模块划分：

网站具体分为前台主体页面，商品订购模块，客户管理模块，网上付款模块，物流管理模块，商务信函模块，商品评论模块。

前台主体页面:前台页面除包含一下几大模块之外,它还具有一些广告和商品展

示的部分。同时，它还包括了为用户提供的在线帮助以及售后服务条款和隐私政策。

客户登记模块功能:分为登记,登陆,修改信息,注销.在登陆的时候可选择是不是会员,会员可享受购物优惠价.新来的客户可以进行登记注册,填写自己的基本信息:帐号,密码,生日,联系方式和信用卡号(存入客户信息表).日后若有变动,可以通过修改信息来改动.注销可以删除该用户在网站上储存的个人信息.经过登记后的客户可以输入帐号和密码登陆网站,使用其他服务功能.

商品订购模块功能:可通过查看前台页面所展示的商品来进行订购.按购物车按钮,然后在页面中输入商品型号,商品名称,订购数量,收货地址后按提交,若存在想要的商品,则显示订购成功,若没有则显示客户想要选择的商品不存在.若购物超过一定的总额,则可升级为会员,享受打折优惠,在下次登陆时可在是否会员的选项上选是.接下来客户的订购信息将存入名为订购单的表中,以便物流管理.系统产生一个订购序列号返还给客户.

网上付款模块功能:在客户完成订购之后,就到此处进行网上支付.可在页面中填写订购序列号和信用卡号,系统进行查询核对后,返回给客户操作结果.

物流管理模块功能:这个模块分为商品进出货管理和商品种类的添加以及商品的查询和信息修改.该模块上有一个管理员登陆界面,管理员登陆后能进行该模块的功能操作.在客户完成订购和网上支付确认后,管理员根据客户的订购单出货,管理员还可以进货来弥补商品数量的不足,亦可引进新商品.

商务信函模块功能:在这个模块上也有一个管理员登陆界面,登陆后可想用户发送信件,可通过查询订购单得到用户帐号再对客户信息表进行查询取得电邮地址,向已订购货物的客户发送一些关于收货或者是商品变动,比如价格变化等最新信息.用户通过该页面向网站发送回函.

商品评论模块功能:这个模块由客户对网站的商品进行评论，可以与其他用户进行书面上的交流，还可以对商品进行打分，以此来体现商品的受欢迎度。

3.4.3 网站操作及具体功能：

该网站分为管理员操作和用户操作两大块：

管理员模式：

进入**主页**，在 RadioButtonList 控件中选择管理员选项，点击登陆，若用户名和密码有错，则页面会显示出错提示，若正确则跳转进入**管理员界面**，管理员可点击查看订单按钮进入**订购单**页面来显示客户的订单并对其进行操作，可在该页面点击进入**仓库**页面进行货物发送；通过输入客户帐号点击查看客户可以进入**客户信息查看**页面查看客户的资料，然后可在该页点击通知客户按钮进入**商务信函**页面对客户进行新消息的通知；同时，管理员也能通过网站的信件提示来查收新的邮件，进入**管理员信件**页面可以查收信邮件，还能对客户发来的邮件进行回

复；可在管理员页面点击仓库按钮进入**仓库**页面执行物流方面的管理操作。在**仓库**页面点击库存信息可查看库存商品的信息，返回给管理员商品的名称，类别，序列号，单价及数量，如果发现库存商品数量短缺，可以对缺货的商品进行进货操作，点击新商品搜索按钮能搜索到最新的商品，然后可以对想要的新商品进行进货操作，**仓库**也可以给已经订购商品的客户发货，这些进货和发货的记录都保存到进货记录表以及发货记录表中，通过点击进货记录和发货记录按钮来查看。点击商品信息修改按钮能进入**商品信息修改**页面，在那里可以对商品最新的变动进行修改以更新网站的商品信息。

客户模式：

进入**主页**，在 RadioButtonList 控件中选择管理员选项，若是会员可选中下面控件中的会员选项，点击登陆，若用户名和密码有错，则页面会显示出错提示，若正确则会显示相应的欢迎信息，这时，原本不可使用的控件如注销，信息修改，登出等都变为可用控件。假如访问者还不是本网站的用户，可点击注册按钮进入**注册条款**页面，在阅读条款之后点击确定进入**注册资料填写**页面，填写完自己的信息之后（有些项目为必填，有些项目要严格按照规定填写，如果有填写格式错误或漏填，则会有相应的错误提示出现。）若是觉得填写有误可按重填，若是觉得 OK 了，则点击提交进入**注册欢迎**页面，该页面显示欢迎信息，并返回一个帐号给客户。然后由系统自动或是客户点击跳转至**主页**登陆。客户登陆之后，可点击**主页**的信息修改按钮进行个人信息的修改，包括客户的登陆密码，点击后进入**身份验证**页面，这是为了确保您的信息不被他人修改而采取的二次验证，这个验证需要的是客户的信用卡号。验证成功后进入**个人信息修改**页面，在此进行信息和密码的修改，提交之后进入**修改成功**页面，系统将在 3 秒后自动跳转会**主页**。客户还可点击**主页**的登出按钮登出网站。点击注销按钮将删除客户的注册资料和帐号，但不影响客户其他记录。点击进入商城按钮可进入**商城**页面，在**商城**页面中选择商品类型可以显示相对应的商品及信息。客户在浏览货物时可点击购物车按钮进行选购，然后在出现的输入框内填入自己想要的数量，点击确定后进入**购物车**页面，在该页面进行购物信息的确认，可以删除不想要的商品，您也可以点击返回急需进行购物。重复此操作，直到客户完成所有购物目的之后，在**购物车**页面点击订单确认按钮进入**订单确认**页面查看订单，若订单无误则点击确认支付按钮进入**网上支付**页面，在该页面填写您的信用卡号进行交易，页面将显示成功与否及会员升级的信息。在商城中挑选自己喜欢的商品的同时，可以点击评论按钮进入**商品评论**页面查看其他客户对该商品的评价以及对它的评分，客户也可以自己发表看法和为该商品打分。在**主页**上还可以进入商务信函模块，客户可在这里查收网站管理人员发送的信件，也可以对网站进行消息反馈。

3.5 网站设计

网站采用传统的两层结构开发：

3.5.1　两层应用程序结构

典型的两层应用程序是使用 ADO.NET 直接与数据库服务器（如 Microsoft SQL Server™）进行通信的客户端应用程序（参见图 1）。除 ADO.NET 外，在客户端应用程序和数据库之间没有任何其他层。有关 ADO.NET 的详细信息，请参阅 .NET 框架文档、本系列的其他文章或使用 MSDN 搜索引擎。

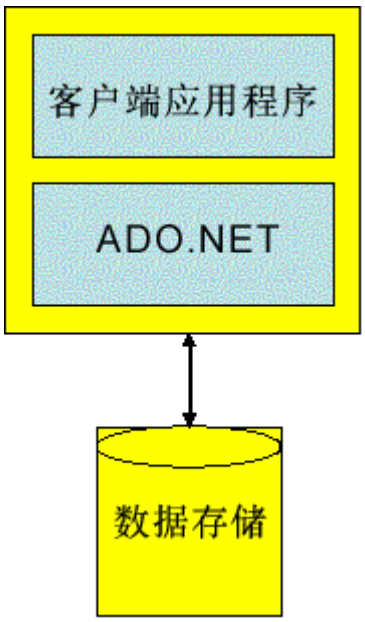


图 1：两层应用程序包括客户端应用程序和数据存储（如 Microsoft SQL Server）

何时使用两层结构

两层应用程序适用于没有或只有少量窗体的小型应用程序。对于使用本文中介绍的其他 N 层技术的应用程序，其原型也可算是两层应用程序。但是，两层应用程序不太适用于企业环境，因为开发和维护的时间及成本不好控制。

典型的实现方式

开发两层应用程序时可以采用多种技术。所有技术均使用 ADO.NET、一个客户端界面（如桌面或基于 Web 的应用程序）和一个数据库（本网站采用 SQL Server）。要使用两层应用程序结构，可以采用以下方式：

- 使用数据绑定技术将 ADO.NET 数据集直接与控件连接。
- 编写代码以访问 ADO.NET 对象，从而手动将数据加载到用户界面的控件中。
- 使用上述两种技术的组合。
- 使用上述任何技术直接在窗体上编写业务规则代码。

优点

两层应用程序具有以下优点：

- 因为可以使用数据绑定将 ADO.NET 数据集直接与用于构建用户界面的很多控件连接，所以开发工作就变得简单而快捷。这有助于迅速建立并运行应用程序的基本功能。
- 只需查看窗体便可以浏览应用程序的全部代码，而无须同时查看窗体和另一个组件。

缺点

两层应用程序开发方法具有以下缺点：

- 所有业务规则均包含在前端代码中。因而，如果需要更改业务规则，则必须更新全部客户端。除非能够进行自动更新，否则这种维护工作将十分繁琐。当然，如果使用 SQL Server，则可以将某些业务规则放到存储过程中，从而减少维护的时间和成本。
- 尽管 SQL 可以在数据库结构和应用程序的其他部分之间提供某种程度的精简，但字段名称通常还是在源代码或控件属性中硬编码的。如果更改字段名称，则必须查找和替换应用程序中所有该字段的名称。如果使用了数据绑定，还必须检查所有窗体并更改属性。
- 很多代码（如 SQL 语句和业务规则）常常在应用程序中重复出现，这是因为不同的窗体使用了某些相同的表。这使得此类应用程序的维护非常困难，因为如果需要更改表或字段的名称，则必须在多个位置进行更改，同时还需要在多个位置进行额外的回归测试。
- 如果数据源发生变化，则对用于将数据加载到数据集的代码的更改将更加困难。例如，如果原来使用文本文件存储数据，然后又希望转换到 SQL Server，其访问方式是截然不同的。此外，要将数据加载到应用程序的数据集，还有很多地方都需要更改代码。

3.6 两层结构设计

3.6.1 前台业务层设计：

1)　WEB SERVICE 简介：

现在 Internet 正在不断地发展着，在互联网应用刚开始的时候，我们浏览的网页只是静态的，不可交互的。而现在随着技术的日益发展，将提供给网页浏览者一个可编程的 Web 站点。这些站点将在组织、应用、服务、驱动上更加紧密的结合在一起，这些站点将通过一些应用软件直接连接到另一个 Web 站点，这些可编程的 Web 站点相比传统的 web 站点来说，将变得更加能重复使用，也更加智能化！

.net 平台给我们提供了一种运行环境，即公用语言运行环境（CLR，Common

Language Runtime）。对 CLR 来说，它提供了一种内置机制来创建一个可编程的站点，、对于 Web 程序开发者来说，这将是一致、熟悉的。这种模型是可以重复使用，也可以再扩展。 它包含了开放的 Internet 标准（HTTP, XML, SOAP, SDL）。以便它能被网页浏览者访问。

ASP.NET 使用.asmx 文件来对 Web Services 的支持。.asmx 文件和.aspx 文件一样都属于文本文件。它包含在.aspx 文件之中，成为 ASP.NET 应用程序的一部分。

本设计中使用了 WEB SERVICE 这一项目。

2） WEB FORM

创建了 WEB SERVICE 之后，可以建立 WEB FORM。

Web Form 代表了一个一个的 Web 页面。总的看来，Form 就像是一个容纳各种控件的容器，各种控件都必须直接或者间接的和它有依存关系。

网站页面的介绍：

AdministorMail.aspx　　//管理员邮箱页面；
AdministorPage.aspx　　//管理员主页面；
Authentication.aspx　　//验证页面；
BusinessMail.axpx　　//管理员信件页面；
BusinessMail2.aspx　　//客户信件页面；
ChangeSuccess.aspx　　//客户信息修改成功页面；
CustomerMail.aspx　　//客户邮箱页面；
HomePage.aspx　　//网站主页面；
Help.aspx　　//在线帮助页面；
ItemComment.aspx　　//商品评论页面；
ItemInfor.aspx　　//商品信息页面；
MailSentSuccess.aspx　　//信件发送成功页面；
MaterialsCirculation.aspx　　//物流页面；
MemberInfor.aspx　　//客户信息页面；
OrderConform.aspx　　//购物车页面；
Payment.aspx　　//网上付款页面；
Regist.aspx　　//客户注册页面；
RegistRequest.aspx　　//注册条款页面；
RegistWelcome.aspx　　//注册欢迎页面；
SecretsPolicy.aspx　　//隐私政策页面；
Shelf.aspx　　//货架页面；
Storage.aspx　　//仓储页面；
Storehouse.aspx　　//仓库页面；
Service.aspx　　//售后服务页面；

3） 控件的加入

在建立了 WEB FORM 之后，在页面中拖入想要加入的控件，并对它们的属性进行定义，可以在右边的属性栏中直接定义，也可以在后台的代码页面中进行修改。

本次设计主要使用的控件为：

BUTTON　　//按钮控件；
DROPDOWNLIST　　//下拉框控件；
DATAGRID　　//数据栏控件；
HYPERLINK　　//超链接控件；
IMAGE　　//图片控件；
IMAGEBUTTON　　//图片按钮控件；
LINKBUTTON　　//链接按钮控件；
LABEL　　//标签控件；
RADIOBUTTON　　//选择控件；
TABLE　　//表控件；
TEXTBOX　　//文本框控件；

4） 页面代码的设计

PAGE_LOAD：

首先，页面的状态被恢复，然后触发 Page_OnLoad 事件。在这个过程中，你可以读取或者重置页面的属性和控件的属性，根据 IsPostBack 属性判定页面是否为第一次被请求，执行数据绑定，等等。

CLICK 事件：

用户按下按钮以后，即将触发的事件。在网站设计中利用此事件完成对用户选择的确认、对用户表单的提交、对用户输入数据的修改等等。在按钮的事件中编写相应的功能的代码，加入判断条件和与后台数据库的连接代码即可实现用户想要实现的功能。

5) 模块与网页的关联（附流程图）

前台页面模块－－主页（Homepage.aspx），在线帮助页面(Help.aspx)，隐私政策页面(SecretsPolicy.aspx)，售后服务页面(Service.aspx)

客户登记模块－－注册条款页面（RegistRequest.aspx），注册页面(Regist.aspx), 注册欢迎页面（RegistWelcome.aspx），验证页面（Authentication.aspx），客户信息页面（MemberInfor.aspx），客户信息修改成功页面（ChangeSuccess.aspx）

商品订购模块－－货架页面（Shelf.aspx），购物车页面（OrderConform.aspx）

网上付款模块－－网上付款页面（Payment.aspx）

仓库管理模块－－仓库页面（Storehouse.aspx），仓储页面（Storage.aspx），物流页面（MaterialsCirculation.aspx），商品信息（ItemInfor.aspx）

商品评论模块－－商品评论页面（ItemComment.aspx）

商务信函模块－－管理员主界面（AdministorPage.aspx），管理员邮箱页面（AdministorMail.aspx），管理员信件页面（BusinessMail.axpx），客户邮箱页面（CustomerMail.aspx），客户信件页面（BusinessMail2.axpx），信件发送成功页面

（MailSentSuccess.aspx）

3.6.2 数据存储层设计：

整个数据库的关键是表结构的设计，它独立于整个数据库的逻辑结构，能充分反映现实世界,包括实体之间的联系能满足用户需求。而且易于扩充和修改。

1） 数据库表设计：

管理员登陆信息表（Administor）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
Administor	管理员帐号	是		否	char	10
AdminPassword	管理员密码				char	20

管理员信件表（AdministorMail）

属性列	属性意义	是否主健	是否外健	可否为空	数据类型	数据长度
Administor	管理员帐号				char	10
CustomerID	客户帐号				int	4
Title	主题				char	100
Content	内容				char	1000
WriteDate	发信时间	是		否	char	100
New	是否新邮件				char	10

商品评论表（Comment）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
CommentRecord	评论序列号	是		否	Int	4
ItemName	商品名				char	20
CustomerID	客户帐号				int	4
Comment	评论				char	150
CommentDate	评论时间				char	20

客户注册信息表（Customer1）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
CustomerID	客户帐号	是		否	int	4
CustomerName	客户姓名				char	8
Sex	客户性别				char	2
Bothday	客户生日				char	14
Adress	客户住址				char	28
Mail	客户电邮				char	20
PhoneNumber	客户电话				float	8
CredictCard	客户信用卡号				float	8
Member	是否会员				char	2

客户登陆信息表(Customer2)

属性列	属性意义	是否主健	是否外健	可否为空	数据类型	数据长度
CustomerID	客户帐号	是		否	int	4
Password	客户密码				char	20
CustomerName	客户姓名				char	8

客户信件表(CustomerMail)

属性列	属性意义	是否主健	是否外健	可否为空	数据类型	数据长度
CustomerID	客户帐号				int	4
Administor	管理员帐号				char	10
Title	主题				char	100
Content	内容				char	1000
WriteDate	发信时间	是		否	char	100
New	是否新邮件				char	10

厂家信息表（Factory）

属性列	属性意义	是否主健	是否外健	可否为空	数据类型	数据长度
FactoryName	厂家名	是		否	char	20

FactoryAdress	厂家地址				char	28
FactoryPhone	厂家电话				float	8

商品信息表（ItemInfor）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
ItemID	商品号	是		否	int	4
ItemType	商品类别				char	10
ItemName	商品名				char	20
ItemCost	商品单价				float	8
ItemMember Cost	商品会员价				float	8
ItemInfor	商品规格				char	100
FactoryName	出产厂家				char	20
SellDate	上市日期				char	10
HotItem	是否热销				char	2
ItemScore	商品分数				float	8

商品库存表（ItemStored）

属性列	属性意义	是否主健	是否外健	可否为空	数据类型	数据长度
ItemID	商品号	是		否	float	8
ItemName	商品名				char	20
ItemCost	商品单价				float	8
ItemStored	商品库存量				float	8

新商品表(NewItemInfor)

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
NewItemName	新商品名	是		否	char	20
NewItemType	新商品类型				char	10
NewItemCost	新商品单价				float	8
NewItemInfor	新商品描述				char	100
NewSellDate	上市日期				char	10
FactoryName	厂家名称	是		否	char	20

订购单表（OrderList）

属性列	属性意义	是否主健	是否外健	可否为空	数据类型	数据长度
RecordNumber	订购单号	是		否	int	4
CustomerID	客户帐号				int	4
ItemName	商品名				char	20
ItemNumber	订购数量				float	8
OrderDate	订购日期				char	20

进货记录表（RecievedItemRecord）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
RecordNumber	进货单号	是		否	int	4
ItemName	商品名				char	20
ItemAdd	进货量				float	8
RecieveDate	进货日期				char	28
FactoryName	厂家名称				char	20

出货记录表（SendItemRecord）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
RecordNumber	出货单号	是		否	int	4
ItemName	商品名				char	20
ItemReduce	出货量				float	8
SendDate	出货日期				char	28
SendAdress	送货地址				char	20

购物车表（TempList）

属性列	属性意义	是否主键	是否外键	可否为空	数据类型	数据长度
ID	购物车号	是		否	int	4
CustomerID	客户帐号				int	4
ItemName	商品名				char	20
ItemCost	商品单价				float	8
ItemMemberCost	商品会员价				float	8
OrederNumber	购买数量				float	8

表与表之间的关联：

表名	关联表	关联键

Administor	AdministorMail	Administor
Administor	CustomerMail	Administor
AdministorMail	Customer1	CustomerID
AdministorMail	Administor	Administor
Comment	Customer1	CustomerID
Customer1	Customer2	CustomerID
Customer1	AdministorMail	CustomerID
Customer1	Comment	CustomerID
Customer1	OrderList	CustomerID
Customer1	TempList	CustomerID
Customer2	Customer1	CustomerID
CustomerMail	Administor	Administor
Factory	ItemInfor	FactoryName
ItemInfor	Factory	FactoryName
ItemInfor	ItemStored	ItemID
ItemStored	ItemInfor	ItemID
NewItemInfor	无	无
OrderList	Customer1	CustomerID
RecievedItemRecord	无	无
SendItemRecord	无	无
TempList	Customer1	CustomerID

2） 存储过程设计

在后台数据库上编写网站所需操作的存储过程可以减少页面上的代码量，同时也可以增加执行效率。

存储过程是一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中。用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。总的来说，存储过程具有以下一些优点：

1.存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般 SQL 语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度。

2.当对数据库进行复杂操作时(如对多个表进行 Update,Insert,Query,Delete 时)，可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用。

3.存储过程可以重复使用,可减少数据库开发人员的工作量，也可以减少网络流量。

4.安全性高,可设定只有某此用户才具有对指定存储过程的使用权。

5. 存储过程的能力大大增强了 SQL 语言的功能和灵活性。存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

6．可保证数据的安全性和完整性。通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全。通过存储过程可以使相关的动作在一起发生，从而可以维护数据库的完整性。

存储过程分类目录：

- Select（查询操作）：

AdministorItemSearch　//查询获取商品信息；
Checkadmin　//查询验证管理员帐号；
CheckadminPasswd　//查询验证管理员密码；
CheckCustomerID　//查询验证客户帐号；
CheckPassword　//查询验证客户密码；
CheckNewMail1　//查询管理员信件状况；
CheckNewMail2　//查询用户信件状况；
CredictCardCheck　//查询验证信用卡号；
CustomerInfoCheck　//查询获取客户资料；
CustomerInfor　//查询获取客户信息；
GetComment　//查询商品评论；
GetID　//查询客户帐号；
GetItem1　//查询商品分类信息；
GetItem2　//查询商品分类信息；
GetItem3　//查询商品分类信息；
GetItem4　//查询商品分类信息；
GetItemInfor　//查询获取修改商品信息；
GetItemScore　//查询商品评分；
GetItemStoredInfor　//查询商品库存信息；
GetMailContent1　//查询管理员信件内容；
GetMailContent2　//查询客户信件内容；
GetNewItemInfor　//查询获取新商品信息；
GetNameAndSex　//查询获取用户姓名和性别；
GetNewMail1　//查询管理员新信件；
GetNewMail2　//查询客户新信件；
GetOldMail1　//查询管理员旧信件；
GetOldMail2　//查询客户旧信件；
GetRecieveItemRecord　//查询获取进货记录；
GetSendItemRecord　//查询获取发货记录；
ItemInforChangeLoad　//查询验证商品修改；
ListCheck　//查询获取客户订购单（管理员）；
ListView　//查询获取客户订购单（客户）；
MemberCheck　//查询验证会员资格；
SearchItem　//搜索商品；

- Insert（插入操作）

InsertComment　　//插入商品评论；
InsertCustomerInfor　　//插入客户注册资料；
InsertNewItem　　//添加新商品；
InsertNewMail1　　//插入管理员信件；
InsertNewMail2　　//插入客户信件；
InsertNewStored　　//添加新商品库存；
InsertPassPermission　　//插入客户登陆验证信息；
InsertRecieveItemRecord　　//添加进货记录；
InsertSendItemRecord　　//添加发货记录；
InsertTempOrder1　　//插入临时购物车；
OrderItem　　//插入订购的商品的信息；
SaveList　　//插入订购单；

- Delete（删除操作）

DeleteCustomerInfor　　//删除客户注册资料和帐号；
DeleteNewItem　　//删除新商品；
DeleteOrderList　　//删除客户订购单；
DeleteOrderRecord　　//删除临时商品；
DeleteTempList　　//清空购物车；

- Update（更新操作）

Passwordchange　　//更新客户密码；
UpdateAdministorMail　　//更新管理员信件状态；
UpdateCustomerInfor　　//更新客户信息；
UpdateCustomerMail　　//更新客户信件状态；
UpdateItemInfor　　//更新商品信息；
UpdateItemScore　　//更新商品评分；
UpdateItemStored（Recieved）　　//更新库存（进货）；
UpdateItemStored（Send）　　//更新库存（发货）；
UpdateTempOrder　　//更新购物车；

对表的操作：

表名	操作
Administor	Checkadmin
	CheckadminPasswd
AdministorMail	CheckNewMail1
	GetMailContent1
	GetNewMail1
	GetOldMail1
	InsertNewMail1
	UpdateAdministorMail
Comment	GetComment

	InsertComment
Customer1	CheckCustomerID
	CredictCardCheck
	CustomerInfoCheck
	CustomerInfor
	GetID
	GetNameAndSex
	MemberCheck
	InsertCustomerInfor
	DeleteCustomerInfor
	UpdateCustomerInfor
	DeleteCustomerInfor
Customer2	CheckPassword
	InsertPassPermission
	DeleteCustomerInfor
	Passwordchange
CustomerMail	CheckNewMail2
	GetMailContent2
	GetNewMail2
	GetOldMail2
	InsertNewMail2
	UpdateCustomerMail
Factory	无
ItemInfor	AdministorItemSearch
	GetItem1
	GetItem2
	GetItem3
	GetItem4
	GetItemInfor
	GetItemScore
	ItemInforChangeLoad
	InsertNewItem
	SearchItem
	UpdateItemInfor
	UpdateItemScore
ItemStored	GetItemStoredInfor
	InsertNewStored
	UpdateItemStored（Recieved）
	UpdateItemStored（Send）
NewItemInfor	GetNewItemInfor
	DeleteNewItem
OrderList	ListCheck
	SaveList

	DeleteOrderList
RecievedItemRecord	GetRecieveItemRecord
	InsertRecieveItemRecord
SendItemRecord	GetSendItemRecord
	InsertSendItemRecord
TempList	ListView
	InsertTempOrder1
	OrderItem
	DeleteOrderRecord
	DeleteTempList
	UpdateTempOrder

第四章 系统的实现

4.1 软硬件环境

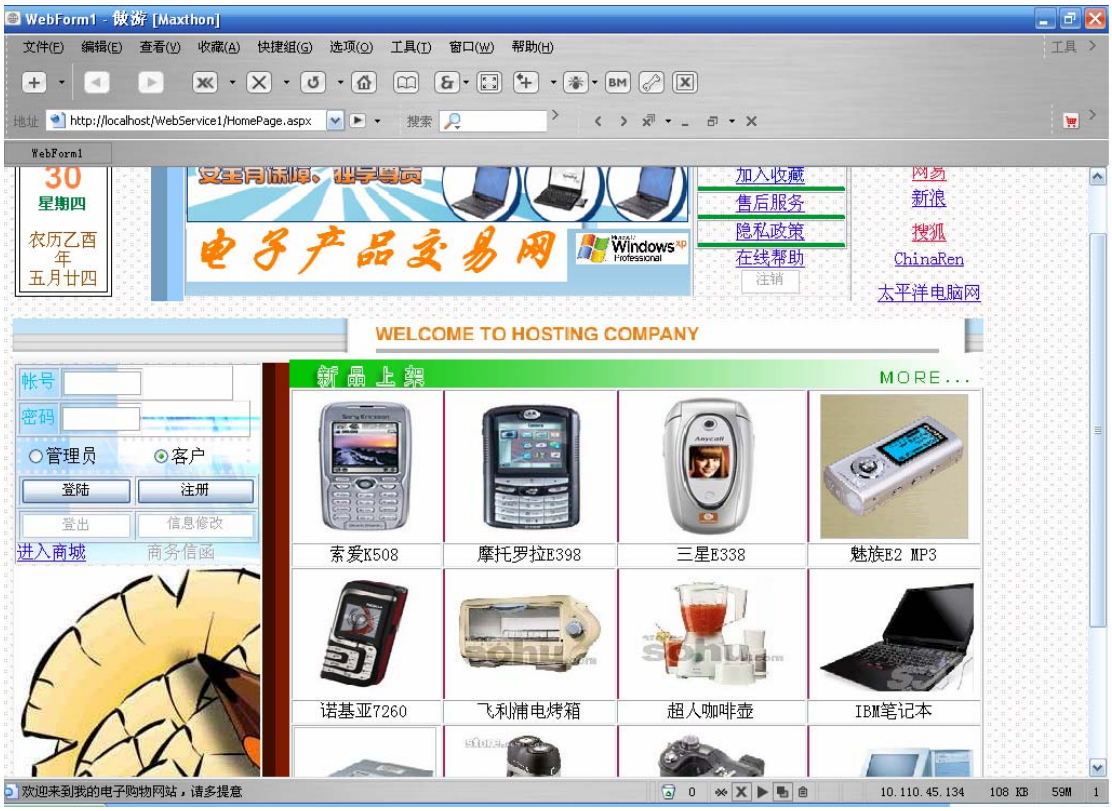
网站实现的硬件环境为：
家用微机一台；
处理器：PIII 866MHZ
内存：384M SDRAM
硬盘：IBM 40GB 7200 转
显示器：PHILIPS 107T
网站实现的软件环境为：
操作系统：Microsoft Windows XP Professional service pack2
浏览器：IE6.0
IIS：5.1
开发软件：Visual Studio 2003
　　SQl Sever 2000 开发版
系统须安装.net Framework 1.0 或以上版本；

电子商务中所涉及的网络技术及数据库技术渐成熟,本文的设计采用了ASP.net 服务器端执行本指令环境。后台的开发工具选用 C#和 SQLSERVER 数据库。这选择的目的是便于系统的改进和扩充。机器硬件可以选用 PentiumIII以上机型、10/100M 自适应卡、内存 128M、硬盘 20G。操作系统可以选用 Win2000,WindowsNT 等。

4.2 网站的实现及其相应关键代码（图文介绍）

本次设计的关键技术主要是存储过程的设计，以及使用 ADO.NET 来实现前台与后台的连接和以使用设计好的存储过程，通过这个方法来实现网站的主要功能。同时还附以页面间参数的传递和一些功能的判断语句来完善网站功能。

以下是网站图示：



网站主页面

4.2.1 页面判断的实现

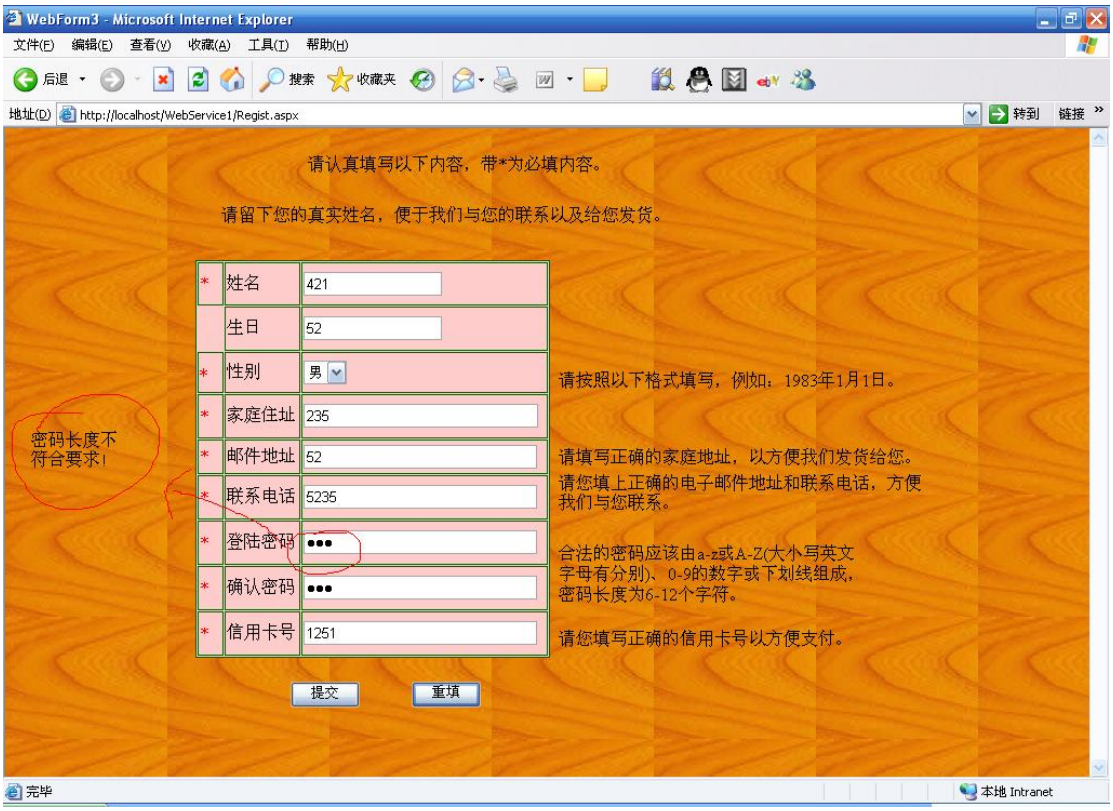
可在页面中写入一些判断语句，以实现客户想要的功能。
例如对这册资料填写的 if 判断语句：

```
if(this.TextBox1.Text=="" || this.TextBox2.Text=="" || this.TextBox5.Text==""||
this.TextBox6.Text=="" || this.TextBox7.Text==""|| this.TextBox8.Text=="" ||
this.TextBox9.Text=="" )
{     this.Label19.Visible=true;
      this.Label26.Visible=false;
      this.Label27.Visible=false;}
else
if(this.TextBox1.Text.Length<6||this.TextBox8.Text.Length<6||this.TextBox1.Text.L
ength>12||this.TextBox8.Text.Length>12)
  {   this.Label27.Visible=true;
      this.Label19.Visible=false;
      this.Label26.Visible=false;}
    else if(this.TextBox8.Text!=this.TextBox1.Text)
  {   this.Label26.Visible=true;
      this.Label19.Visible=false;
      this.Label27.Visible=false;}
  else
{   Cache["Text2"]=this.TextBox2.Text;
```

```
Cache["Text3"]=this.DropDownList1.SelectedItem.Text;
Cache["Text4"]=this.TextBox9.Text;
SqlDataAdapter dCommand = new SqlDataAdapter();
 dCommand.InsertCommand = new SqlCommand();
 dCommand.InsertCommand.Connection = new SqlConnection(SqlConn.ConStr);
 dCommand.InsertCommand.Connection.Open();
 SqlCommand command1 = dCommand.InsertCommand;
command1.CommandText = "InsertCustomerInfor";
command1.CommandType = CommandType.StoredProcedure;
 command1.Parameters.Clear();
 command1.Parameters.Add("@CustomerName",this.TextBox2.Text.ToString());
 command1.Parameters.Add("@Sex",this.DropDownList1.SelectedValue.ToStrin
g());
 command1.Parameters.Add("@Borthday",this.TextBox4.Text.ToString());
 command1.Parameters.Add("@Adress",this.TextBox5.Text.ToString());
 command1.Parameters.Add("@Mail",this.TextBox6.Text.ToString());
 command1.Parameters.Add("@PhoneNumber",Convert.ToInt32(this.TextBox7.T
ext));
 command1.Parameters.Add("@CredictCard",Convert.ToInt32(this.TextBox9.Tex
t));
 command1.ExecuteNonQuery();
 command1.CommandText = "InsertPassPermission";
 command1.CommandType = CommandType.StoredProcedure;
 command1.Parameters.Clear();
 command1.Parameters.Add("@Password",this.TextBox8.Text.ToString());
 command1.Parameters.Add("@CustomerName",this.TextBox2.Text.ToString());
 command1.ExecuteNonQuery();
 dCommand.InsertCommand.Connection.Close();
 Response.Redirect("http://localhost/webservice1/RegistWelcome.aspx");}
                    }
```

上述代码用来判断注册资料填写的正确性和密码输入的格式限制。

图示：



4.2.2 存储过程的实现

存储过程的作用：

存储过程可以使得对数据库的管理、以及显示关于数据库及其用户信息的工作容易得多。存储过程是 SQL 语句和可选控制流语句的预编译集合，以一个名称存储并作为一个单元处理。存储过程存储在数据库内，可由应用程序通过一个调用执行，而且允许用户声明变量、有条件执行以及其它强大的编程功能。存储过程可包含程序流、逻辑以及对数据库的查询。它们可以接受参数、输出参数、返回单个或多个结果集以及返回值。可以出于任何使用 SQL 语句的目的来使用存储过程。

存储过程的运用：

本次设计中，存储过程运用于各个模块的主要功能中，比如人事管理，商品购买和物流以及商品评论和信件收发等等。

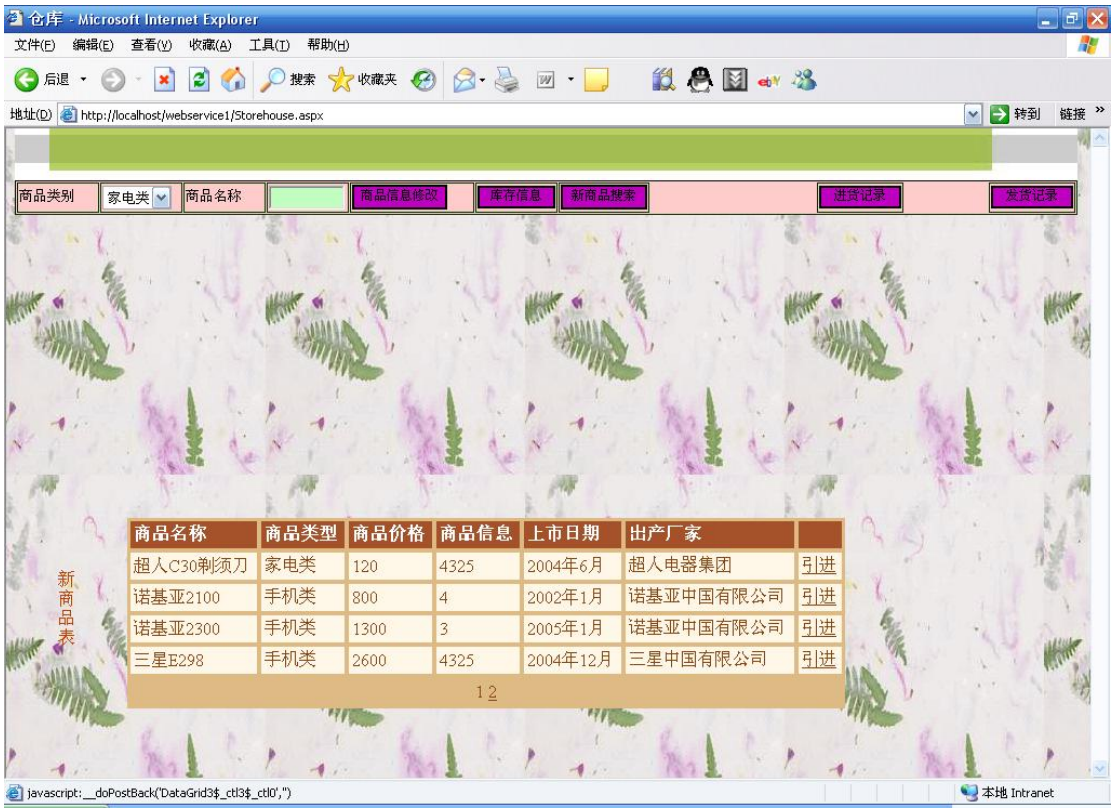
以下是我编写的一个存储过程的实例：

```
CREATE PROCEDURE InsertNewItem
@NewItemName char(20),
@NewItemType char(10),
@NewItemCost float,
@NewItemInfor char(100),
@NewSellDate char(10),
@FactoryName char(20)
AS
begin transaction
```

```
insert into ItemInfor(ItemType,ItemName,ItemCost,ItemInfor,FactoryName,SellDate)
values(@NewItemType,@NewItemName,@NewItemCost,@NewItemInfor,@Factory
Name,@NewSellDate)
if @@Error<>0 goto problem
insert into ItemStored(ItemName,ItemCost)
values(@NewItemName,@NewItemCost)
if @@Error<>0 goto problem
commit transaction
return 0
problem:
Rollback transaction
print 'Insert Failed!'
return 1
GO
```

这个存储过程实现的功能是将新引进的商品的信息同时插入到商品信息表和商品库存表中，并对其插入的成功性作出判断，返回结果值。判断插入成功性语句为 if @@Error<>0 goto problem，出错将显示 problem 所定义的事件，即返回语句'Insert Failed!'

图示：



4.2.3 前台对后台的访问的实现

本设计采用 ADO.NET 技术处理页面与后台数据库的连接和操作，以下是具体设计：

DataAdapter 在连接到数据库时工作。单个 DataAdapter 的作用是使用数据库中的数据填充某个 DataTable，或将 DataTable 中的更改写回到数据库，或者二者兼而有之。

DataAdapter 要求 Command 对象执行各种数据库操作。Command 对象存放 SQL 语句或指定数据访问实现方法的存储过程名称。每个 DataAdapter 有四个属性，指定用于四种数据访问类型之一的命令对象。

- SelectCommand：　用于从数据库中选择数据。
- UpdateCommand：　用于更新数据库中的现有记录。
- InsertCommand：　用于向数据库中插入新记录。
- DeleteCommand：　用于删除数据库中的现有记录

可以向数据库发出命令，以针对数据存储区执行操作，它包括可向数据库发出的任何语句。可以使用 SqlCommand 类获取向数据存储区发出的命令。在 ADO.NET 中，只有 Command 对象 SqlCommand 运行命令。对 System 和 System.Data 命名空间使用 using 语句，这样，在后面的代码中就无需限定这些命名空间中的声明了。也可以包含 System.Data.SqlClient，具体取决于所使用的命名空间。

```
using System;
using System.Data;
using System.Data.SqlClient;
```

在创建与数据库的连接之前，必须具有一个连接字符串。连接字符串包含建立数据库连接所需的所有信息，包括服务器名称、数据库名称、用户 ID 以及密码。例如，以下连接字符串指向运行 SQL Server 的本地计算机：

```
sever=<SeverName>;userID=<UID>;pwd=<PWD>;database=<DBName>
```

然后使用 Visual Studio 创建一个静态类 SqlConn。在该类中，声明一个字符串变量并存储数据库的相应连接字符串。

```
public class SqlConn
    {
    Public static string ConStr = "server=JACKY;uid=Jacky;pwd=830101;
    database=dzsw";
        public SqlConn()
        {
            }
     }
```

使用此连接字符串，新建 SqlConnection 对象，并调用其 Open 方法以建立与数据库的连接：

```
SqlDataAdapter dsCommand = new SqlDataAdapter();
dsCommand.DeleteCommand = new SqlCommand();
dsCommand.DeleteCommand.Connection=new
SqlConnection(SqlConn.ConStr);
dsCommand.DeleteCommand.Connection.Open();
```

创建 SqlCommand 对象，并传入要运行的命令以及在上一步中创建的连接

对象。

```
SqlCommand command = dsCommand.DeleteCommand;
command.CommandText = "DeleteTempList";
command.CommandType = CommandType.StoredProcedure;
command.Parameters.Clear();
command.Parameters.Add("@CustomerID",i);
```

创建 SqlCommand 对象之后，可调用 ExecuteNonQuery 方法来运行它所表示的命令。ExecuteNonQuery 用于不返回任何结果的命令（如 DELETE、UPDATE 和 INSERT 语句）。如果运行该语句时没有引发任何异常则说明已对数据库成功执行了该命令。

```
command.ExecuteNonQuery();
```

4.4.4 页面间参数传递的实现

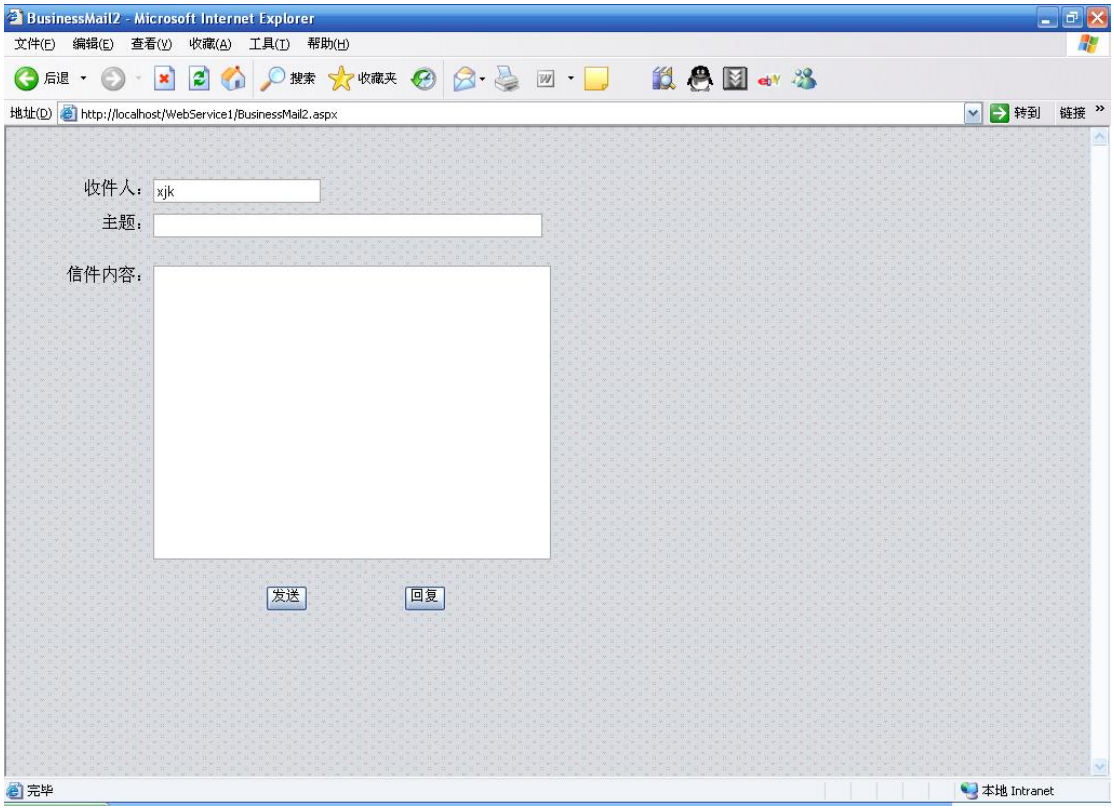
在页面传参方面，使用了 Cache 缓存技术，即在前一页面将想要传递的参数存入缓冲区，在页面跳转后，再将相应的参数取出使用。

图示：


```
if(e.CommandName=="watch")
  {     send="a";
        Cache["Send"]=send;
       Cache["fr"]=e.Item.Cells[2].Text.ToString();
        Cache["wr"]=e.Item.Cells[3].Text.ToString();
```

```
        SqlDataAdapter dCommand = new SqlDataAdapter();
       dCommand.UpdateCommand = new SqlCommand();
 dCommand.UpdateCommand.Connection = new SqlConnection(SqlConn.ConStr);
      dCommand.UpdateCommand.Connection.Open();
      SqlCommand command = dCommand.UpdateCommand;
      command.CommandText = "UpdateCustomerMail";
      command.CommandType = CommandType.StoredProcedure;
      command.Parameters.Clear();
      command.Parameters.Add("@CustomerID",s);
      command.Parameters.Add("@Administor",e.Item.Cells[2].Text.ToString());
      command.Parameters.Add("@WriteDate",e.Item.Cells[3].Text.ToString());
      command.ExecuteNonQuery();
     dCommand.UpdateCommand.Connection.Close();
      Response.Redirect("BusinessMail2.aspx");}
```

上述代码先将要传递的参数放入 Cache 中。




```
send=Cache["Send"].ToString();
      if(send=="a")
{     s=Convert.ToInt32(Cache["Text"]);
      fr=Cache["fr"].ToString();
      wr=Cache["wr"].ToString();
      if(!this.IsPostBack)
      {SqlDataAdapter dCommand = new SqlDataAdapter();
      dCommand.SelectCommand = new SqlCommand();
```

```
dCommand.SelectCommand.Connection = new SqlConnection(SqlConn.ConStr);
dCommand.SelectCommand.Connection.Open();
SqlCommand command2 = dCommand.SelectCommand;
command2.CommandText = "GetMailContent2";
command2.CommandType = CommandType.StoredProcedure;
command2.Parameters.Clear();
command2.Parameters.Add("@CustomerID",s);
command2.Parameters.Add("@Administor",fr);
command2.Parameters.Add("@WriteDate",wr);
SqlDataReader reader =    command2.ExecuteReader();
bool i = reader.HasRows;
reader.Read();
this.TextBox1.Text=reader.GetString(0).ToString();
this.TextBox2.Text=reader.GetString(1).ToString();
reader.Close();
dCommand.SelectCommand.Connection.Close();}                }
else if(send=="b")
{this.Label3.Visible=true;
this.TextBox3.Visible=true;}// 在此处放置用户代码以初始化页面
      }
```

上述代码表示将前一页面中存入 Cache 的参数取出，并引用于当前页面的存储过程参数输入。

第五章 工作总结

5.1 设计总结

本文的课题电子商务是未来商业经营的主要形式，是现代业参与国际市场竞争的最重要的手段。“科技是第一生产力”，我的设计目的是将最新的电脑技术带入到传统的商业模式中，以期实现一个现代化的购物方式－－电子商务网站。

电子商务网站如同 Internet 上的一个个企业或商业机构，通过它，电子商务才能得以实施和运作。与普通的网站相比，电子商务网站的主要区别体现在数据处理和数据传输要求更高、数据流程更复杂。网上进行商品交易，商务网站不仅要接收与处理量零散而复杂的商务数据和信息，而且要保证数及信息传输的安全性。实践表明，电子商务网站不是网页的简单堆砌，其构建是为实现企业网上营目的服务的，所以，电子商务网站的功能既要大又要满足商业流程。因此，电子商务网站的设与管理是一项复杂的系统工程，是对商务流程进行整合和对内外部信息进行集成的工作，也是网络信息资源开发与管理的过程。本文从体上全面论述了一个电子产品交易网站的设计和实现，其中，规划与设计网站是企业通向电子商务的基础，管理网站是电子商务有效运转的保障。

本文所实现的电子商务网站是由前台系统和后台系统两部分构成的。前台系统是供商品销售使用的浏览器界面,在这里可以注册会员、浏览商品、购物、留言及网上支付。后台系统主要是进行商品信息的管理、发布和修改,同时要进行会员管理、商品配送、报表统计,以及系统运转及安全有效的账号系统。网站完成后，客户可以运行一个完整的注册和购物流程。享受网络购物所带来的方便和快捷。

本文所论述的电子商务网的设计采用了传统的两层结构模式，通过将整个系统划分为不同的逻辑模块，从而大大降低了应用程序开发维护的成本，提高了系统模块的复用性，同时使系统的可扩展性大大增强，由于 ASP.NET 自身的优势，使得网站的访问效率大大提高，另一方面，由于恰当地使用了存储过程，使得访问效率不再是问题。

在本次商务网站的开发过程中，我体验到了.NET 技术带来的无限魅力。.NET 技术正以不同方式影响着企业，个人和开发人员。对于个人，这些变化将产生及其个性化，集成的计算体验，对于企业和开发人员，它将改变生成软件和销售产品的方式，使 IT 成为企业成功的重要因素并映入新的业务模式。这次设计也使我更深入的掌握了 SQL SEVER 等开发工具的实际开发使用，在技术上有了新的提高。并且使得自己具备了独立开发一个基本功能晚上的商务网站的经验。对于以后的工作是一次很好的锻炼。

5.2 不足之处

在此次毕业设计中，由于技术和经验上的欠缺，或多或少出现了一些不足之处，比如电子商务一些需求功能上的缺失，或是已设计的功能的设计思路还不够完善，实现的方法也不够正式。

在前台页面的设计上，由于我在网页美工上的经验不足，页面的美观度不够，而且对于控件的排放上也没有特别正规的体系，造成网站的吸引力有所降低。商品的分类不够丰富，商品的浏览方式不够便捷，对 Panel 控件等其他 ASP.NET 控件的使用率不够，这样就造成了页面的繁琐，

在后台数据库的设计上也有着缺点，在前期的设计中对表的设计考虑的不够详尽，使得表的某些属性出现了冗余，在一些表的互相关联上做的也不是特别合理，同时也有一些现实中商业属性的缺失。在存储过程的编写中，相对于真正的商务网站显得比较简单，对多表操作的存储过程不多，使得功能上受到了一定的限制，不能完全实现自己的构思。

在功能的设计也存在着与许多正规电子商务功能的差距，一些功能的设计不够人性化。仓库的物流系统也做的不够详尽，报表的属性比较简单。另外，还存在着一些想要实现而本次设计中未能实现的功能，比如大量的模糊网页查询技术，人事管理的安全性机制，和对缓冲池的使用的缺失。在页面间传递参数的方法比较单一，其实还可加入 COOKIE 和 SESSION 方法。

在结构的设计上，由于使用了传统的两层结构，所以页面代码的冗余有所增加，而且如果对于正式的商务网站，两层架构在安全性的封装上也不尽人意。当客户端数目激增时，服务器端的性能会因为负载过重而大大衰减；一旦应用的需求发生变化，浏览器端和服务器端的应用程序都需要进行修改，给应用维护和升级带来了极大的不便，同时，大量的数据传输增加了网络的负载等等。

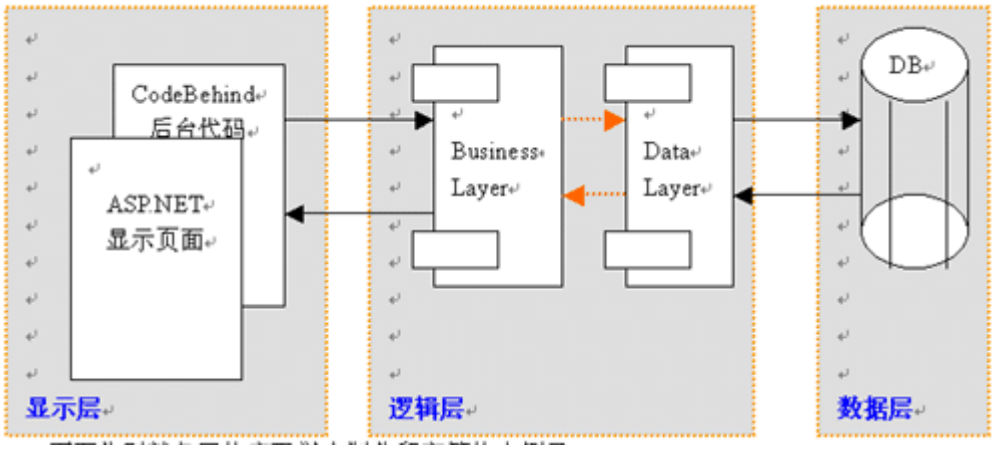
5.3 未来研究方向

电子商务的未来发展方向是将注重跟踪国内外发展动态，在对电子商务研究领域进行相关理论研究的基础上，注重改革创新，围绕着电子商务平台、安全技术、法律与心理等方面进行开拓性、创新性研究。

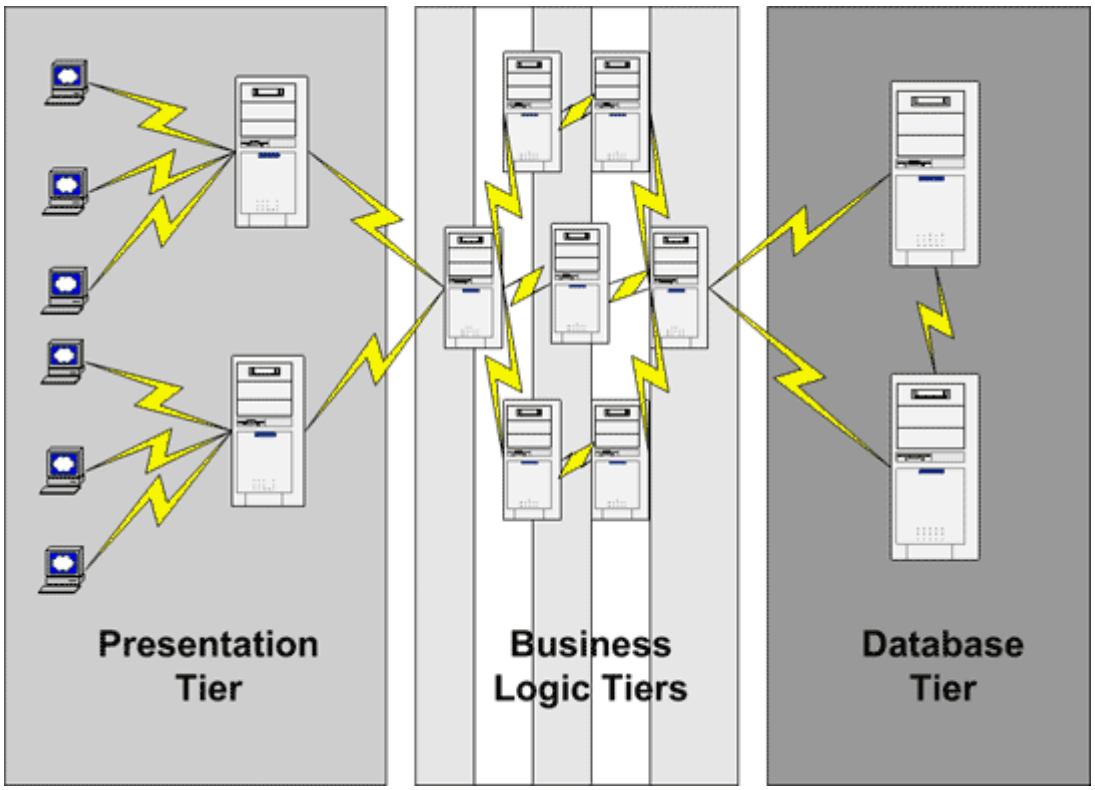
基于上文的发展方向和所提到的本次设计的不足之处，我的未来研究方向订为：

加强网页的美工设计，完善数据库的设计，减少表中属性的冗余度，使得表之间的关联清晰，提高存储过程的功能，增加 ASP.NET 中的其他控件和技术的利用率，以完善网站的功能，让网站更接近和达到现实电子商务的需求。最重要的一点是在网站的结构设计上，将舍弃传统的两层开发模式，采用全新的三层甚至 N 层结构。

三层结构（N 层结构）的研究：



三层结构图例



N 层结构图例

三层结构模型是企业级应用软件开发模型。它不是物理上的三层,而是逻辑上的三层,即表示层业务层、数据层,如图 1 所示。

其中:表示层是人机交互界面,负责处理用户的输入和向用户的输出。

业务层也叫事务逻辑层或中间层。这一层包括数据访问子层和业务服务子层。

数据访问子层主要解决如何从数据库中提取和保存数据;业务服务子层主要处理一些业务逻辑和业务规则。在许多应用程序中,程序员往往将业务逻辑和数据访问放在同一逻辑层中,这样虽然可以达到预期效果,但通常并不实用。分为两子层后,当数据库对象发生改变,如从 SQL Server 数据库改为从 Oracle 数据库中提

取数据时,只需简单地修改数据服务层组件,完全不用担心对应用程序的影响。

数据层通俗一点讲,就是我们传统的数据库管理系统,如 SQLServer、Oracle 等。每一层只能与它的直接相邻层通信。例如表示层只能与业务服务层通信,不能与数据层或数据访问层通信。

三层结构与传统的两层结构相比体现了集中式计算的优越性：具有良好的开放性；减少整个系统的成本，维护升级十分方便；系统的可扩充性良好；系统管理简单，可支持异种数据库，有很高的可用性；可以进行严密的安全管理。

除了三成结构外，在未来的研究中还必须提高设计的安全性。

授权认证：防止黑客和非法使用者无法进入网络并访问信息资源，为特许用户提供符合身份的访问权限并且有效的控制这种权限。

数据加密和信息传输加密 ：利用加密技术可以解决以下一些环节上的安全问题：钥匙管理和权威部门的钥匙分发工作、保证信息的完整性、数据加密传输、密钥解读和数据存储加密等。

防火墙 通常网络采用防火墙的主要目的是对系统外部的(一定的空间时间)范围内访问者实施隔离的一种技术措施。在网络系统中，使用防火墙可以起到如下作用：

访问控制 对内部与外部，内部不同部门之间实行的隔离

权限认证 授权并对不同用户访问权限的隔离；

安全检查 对流入网络内部的信息流进行检查或过滤，防止病毒和恶意攻击的干扰破坏(安全隔离)。

加密 提供防火墙与移动用户之间在信息传输方面的安全保证，同时也保证防火墙与防火墙之间信息安全。

对网络资源实施不同的安全对策，提供多层次和多级别的安全保护。

集中管理和监督用户的访问。

报警功能和监督记录。

在以上的技术基础上，把电子商务网站开发成更快捷，更方便，更安全，更具人性化的购物模式。

致谢

本此设计从最初的准备工作到最终的完成，我得到了很多人的支持也帮助，在这里我要衷心的感谢他们对我自始至终的支持。

首先感谢我本次设计的指导老师陈任老师。我对这次设计的提出得到了陈老师的支持，并且他对我的设计思想提出了很多宝贵的意见。在毕业设计期间，陈老师经常抽空与我会面讨论毕业设计的详细事宜，不断从旁指导和督促我完成工作。我们还经常通过电子邮件保持联络，同时他也十分关心我的毕业设计进程，让我能够按时按规定地完成毕业设计。在毕业论文的制作过程中，陈老师也对我提出了中肯的评价和修改意见，让我能够顺利完成这篇论文。

其次是要感谢这四年来教导过我的诸位老师，正是得益于他们传授的知识与经验，我才能拥有完成此次毕业设计的能力。

再次我还要感谢我身边的同学。他们在我的毕业设计期间给予了我很多帮助，也协助我解决了许多技术上遇到的难题。

最后我要感谢在百忙中抽出时间和精力来评阅本篇论文的老师们，衷心感谢你们辛勤的工作。

当然我还要感谢上海大学，我的母校，是你提供了我学习的园地，帮助我不断成长为一名合格的毕业生，让我具备了踏入社会工作的能力。

参考文献

[一]P S Wang, S Katila，Web 设计与编程导论（英文影印版），北京：高等教育出版社，2004.8，第一版

[二]J Buyens 等，WEB 数据库程序设计（.NET 版），北京：清华大学出版社，2002.10，第一版

[三]A Homer. D Sussman，ASP.NET 1.1 高级编程，北京：电子工业出版社，2001.02，第一版

[四]唐大仕 C＃程序设计教程，北京：清华大学出版社，2003.08，第一版

[五]白以恩，韩冬梅，王进，电子商务网站的建设，哈尔滨商业大学学报（自然科学版），第 19 卷，第一期，36－39 页，2003.02

[六]夏阳，张强，陈小林，基于 ASP.NET 的电子商务网站开发和设计，计算机工程与设计，第 25 卷，第 11 期，2027－2029 页，2004.11

[七]赵其真，中国电子商务发展之我见，河北青年干部管理学院学报，第三期，78－79 页，2004.09

[八]班奕，我国电子商务发展中的问题及解决对策，西安航空技术高等专科学校学报，第 22 卷，第 6 期，21－23 页，2004.11

[九]张红，孙富元，关于电子商务的研究，天津理工学院学报，第 20 卷，第 4 期，103－105 页，2004.12

[十]黄颖，谢忠，ASP.NET, SQL SEVER 技术在动态网站开发中应用，编程园地

[十一]唐金文，ADO.NET 访问 WEB 数据库方法的研究，曲靖师范学院学报，第 23 卷，第 6 期，53－56 页，2004.11

附录

英文翻译

DataReader Operations

The next few sections will highlight the main functionality of the datareaders. Among other things we will see how we can traverse results sets, how data can be retrieved, and how to get detailed information about the result set we're working with. To end this section, we'll bring together a number of the most important aspects of the datareader by creating a simple application.

Let's start by looking at the basics of using a datareader.

Using DataReaders

Creating a DataReader

It is not possible to create a datareader explicitly. The constructors for datareader objects are marked as public internal, which means that only objects in the same assembly as the DataReader classes can create a DataReader. To gain access to a DataReader we must first of all have a Command object. We can create a useable datareader through the Command object.

As we saw in chapter 2, a command object represents a SQL statement or a stored procedure. In ADO.NET there are two Command objects: t hese are SqlCommand and OleDbCommand. All command objects in ADO.NET implement the IDbCommand interface. The IDbCommand interface provides a template for all Command objects. Through this interface we can look at the SQL command to be executed, any command parameters, the database connection, and so on. It's through one of the methods on this interface that we can create a datareader. The method in question is called ExecuteReader. We'll have a quick look at the ExecuteReader method next.

The ExecuteReader Method

This is where it all begins, with a call to ExecuteReader. This method executes the SQL or stored procedure that the Command object represents and returns a datareader object.

There are two overrides of the ExecuteReader method: the first takes no parameters and the second takes an enumerated value of CommandBehavior type that is defined in System.Data. The table below shows the possible values and their use:

Value Use

CloseConnection Closes the connection when the datareader is closed.

KeyInfo The query returns column and primary key information. Using KeyInfo with the SQL Server Managed Provider causes a FOR BROWSE clause to be appended to the statement being executed. This may have side effects, such as interference with the use of SET FMTONLY ON.

SchemaOnly The query returns column information only and does not affect the

database state.
SequentialAccess The results are read sequentially to the column. This enables applications to
read large binary values using methods such as GetBytes.
SingleResult The query returns a single result. Execution of the query may affect the database state.
SingleRow　The query expects to return a single row. This may improve performance because providers can use this information to optimize the command.
So we can either get default behavior by using the parameterless method:
OleDbDataReader reader = command.ExecuteReader ();
Or we can be more specific in what we want and use the second implementation like this:
OleDbDataReader reader = command.ExecuteReader
(CommandBehavior.SingleRow);
Chapter 4
We can also use combinations of CommandBehavior values. So, for example, we could write:
OleDbDataReader reader = command.ExecuteReader
(CommandBehavior.CloseConnection | CommandBehavior.SingleResult)
This particular piece of code would cause the Connection object to close when the datareader closes and
specifies that only a single result set should be returned regardless of how many result sets are found.
I'm not going to dwell any further on this method or the Command objects.

Creating and Using a DataReader

Now, let's have some code. We've already briefly seen an example of a datareader being created, but now we'll go into it in slightly more depth. Here is a simple case using a datareader:

```
OleDbConnection conn = new OleDbConnection
( "Provider=SQLOLEDB;Data Source=localhost;Initial Catalog=Northwind;"
+ "Integrated Security=SSPI;" );
OleDbCommand command = new OleDbCommand ( "select * from CUSTOMERS",
conn );
conn.Open ();
OleDbDataReader reader = command.ExecuteReader ();
while ( reader.Read () )
{
// code in here
}
reader.Close ();
conn.Close ();
```

For demonstration purposes, this example uses the OleDb managed provider, even though it's accessing a SQL Server 2000 database. The first thing we do is to create an OleDbConnection object to give us the connection to the database.

Next we construct the Command object that will be used to execute the command. Again, because we're using the OleDb provider, we're using the OleDbCommand class. We construct the OleDbCommand object by passing it the SQL we want executed and the connection. Next, the connection to the database is opened. If the connection isn't opened before we call ExecuteReader then an InvalidOperationException will be raised. This behavior differs from the data adapter objects. A data adapter will open a connection when executing a command if it's not already open and then close it when finished. The connection **must** always be open before the datareader is created using ExecuteReader. Finally, with the connection open we can execute the SQL against the database; this returns us (in this case) an OleDbDataReader.

Using the Read method on the datareader, we can now traverse the records generated by the SQL. The code example above simply keeps calling Read until it returns false, at which point ther e are no more records in the result set. The final two method calls close the datareader and the connection.

Using DataReaders

It's worth mentioning the Close method on the datareader. Unless CommandBehavior.CloseConnection was passed to ExecuteReader, this doesn't close t he connection that the datareader is using: it simply closes the datareader itself. In practical terms, this means that other objects can now use the connection that the datareader was using. The sharing of an existing connection is more desirable than creating a new one. Creating and opening connections is an extensive process that should be avoided wherever possible. If we tried to use the connection without closing the datareader, we'd get an InvalidOperationException telling us that the connection was already being used.

Call the Close method before trying to use the connection again. An open datareader going out of scope **does not** free the connection. So, to summarize this section, we should note that:

- Datareaders can only be created using command objects
- The connection must always be open before we call ExecuteReader
- Use the ExecuteReader method to create a datareader
- Always call Close on the datareader when it's finished with; failing to do so will prevent other objects using the connection

Now we've seen how we go about creating a datareader, we will look at what we can do with the datareader in more detail.

Simple Data Retrieval With the DataReader

Now we'll move on to a fuller example. We're going to create an OleDbCommand object that selects all the records from the Orders table in the Northwind database. Using this Command object, we'll create an OleDbDataReader, read all the records, and dump them to the console. So, let's have a look at the code (this code is in the SqlDataRetrieval and OleDbDataRetrieval projects).

```
OleDbConnection conn = new OleDbConnection ("Provider=SQLOLEDB;Data
Source=localhost;Initial Catalog=Northwind;Integrated Security=SSPI;");
OleDbCommand selectCommand = new OleDbCommand("select * from ORDERS",
```

```
conn);
conn.Open ();
OleDbDataReader reader = selectCommand.ExecuteReader ( );
while ( reader.Read () )
{
object id = reader["OrderID"];
object date = reader["OrderDate"];
object freight = reader["Freight"];
Console.WriteLine ( "{0}\t{1}\t\t{2}", id, date, freight );
}
reader.Close ();
conn.Close ();
```

As we have seen before, the first thing we do is create a Connection object. In this case we're using the OleDb managed provider, so we create an OleDbConnection. Next, we create the SqlCommand object. This represents the Transact -SQL command we wish to execute, and will y ield the OleDbDataReader to us.

Chapter 4

142

Before we call ExecuteReader, we open the connection; if we fail to do this, the SqlCommand object will raise an exception. Once ExecuteReader is called (assuming everything worked) we're in a position to read directly from the Northwind database. We traverse over the records by using the Read method on the DataReader. This has the effect of moving the DataReader to the next record. When a DataReader is first returned, it is actually positioned before the first record in the result set. This being the case, we must call Read before trying to access any of the values in the DataReader columns. Failure to do so raises an exception.

Read must always be called before any of the data access methods are used.

Now we are into the while loop. Each time Read is called we advance to the next record. If Read is called and there are no more records left, then false is returned and the while loop terminates.

Let's look at the code in the loop. The first three lines are the most interesting. Using the Item indexer property, we retrieve values for three of the columns in the datareader. Any object that supports an indexer property essentially allows us to treat that object as an array. If we consider how we might retrieve an element from an array:

```
int []values = new int[] {1, 5, 9, 15, 26};
int theValue = values[3];
```

We can see that the syntax values[3] isn't that different to our previous example where we use reader["OrderID"]. The main obvious difference is that when we retrieve the integer from our array we specify the element position, in this case element 3 – which would give us the value 15. But when we use the datareader indexer we specify the column name – OrderID. We could, if wanted to, specify the ordinal value of the column OrderID, which is 0. To do this we'd write:

```
object id = reader[0];
```

It has the same effect as specifying the column name (though as we'll see at the end of chapter it does improve performance). The ordinal values of columns are always zero-based, so the first column in the table has an ordinal value of 0, the next column has a value of 1 and so on.

Note that the indexer has returned us values of type object. Using the datareader indexer always returns object types. This is because an indexer has to be able to return any type of data that can be stored in a database and in .NET the object type is the base class of all objects. By returning an object an indexer is capable of returning anything.

In this example, the final line of the loop outputs the values to the console. In many cases we'd probably want to access the data in a type-safe manner, which we will learn more about later.

The final two lines close the datareader and the connection. If we don't close the datareader, no other objects will be able to use the connection. Connections are scarce resources and shouldn't be held on to for a moment longer than they're needed.

Always close the DataReader as soon as it's finished with.

Using DataReaders

If we run the program, the output looks like this:

The output shows a list containing three columns. In order, it shows OrderID, OrderDate, and finally the Freight column from the Orders table.

That's how we execute SQL statements – as you can see, it's pretty straightforward. Now, we'll move on to executing stored procedures.

Executing Stored Procedures with a DataReader

We have seen how to execute simple SQL statements against the database, so now we're going to look at how we use the command object and the datareader to execute a stored procedure. We'll be using the CustOrdersDetail stored procedure that comes with Northwind in this example. The stored procedure SQL is shown below; don't worry too much about its contents, all we need to know is that it takes an order ID and returns the details for that order:

```
CREATE PROCEDURE CustOrdersDetail @OrderID int
AS
SELECT ProductName,
UnitPrice=ROUND(Od.UnitPrice, 2),
Quantity,
Discount=CONVERT(int, Discount * 100),
ExtendedPrice=ROUND(CONVERT(money, Quantity * (1 - Discount) *
Od.UnitPrice),
2)
FROM Products P, [Order Details] Od
WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
```

The Command object does most of the work when executing stored procedures. We still use the datareader in exactly the same way we normally would. The entire code for the example can be found in the SqlStoredProcedureExample project. As usual, the code for the OleDb provider is almost identical and this can be found in the

OleDbStoredProcedureExample. The listing below shows the main method of the SqlClient code:

```
SqlConnection conn = new SqlConnection ( "Initial Catalog=Northwind;Data
Source=localhost;Integrated Security=SSPI" );
SqlCommand storedProcCommand = new SqlCommand ( "CustOrdersDetail",
conn );
storedProcCommand.CommandType = CommandType.StoredProcedure;
storedProcCommand.Parameters.Add ( "@OrderID", 10248 );
Chapter 4
conn.Open ();
ArrayList rowList = new ArrayList ();
SqlDataReader reader = storedProcCommand.ExecuteReader ( );
while ( reader.Read () )
{
object []values = new object[ reader.FieldCount ];
// retrieves all the fields values into an array of objects
reader.GetValues ( values );
rowList.Add ( values );
}
reader.Close ();
conn.Close ();
foreach ( object []row in rowList )
{
foreach ( object column in row )
Console.WriteLine ( column );
Console.WriteLine ( "\n" );
}
```

Now we'll look at the code in more detail. First we'll have a look at how we create the Connection object. In this example we're using the SqlCient managed provider, so we'll be creating a SqlConnection object:

```
SqlCommand storedProcCommand = new SqlCommand ( "CustOrdersDetail",
conn );
storedProcCommand.CommandType = CommandType.StoredProcedure;
storedProcCommand.Parameters.Add ( "@OrderID", 10248 );
```

The first line constructs the SqlCommand object, and we pass in two parameters. The second parameter is the usual connection object; it's the first parameter that's more interesting. Instead of a SQL string, we've passed in the name of the stored procedure as it exists in the Northwind database – in this case CustOrdersDetail. This tells the SqlCommand object which stored procedure to execute.

The second line tells the Command object that a stored procedure is going to be executed. We do this by setting the CommandType property to StoredProcedure. If we didn't do this, the Command would try to execute as a normal SQL statement and a syntax exception would be raised.

Finally we need to add some parameters to the command. The CustOrdersDetail

stored procedure expects to receive the ID of a customer order, so we add a new parameter to the SqlCommand's parameter collection.
The name of the parameter is @OrderID and the order we're interested in is 10248.
Note the use of the @symbol? Ultimately, all the information in the Command object is constructed to make a SQL statement. The @symbol in a SQL statement is used to denote a parameter.
Now we're ready to create the datareader:
Using DataReaders

```
ArrayList rowList = new ArrayList ();
SqlDataReader reader = storedProcCommand.ExecuteReader ( );
while ( reader.Read () )
{
object []values = new object[ reader.FieldCount ];
reader.GetValues ( values );
rowList.Add ( values );
}
```

I've introduced a couple of new properties and methods this time, so we'll have a look at them first. The first line in the while loop is creating an array of objects. Each object in the array represents a column in the datareader. We know how many columns are in the datareader by using the FieldCount property.
Now we have the array of objects we need to get some values. If we wanted, we could get each value individually and add it to the array; another way is to use the GetValues method. This method will populate our object array with the column values currently in the datareader. The tradeoff for this ease of use is the fact that we lose the type safety, but, as usual, that's a choice that has to be made in the context of the development situation.
If you've been wondering why we created an ArrayList at the top of code sample, you're about to find out!
In the last line of code we add the object array to the ArrayList object called rowList. Why are we doing this? Well, bear in mind that while the datareader is open. It's effectively "blocking" the connection so that no other objects can access it. It is often a good idea to cache away the values retrieved from the datareader and process them after the datareader is closed. The less time we're blocking the data connection the better.

Process datareader results after the datareader is closed.

Finally, after we've closed the datareader and the connection, we can output the retrieved values to the screen.
We use the following code to do this:

```
foreach ( object []row in rowList )
{
foreach ( object column in row )
Console.WriteLine ( column );
Console.WriteLine ( "\n" );
}
```

This code is fairly straightforward. By this point, we have an ArrayList containing arrays of objects, and each object in the array represents a value from a column in the datareader. So, in the outer loop we get the array of column values, and in the inner loop we iterate over the column values and output them to the Console. Finally, we write a blank line to the screen: this just makes the output clearer to see.

中文译文

DataReader 操作

下面几段将会着重介绍 Datareaders 的主要功能。 另外我们将会见到我们能如何遍历结果集，数据如何被取回，和该如何得到被处理的结果集的详细信息。在结尾处，将通过建立一个应用程序来综合这些关于Datareader 的知识。让我们藉由看着使用 Datareader 的基础知识开始。

使用 DataReaders

创造 DataReader

显示地创建一个 Datareader是不可能的. Datareader 对象的构造器被标志成 Public internal，意指只有和 DataReader 类处在同一Assembly中的对象才能创造 DataReader。 未来能够创造一个 DataReader，首先要有一个Command对象， 我们能通过这个Command对象创造一个可用的 Datareader。

就如我们在第 2 章中见到的，一个Command对象表示一个 SQL 语句或一个存储过程。在 ADO.NET 中有两种Command对象:它们是 SqlCommand 和 OleDbCommand。所有的Command对象在ADO.NET 中都实现了 IDbCommand 接口。 IDbCommand 接口提供一个模板给全部Command对象。通过这一接口我们能查看将要被运行的SQL 指令，任何的命令叁数，数据库连接。通过该接口的某种方法我们能创造一个 Datareader。 这个方法叫做 ExecuteReader。 我们将会在下面简要介绍 ExecuteReader 方法。

ExecuteReader *方法*

藉由对一个ExecuteReader 的调用是所有任务开始的地方，这一方法执行 Command对象表示的 SQL 语句并且返回一个 Datareader 对象。

ExecuteReader方法有两种形式：一种没有任何参数，另一种是利用定义在 System.Data中的CommandBehavior 类型的枚举值作为参数。下表给出了可选用的值和使用方法：

参数使用

CloseConnection	结束连接当 Datareader关闭的时候。
KeyInfo	查询返回列和主键信息。利用SQL Server Managed Provider使用KeyInfo将一个For Browse子句添加到被执行的语句后面。 这可能有副作用，比如和SET FMTONLY ON 的使用冲突。
SchemaOnly	查询只返回列信息，不影响数据库状态
SequentialAccess	结果顺序地被读入列中。这使的应用程序能够利用读取最大的二进制数的方法 ，例如GetBytes。

SingleResult	查询返回一个单一结果。查询的执行可能影响到数据库状态。
SingleRow	查询只返回一行结果。由于提供者可利用该信息优化命令，所以能提高性能。

我们也可以使用默认的无参数地方法
OleDbDataRead reader = command.ExecuteReader ();
也可以用第二种方法明确的指出想要的参数来使用该方法，例如：
OleDbDataRead reader =
command.ExecuteReader(CommandBehavior.SingleRow);
第四章
另外，我们也能使用 CommandBehavior 组合值。 例如：我们可以写:
OleDbDataReader reader = command.ExecuteReader
(CommandBehavior.CloseConnection | CommandBehavior.SingleResult)
这段代码将在Datareader 关闭的时候使Connection对象关闭。并且无论找到多少结果集，都只返回一个结果集。
在此不再深入讨论这种方法或Command对象。

创建并使用 DataReader

现在，让我们分析下面代码。 我们已经简要分析了如何创建Datareader 的方法，现在将更加深入的学习它。这里是一个简单的使用 Datareader的例子:

```
OleDbConnection conn = new OleDbConnection
( "Provider=SQLOLEDB;Data Source=localhost;Initial Catalog=Northwind;"
+ "Integrated Security=SSPI;" );
OleDbCommand command = new OleDbCommand ( "select * from CUSTOMERS",
conn );
conn.Open ();
OleDbDataReader reader = command.ExecuteReader ();
while ( reader.Read () )
{
// code in here
}
reader.Close ();
conn.Close ();
```

为达到阐述的目的，虽然本例是对SQL SEVER 2000数据库进行存取，我们仍将使用OLEDB受管提供者。我们做的第一件事是创建一个OleDbConection对象以便连接到数据库。
然后创建用以执行命令的Command对象。由于要使用OleDb 供给者，所以需要利用 OleDbCommand 对象。我们可以藉由传递所要执行的 SQL语句和连接来创建OleDbCommand对象。接下来，打开连接数据库的连接。如果在调用在ExecuteReader前连接没有打开，则会引发 InvalidOperationException 。该行为不同于数据适配器行为，在用ExecuteReader创建DataReader前连接必须保持打开状态。最后，在连接打开的状态下，对数据库执行SQL，这将返回(本例中) OleDbDataReader。
在 Datareader里，Close方法是值得注意的。除非CommandBehavior.closeConnection被传到ExecuteReader，否则它不会关闭

Datareader正在使用的连接，它仅关闭DataReader本身。这意味着其他的对象仍可以使用DataReader所用的连接。这种连接的共享比创建一个新的连接更受欢迎。创建并打开一个连接是应该经可能的避免。如果我们没有关闭DataReader就去使用连接，就将会得到一个InvalidOperationException，提示连接正在被使用。
在试图再次使用连接前应该调用Close方法。一个打开的DataReader即使离开了它的活动范围也并不释放连接。因此，综合本节内容，需要注意下列问题：
DataReader只能利用命令对象创建
在调用ExecuteReader之前连接必须是打开的
数用ExecuteReader方法创建一个DataReader
在要关闭DataReader的时候应该调用Close，如果不这样做将使得其他对象不能使用该连接
现在，大家应该明白怎么创建一个DataReader了，后面我们将急需介绍DataReader的更多细节。

利用DataReader 进行简单数据查询

现在我们来看一个完全的实例。我们要创建一个Northwind 数据库的Orders表选取所有的记录的OleDbCommand对象。然后使用该Command对象创建一个 OleDbDataReader ，读取所有的记录，并将它们输出控制台。因此，让我们看下列代码：

```
OleDbConnection conn = new OleDbConnection ("Provider=SQLOLEDB;Data
Source=localhost;Initial Catalog=Northwind;Integrated Security=SSPI;");
OleDbCommand selectCommand = new OleDbCommand("select * from ORDERS",
conn);
conn.Open ();
OleDbDataReader reader = selectCommand.ExecuteReader ( );
while ( reader.Read () )
{
object id = reader["OrderID"];
object date = reader["OrderDate"];
object freight = reader["Freight"];
Console.WriteLine ( "{0}\t{1}\t\t{2}", id, date, freight );
}
reader.Close ();
conn.Close ();
```

就如我们前面所介绍的，我们做的第一件事是创建一个Conection对象。在这个例子中，我们将使用 OleDb受管供给者，因此，我们创建了 OleDbConnection。然后，我们创建 SqlCommand 对象。它表示所要运行的T － SQL 命令，并且将生成 OleDbDataReader。
在调用ExecuteReader前需要打开连接，如果我们无法打开，SqlCommand 对象就会产生异常。一旦 ExecuteReader 被调用了（ 假设一切情况正常），就可以从 Northwind 数据库中直接读取数据。
我们使用DataReader的Read方法对记录进行遍历。这将达到将DataReader移动到下一条的效果。当 DataReader 首先返回的时候，它实际上位于结果集的第一条记录之前。这种情形下，我们一定在尝试访问任何DataReader 列的值前调用 Read。不这么做的就会引发异常。

读取操作必须在任何数据访问方法被使用之前调用。

现在，看看WHILE循环。每次调用Read，都向前跳到下一记录。如果调用Read而结果集中没有了记录，那么将返回False，while循环将终止，

再看看循环中的代码。前三行是最有意思的。用Item查询属性，我们可以在Datareader中为三个列获取值。任何支持索引属性的对象在本质上允许我们视对象为数组。我们考虑一下，怎么从数组中检索元素：

从排列：

```
int []values = new int[] {1, 5, 9, 15, 26};
int theValue = values[3];
```

我们能见到，语法Values[3] 与先前使用reader[“order ID”]的示例并没有什么区别，最为明显的区别是当从数组值中获取整数时需要指定元素位置，本例中，元素3被赋予值15。但是在使用DataReader索引时，需指定列名（orderID）。如果愿意可指定列OrderID的序数值（此时为0），代码如下：

```
object  id=reader[0];
```

这与指定列名的效果是一样的(虽然它提高了性能)，列的序数值一般都是从零开始的，因此表中第一列的序数值为0，第二个列值为1。

注意，索引返回的是object类型的值，使用Datareader 索引通常返回object类型，这是因为索引必须能够返回可以存于数据库的任何数据类型，而且在.net中object类型是所有对象的基类。通过返回一个对象，索引有能力返回任何值。本例中，循环的最后一行将值输出到控制台。在大多数情况下，可用类型安全的方法访问数据，更多相关内容将在后面章节介绍。

最后两行代码关闭 Datareader 和连接。 如果不关闭 Datareader, 其他对象将不能使用该连接。连接是罕有资源，在不再需要时就不该继续占用。

当DataReader完成任务时，应该尽早关闭它。

使用 DataReaders

如果我们执行程序, 输出结果就像这样:

输出列表有三列，依次为Order表中的OrderID ， OrderData和Freight列。

如上所述，这就是SQL语句的执行过程。下面讨论怎么执行存储过程。

用 DataReader 执行存储过程

我们已经知道该如何对数据库执行简单 SQL 语句，所以现在将介绍怎么用命令对象和datareader来执行存储过程。在本例中，我们要使用来自Northwind数据库的CustOrdersDetail 储存过程。该存储过程SQL如下所示，大家无需关心它的内容，需要了解的是它接受一个订单ID作为输入参数，并返回该订单的相关细节：

```
CREATE PROCEDURE CustOrdersDetail @OrderID int
AS
SELECT ProductName,
UnitPrice=ROUND(Od.UnitPrice, 2),
Quantity,
Discount=CONVERT(int, Discount * 100),
ExtendedPrice=ROUND(CONVERT(money, Quantity * (1 - Discount) *
Od.UnitPrice),
2)
FROM Products P, [Order Details] Od
WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
```

Command对象在执行存储过程时做了大量工作。本例中我们仍然以通常方式使用Datareader。本例的所有代码都可以在SqlStoredProcedureExample工程中找到。通常，对于OleDb 供给者的代码几乎与此相同，可以在 OleDbStoredProcedureExample 中发现。下面给出Sqlclient代码的main方法：

```
SqlConnection conn = new SqlConnection ( "Initial Catalog=Northwind;Data
Source=localhost;Integrated Security=SSPI" );
SqlCommand storedProcCommand = new SqlCommand ( "CustOrdersDetail",
conn );
storedProcCommand.CommandType = CommandType.StoredProcedure;
storedProcCommand.Parameters.Add ( "@OrderID", 10248 );
Chapter 4
conn.Open ();
ArrayList rowList = new ArrayList ();
SqlDataReader reader = storedProcCommand.ExecuteReader ( );
while ( reader.Read () )
{
object []values = new object[ reader.FieldCount ];
// retrieves all the fields values into an array of objects
reader.GetValues ( values );
rowList.Add ( values );
}
reader.Close ();
conn.Close ();
foreach ( object []row in rowList )
{
foreach ( object column in row )
Console.WriteLine ( column );
Console.WriteLine ( "\n" );
}
```

现在，让我们更详细地看看关于该代码的更多细节。首先看看怎么创建Connection对象。本例使用 SqlCient 受管供给者，因此，我们需要创建一个 SqlConnection 对象：

```
SqlCommand storedProcCommand = new SqlCommand ( "CustOrdersDetail",
conn );
storedProcCommand.CommandType = CommandType.StoredProcedure;
storedProcCommand.Parameters.Add ( "@OrderID", 10248 );
```

第一行代码创建了一个 SqlCommand 对象，而且传递了两个叁数。第二个叁数是通用连接对象；这里第一个叁数比较重要。我们传递的第一个参数不是SQL字符串，而是存储过程在Northwind 数据库中的名称，本例中是CustOrdersDetail。 这将告诉SqlCommand对象所要执行的存储过程。

第二行代码告知Command对象有一个存储过程将被执行。通过将CommandType属性值置为StoredProcedure来完成。如果我们没有这样做，Command会试着像执行一个标准的SQL语句那样执行，这会导致一个语法异常。

最后，我们需要把一些叁数加入命令。存储过程CustOrdersDetail将对消费者

订单的ID进行检索。因此要把一个新的叁数加入 SqlCommand的叁数集合。叁数的名字是 @ OrderID ，我们需要的序号是 10248 。 基本上，Command对象的所有信息都可用来构造SQL语句。@标志是在SQL语句中用来标志一个叁数。
现在我们开始创建一个Datareader:

```
Using DataReaders
ArrayList rowList = new ArrayList ();
SqlDataReader reader = storedProcCommand.ExecuteReader ( );
while ( reader.Read () )
{
object []values = new object[ reader.FieldCount ];
reader.GetValues ( values );
rowList.Add ( values );
}
```

上面代码引入一对新的参数和方法，所以首先分析一下它们。While循环的第一行创建一个对象数组。数组中每个对象都表示Datareader 的一列。藉由查看FieldCount属性，我们可以知道Datareader 中有多少列。
现在我们有一个需要数值的对象数组。如果我们需要, 可以为每个对象赋值并把它们加入到数组中，另外的一个方法是使用 GetValues 方法。这一个方法将会把当前Datareader 中的列值填充到对象数组中。这种简单方法的代价实际上是放弃了类型安全性，但是，这通常是在开发环境中不得不作出的选择。
如果你正觉得奇怪为什么我们在示例程序顶端创建了一个 ArrayList, 那么来研究一下它。我们在最后一行将对象数组加入到名为 rowList 的 ArrayList的对象中。我们为什么这么做？好吧，请记住， Datareader 打开的时候，它会有效地阻断连接以便使其他对象无法访问连接。将从Datareader检索到的数据存储到高速缓存中，并在Datareader关闭之后处理它往往是一个很好的解决办法。阻塞连接时间越短，数据连接越完善。
在Datareader关闭后，才对它进行处理。
最后，在我们关闭Datareader 和连接后，可以将取得的值输出到荧幕上，使用如下代码：

```
foreach ( object []row in rowList )
{
foreach ( object column in row )
Console.WriteLine ( column );
Console.WriteLine ( "\n" );
}
```

这些代码非常易于理解。到目前为止，我们已经有了一个 ArrayList 用于对象数组，而数组里的每个对象都代表 Datareader 中的一个列值。因此， 我们在外循环中获得列值数组，在内层循环反复读取值并将它们输出到控制台。最后，在屏幕上打印一个空行，这可使输出结果更加清晰可见。

源代码

主页的一些主要功能代码：

```
private void Page_Load(object sender, System.EventArgs e)
{                this.Label8.Visible=false;// 在此处放置用户代码以初始化页面
                 this.Label5.Visible=false;                              }
private void Button1_Click(object sender, System.EventArgs e)
{                this.Label4.Visible=false;
                 this.Label5.Visible=false;
                 this.Label7.Visible=false;
                 this.Label8.Visible=false;
                 this.Label12.Visible=false;
                this.Label13.Visible=false;
                this.Label14.Visible=false;
                 SqlDataAdapter dsCommand = new SqlDataAdapter();
                 dsCommand.SelectCommand = new SqlCommand();
                 dsCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                 dsCommand.SelectCommand.Connection.Open();
                 SqlCommand command = dsCommand.SelectCommand;
                 SqlParameter param = new SqlParameter();
               if (this.RadioButtonList1.SelectedValue.Equals ("客户"))
            {   command.CommandText = "CheckCustomerID";
                 command.CommandType = CommandType.StoredProcedure;
                 param = new SqlParameter("@CustomerID", SqlDbType.Int,4);
                 param.Value = this.TextBox1.Text.ToString();
                 command.Parameters.Clear();
                 command.Parameters.Add(param);
                SqlDataReader reader1 =
command.ExecuteReader(CommandBehavior.Default);
                 bool b = reader1.Read();
                 reader1.Close();
                  if(b==false)
                       this.Label12.Visible=true;
                  else
                  {    command.CommandText = "CheckPassword";
                       command.CommandType = CommandType.StoredProcedure;
                       param =    new SqlParameter("@Password", SqlDbType.Float,8);
                       param.Value = this.TextBox2.Text.ToString();
                       command.Parameters.Clear();
                       command.Parameters.Add(param);
                       SqlDataReader reader2 =
```

```
command.ExecuteReader(CommandBehavior.Default);
                    bool c = reader2.Read();
                    reader2.Close();
                    if (c==false)
                        this.Label13.Visible=true;
                    else
                    {    command.CommandText = "GetNameAndSex";
                        command.CommandType = CommandType.StoredProcedure;
                        param =    new SqlParameter("@CustomerID",
SqlDbType.Int,4);
                        param.Value = this.TextBox1.Text.ToString();
                        command.Parameters.Clear();
                        command.Parameters.Add(param);
                        SqlDataReader reader3 =
command.ExecuteReader(CommandBehavior.Default);
                        reader3.Read();
                        String s2 = reader3.GetString(0);
                        String s3 = reader3.GetString(1);
                        this.Label4.Text=s2;
                        this.Label4.Visible=true;
                        reader3.Close();
                        s=Int32.Parse(this.TextBox1.Text);
                        Cache["Text"]=this.TextBox1.Text;
                        i=1;
                        Cache["CheckIn"]=i.ToString();
                        if(s3=="男")
                        {    this.Label7.Text="先生";
                            this.Label7.Visible=true;
                            this.Label5.Visible=true;
                        }
                        else if(s3=="女")
                        {    this.Label7.Text="女士";
                            this.Label7.Visible=true;
                            this.Label5.Visible=true;
                        }
                        this.Button3.Enabled=true;
                        this.Button4.Enabled=true;
                        this.Button5.Enabled=true;
                         this.HyperLink2.Enabled=true;}
                }
                }
             else if(this.RadioButtonList1.SelectedValue.Equals ("管理员"))
            {  command.CommandText = "Checkadmin";
               command.CommandType = CommandType.StoredProcedure;
```

```
            command.Parameters.Clear();
        command.Parameters.Add("@Administor",this.TextBox1.Text.ToString());
            SqlDataReader reader4 =
command.ExecuteReader(CommandBehavior.Default);
             bool d = reader4.Read();
            reader4.Close();
            if(d==false)
             this.Label12.Visible=true;
              else
             {    command.CommandText = "CheckadminPasswd";
                  command.CommandType = CommandType.StoredProcedure;
                  command.Parameters.Clear();
    command.Parameters.Add("@AdminPassword",this.TextBox2.Text.ToString());
                  SqlDataReader reader5 =    command.ExecuteReader();
                  bool f = reader5.Read();
                  reader5.Close();
                  if (f==false)
                       this.Label13.Visible=true;
                  else
                       Cache["admin"]=this.TextBox1.Text.ToString();
    Response.Redirect("http://localhost/webservice1/AdministorPage.aspx");
                  dsCommand.SelectCommand.Connection.Close();
             }
             }
        }
        private void Button2_Click(object sender, System.EventArgs e)
        {    Response.Redirect("http://localhost/webservice1/RegistRequest.aspx");
        }
        private void Button4_Click(object sender, System.EventArgs e)
        {
            Response.Redirect("http://localhost/webservice1/Authentication.aspx");
        }
        private void Button5_Click(object sender, System.EventArgs e)
        {     this.Label3.Enabled=false;
            this.Label4.Enabled=false;
              this.Label8.Visible=true;
            this.Button3.Enabled=false;
            this.Button4.Enabled=false;
            this.Button5.Enabled=false;
            this.Label14.Visible=false;
            this.Label15.Visible=false;
            i=0;
            Cache["ChekIn"]=i.ToString();
        }
```

```
        private void Button3_Click(object sender, System.EventArgs e)
        {    this.Label11.Visible=true;
             this.Button6.Visible=true;
             this.Button7.Visible=true;
        }
        private void Button6_Click(object sender, System.EventArgs e)
        {SqlDataAdapter sCommand = new SqlDataAdapter();
             sCommand.DeleteCommand = new SqlCommand();
             sCommand.DeleteCommand.Connection = new
SqlConnection(SqlConn.ConStr);
             sCommand.DeleteCommand.Connection.Open();
             SqlCommand command1 = sCommand.DeleteCommand;
             command1.CommandText = "DeleteCustomerInfor";
             command1.CommandType = CommandType.StoredProcedure;
             command1.Parameters.Clear();
             command1.Parameters.Add("@CustomerID",s);
             if (command1.ExecuteNonQuery() != 0)
             {this.Label14.Visible=true;
                  this.Label11.Visible=false;
                  this.Button6.Visible=false;
                  this.Button7.Visible=false;
                  this.Label4.Visible=false;
                  this.Label5.Visible=false;
                  this.Label7.Visible=false;
                  this.Button3.Enabled=false;
                  this.Button4.Enabled=false;
                  this.Button5.Enabled=false;
                    }
             else
                     this.Label15.Visible=true;
        }
        private void Button7_Click(object sender, System.EventArgs e)
        {    this.Label11.Visible=false;
             this.Button6.Visible=false;
             this.Button7.Visible=false;
        }
        private void ImageButton2_Click(object sender,
System.Web.UI.ImageClickEventArgs e)
        {    Response.Redirect("shelf.aspx");
        }
    }
}
```

商城页面代码，主要实现商品的浏览和购买：

```
private void Page_Load(object sender, System.EventArgs e)
        {this.Image13.Visible=false;
            this.Button1.Visible=false;
            this.TextBox1.Visible=false;
            i=Convert.ToInt32(Cache["CheckIn"]);
            a=Convert.ToInt32(Cache["Text"]);// 在此处放置用户代码以初始化
页面}
        private void ImageButton7_Click(object sender,
System.Web.UI.ImageClickEventArgs e)
        {
            Response.Redirect("http://localhost/webservice1/OrderConform.aspx");
        }
        private void ImageButton2_Click(object sender,
System.Web.UI.ImageClickEventArgs e)
        {   DataTable table    = new DataTable();
            SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "GetItem2";
            command.CommandType = CommandType.StoredProcedure;
            aCommand.Fill(table);
            this.DataGrid1.DataSource = table;
            DataGrid1.DataBind();
            aCommand.SelectCommand.Connection.Close();
                }
        private void ImageButton3_Click(object sender,
System.Web.UI.ImageClickEventArgs e)
        {   DataTable table1    = new DataTable();
            SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "GetItem3";
            command.CommandType = CommandType.StoredProcedure;
            aCommand.Fill(table1);
            this.DataGrid1.DataSource = table1;
            DataGrid1.DataBind();
            aCommand.SelectCommand.Connection.Close();
```

```
                }
        private void ImageButton4_Click(object sender,
System.Web.UI.ImageClickEventArgs e)
        {   DataTable table2   = new DataTable();
            SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "GetItem1";
            command.CommandType = CommandType.StoredProcedure;
            aCommand.Fill(table2);
            this.DataGrid1.DataSource = table2;
            DataGrid1.DataBind();
            aCommand.SelectCommand.Connection.Close();                    }
        private void ImageButton5_Click(object sender,
System.Web.UI.ImageClickEventArgs e)
        {   DataTable table3   = new DataTable();
            SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "GetItem4";
            command.CommandType = CommandType.StoredProcedure;
            aCommand.Fill(table3);
            this.DataGrid1.DataSource = table3;
            DataGrid1.DataBind();
            aCommand.SelectCommand.Connection.Close();   }
        private void Button1_Click(object sender, System.EventArgs e)
        {   SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.UpdateCommand = new SqlCommand();
            aCommand.UpdateCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.UpdateCommand.Connection.Open();
            SqlCommand command = aCommand.UpdateCommand;
            command.CommandText = "UpdateTempOrder";
            command.CommandType = CommandType.StoredProcedure;
            command.Parameters.Clear();
    command.Parameters.Add("@OrderNumber",Convert.ToInt32(this.TextBox1.Te
xt.ToString()));
            command.Parameters.Add("@ItemName",s);
```

```
                command.Parameters.Add("@CustomerID",a);
                command.ExecuteNonQuery();
                aCommand.UpdateCommand.Connection.Close();
                Response.Redirect("OrderConform.aspx");            }
            private void DataGrid1_ItemCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
            {    if (i==1)
                {    switch(e.CommandName)
                    {          case "Buy":
                               this.TextBox1.Visible=true;
                               this.Button1.Visible=true;
                               this.Image13.Visible=true;
                               SqlDataAdapter aCommand = new SqlDataAdapter();
                               aCommand.InsertCommand = new SqlCommand();
                               aCommand.InsertCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                               aCommand.InsertCommand.Connection.Open();
                               SqlCommand command = aCommand.InsertCommand;
                               command.CommandText = "InsertTempOrder1";
                               command.CommandType =
CommandType.StoredProcedure;
                               command.Parameters.Clear();
                               command.Parameters.Add("@CustomerID",a);

     command.Parameters.Add("@ItemName",e.Item.Cells[0].Text.ToString());

     command.Parameters.Add("@ItemCost",Convert.ToDouble(e.Item.Cells[1].Text
));
     command.Parameters.Add("@ItemMemberCost",Convert.ToDouble(e.Item.Cells
[2].Text));
                               command.ExecuteNonQuery();
                               s=e.Item.Cells[0].Text.ToString();
                               aCommand.InsertCommand.Connection.Close();
                               break;
                          case "Comment":
                               Cache["Comment"]=e.Item.Cells[0].Text.ToString();
                               Response.Redirect("ItemComment.aspx");
                               break;
                    }
                }
                else
                {this.Label5.Visible=true;
                 this.HyperLink1.Visible=true;
                 this.Label6.Visible=true;}
```

```
        }
            private void Button2_Click(object sender, System.EventArgs e)
        {    DataTable table4    = new DataTable();
            SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "SearchItem";
            command.CommandType = CommandType.StoredProcedure;
            command.Parameters.Clear();
        command.Parameters.Add("@ItemName",this.TextBox2.Text.ToString());
            SqlDataReader reader =    command.ExecuteReader();
             bool f = reader.Read();
            reader.Close();
            if (f==false)
                this.Label8.Visible=true;
            else
            {aCommand.Fill(table4);
            this.DataGrid1.DataSource = table4;
            DataGrid1.DataBind();
                aCommand.SelectCommand.Connection.Close();}
        }
    }
}
```

仓库页面代码，主要实现货物的盘点以及新商品的引进：

```
private void Page_Load(object sender, System.EventArgs e)
        {    this.Label4.Visible=false;
            this.Label5.Visible=false;
            this.Label6.Visible=false;// 在此处放置用户代码以初始化页面
        }
        private void Button7_Click(object sender, System.EventArgs e)
        {     this.Label6.Visible=true;
            DataTable table2    = new DataTable();
            SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "GetNewItemInfor";
            command.CommandType = CommandType.StoredProcedure;
```

```
            aCommand.Fill(table2);
            this.DataGrid3.DataSource = table2;
            DataGrid3.DataBind();
            aCommand.SelectCommand.Connection.Close();
        }
        private void Button3_Click(object sender, System.EventArgs e)
        {   SqlDataAdapter dCommand = new SqlDataAdapter();
            dCommand.SelectCommand = new SqlCommand();
            dCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            dCommand.SelectCommand.Connection.Open();
            SqlCommand command = dCommand.SelectCommand;
            command.CommandText = "ItemInforChangeLoad";
            command.CommandType = CommandType.StoredProcedure;
            command.Parameters.Clear();
    command.Parameters.Add("@ItemType",this.DropDownList1.SelectedValue.To
String());
          command.Parameters.Add("@ItemName",this.TextBox1.Text.ToString());
            SqlDataReader reader =   command.ExecuteReader();
            bool s = reader.Read();
            if(s==false)
            {this.Label3.Visible=true;
            reader.Close();}
              else
            {string s1=reader.GetInt32(0).ToString();
                Cache["ItemID"]=s1;
                reader.Close();
                    dCommand.SelectCommand.Connection.Close();
                Response.Redirect("http://localhost/webservice1/ItemInfor.aspx");}
        }
            private void Button4_Click(object sender, System.EventArgs e)
        {   DataTable table   = new DataTable();
            SqlDataAdapter dCommand = new SqlDataAdapter();
            dCommand.SelectCommand = new SqlCommand();
            dCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            dCommand.SelectCommand.Connection.Open();
            SqlCommand command = dCommand.SelectCommand;
            command.CommandText = "GetRecieveItemRecord";
            command.CommandType = CommandType.StoredProcedure;
            dCommand.Fill(table);
            this.DataGrid1.DataSource = table;
            DataGrid1.DataBind();
            dCommand.SelectCommand.Connection.Close();
```

```
            this.Label4.Visible=true;
        }
        private void Button5_Click(object sender, System.EventArgs e)
        {
            DataTable table1    = new DataTable();
              SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand();
            aCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
            aCommand.SelectCommand.Connection.Open();
            SqlCommand command = aCommand.SelectCommand;
            command.CommandText = "GetSendItemRecord";
            command.CommandType = CommandType.StoredProcedure;
            aCommand.Fill(table1);
            this.DataGrid2.DataSource = table1;
            DataGrid2.DataBind();
            aCommand.SelectCommand.Connection.Close();
            this.Label5.Visible=true;
        }
        private void Button6_Click(object sender, System.EventArgs e)
        {    Response.Redirect("Storage.aspx");
        }
        private void DataGrid3_ItemCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {    if(e.CommandName=="Get")
        {           SqlDataAdapter dCommand = new SqlDataAdapter();
                    dCommand.InsertCommand = new SqlCommand();
                    dCommand.InsertCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                    dCommand.InsertCommand.Connection.Open();
                    SqlCommand command1 = dCommand.InsertCommand;
                    command1.CommandText = "InsertNewItem";
                    command1.CommandType = CommandType.StoredProcedure;
                    command1.Parameters.Clear();
    command1.Parameters.Add("@NewItemName",e.Item.Cells[0].Text.ToString());
    command1.Parameters.Add("@NewItemType",e.Item.Cells[1].Text.ToString());
    command1.Parameters.Add("@NewItemCost",Convert.ToDouble(e.Item.Cells[2]
.Text));
    command1.Parameters.Add("@NewItemInfor",e.Item.Cells[3].Text.ToString());
    command1.Parameters.Add("@NewSellDate",e.Item.Cells[4].Text.ToString());
    command1.Parameters.Add("@FactoryName",e.Item.Cells[5].Text.ToString());
                    command1.ExecuteNonQuery();
                    dCommand.InsertCommand.Connection.Close();
                    dCommand.DeleteCommand =new SqlCommand();
```

```
                    dCommand.DeleteCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                    dCommand.DeleteCommand.Connection.Open();
                     SqlCommand command2 = dCommand.DeleteCommand;
                     command2.CommandText = "DeleteNewItem";
                     command2.CommandType =
CommandType.StoredProcedure;
                     command2.Parameters.Clear();
command2.Parameters.Add("@NewItemName",e.Item.Cells[0].ToString());
                    command2.ExecuteNonQuery();
                    dCommand.DeleteCommand.Connection.Close();
                     Response.Redirect("Storehouse.aspx");}
        }
```

网上付款页面的代码，只要实现客户支付的验证：

```
private void Page_Load(object sender, System.EventArgs e)
        {     i=Convert.ToInt32(Cache["Text"]);
            if(!this.IsPostBack)
            {SqlDataAdapter aCommand = new SqlDataAdapter();
            aCommand.SelectCommand = new SqlCommand("ListView", new
SqlConnection(SqlConn.ConStr));
            aCommand.SelectCommand.CommandType =
CommandType.StoredProcedure;
            aCommand.SelectCommand.Parameters.Add("@CustomerID", i);
            aCommand.SelectCommand.Connection.Open();
            aCommand.Fill(table);
            this.DataGrid1.DataSource = table;
            DataGrid1.DataBind();
            aCommand.SelectCommand.Connection.Close();}// 在此处放置用户代
码以初始化页面}
        private void Button1_Click(object sender, System.EventArgs e)
        {       DateTime time = DateTime.Now;
                 string t=time.ToLongDateString();
                if(DropDownList1.SelectedValue.Equals("普通客户"))
            {   SqlDataAdapter dsCommand = new SqlDataAdapter();
                dsCommand.SelectCommand = new SqlCommand();
                dsCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                dsCommand.SelectCommand.Connection.Open();
                SqlCommand command = dsCommand.SelectCommand;
                command.CommandText = "CredictCardCheck";
                command.CommandType = CommandType.StoredProcedure;
                command.Parameters.Clear();
                command.Parameters.Add("@CustomerID",i);
```

```
    command.Parameters.Add("@CredictCard",Convert.ToDouble(this.TextBox1.Te
xt));
                    SqlDataReader reader =    command.ExecuteReader();
                    bool f = reader.Read();
                    dsCommand.SelectCommand.Connection.Close();
                    reader.Close();
                    if (f==false)
                        this.Label4.Visible=true;
                    else
                    {       for(int a=0;a<table.Rows.Count;a++)
                       {    dsCommand.InsertCommand = new SqlCommand();
                            dsCommand.InsertCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                            dsCommand.InsertCommand.Connection.Open();
                            SqlCommand command1 = dsCommand.InsertCommand;
                            command1.CommandText = "SaveList";
                            command1.CommandType =
CommandType.StoredProcedure;
                            command1.Parameters.Clear();
                            command1.Parameters.Add("@CustomerID",i);
command1.Parameters.Add("@ItemName",table.Rows[a][0].ToString());
command1.Parameters.Add("@ItemNumber",Convert.ToDouble(table.Rows[a][3]));
                            command1.Parameters.Add("@OrderDate",t);
                            command1.ExecuteNonQuery();
                            dsCommand.InsertCommand.Connection.Close();
                            dsCommand.DeleteCommand= new SqlCommand();
                            dsCommand.DeleteCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                            dsCommand.DeleteCommand.Connection.Open();
                            SqlCommand command2 = dsCommand.DeleteCommand;
                            command2.CommandText = "DeleteTempList";
                            command2.CommandType =
CommandType.StoredProcedure;
                            command2.Parameters.Clear();
                            command2.Parameters.Add("@CustomerID",i);
                            command2.ExecuteNonQuery();
                            dsCommand.DeleteCommand.Connection.Close();
                                this.Label3.Visible=true;
                                this.HyperLink1.Visible=true;
                                  }
                    }
                }
                else
```

```
{   SqlDataAdapter dCommand = new SqlDataAdapter();
    dCommand.SelectCommand = new SqlCommand();
    dCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
    dCommand.SelectCommand.Connection.Open();
    SqlCommand command = dCommand.SelectCommand;
    command.CommandText = "MemberCheck";
    command.CommandType = CommandType.StoredProcedure;
    command.Parameters.Clear();
    command.Parameters.Add("@CustomerID",i);
command.Parameters.Add("@CredictCard",Convert.ToDouble(this.TextBox1.Text));
    SqlDataReader reader1 =   command.ExecuteReader();
    bool g = reader1.Read();
    dCommand.SelectCommand.Connection.Close();
    reader1.Close();
    if (g==false)
        this.Label6.Visible=true;
    else
    {   SqlDataAdapter gCommand = new SqlDataAdapter();
        gCommand.SelectCommand = new SqlCommand();
        gCommand.SelectCommand.Connection = new
SqlConnection(SqlConn.ConStr);
        gCommand.SelectCommand.Connection.Open();
        SqlCommand command1 = gCommand.SelectCommand;
        command1.CommandText = "CredictCardCheck";
        command1.CommandType = CommandType.StoredProcedure;
        command1.Parameters.Clear();
        command1.Parameters.Add("@CustomerID",i);
command1.Parameters.Add("@CredictCard",Convert.ToInt32(this.TextBox1.Text));
        SqlDataReader reader2 =   command.ExecuteReader();
        bool h = reader2.Read();
        gCommand.SelectCommand.Connection.Close();
        reader2.Close();
        if (h==false)
            this.Label4.Visible=true;
        else
        {   for(int b=0;b<table.Rows.Count;b++)
            {   gCommand.InsertCommand = new SqlCommand();
                gCommand.InsertCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                gCommand.InsertCommand.Connection.Open();
                SqlCommand command3 =
gCommand.InsertCommand;
                command3.CommandText = "SaveList";
```

```
                                command3.CommandType =
CommandType.StoredProcedure;
                                command3.Parameters.Clear();
                                command3.Parameters.Add("@CustomerID",i);
command3.Parameters.Add("@ItemName",table.Rows[b][0].ToString());
command3.Parameters.Add("@ItemNumber",Convert.ToDouble(table.Rows[b][3]));
                                command3.Parameters.Add("@OrderDate",t);
                                command3.ExecuteNonQuery();
                                gCommand.InsertCommand.Connection.Close();
                                gCommand.DeleteCommand= new SqlCommand();
                                gCommand.DeleteCommand.Connection = new
SqlConnection(SqlConn.ConStr);
                                gCommand.DeleteCommand.Connection.Open();
                                SqlCommand command4 =
gCommand.DeleteCommand;
                                command4.CommandText = "DeleteTempList";
                                command4.CommandType =
CommandType.StoredProcedure;
                                command4.Parameters.Clear();
                                command4.Parameters.Add("@CustomerID",i);
                                command4.ExecuteNonQuery();
                                gCommand.DeleteCommand.Connection.Close();
                                    this.Label3.Visible=true;
                                    this.Label4.Visible=false;
                                    }
                            }
                        }
                    }
                }
            }
        }
```