摘　要

随着我国文艺演出和体育比赛等演出项目越来越多，出售传统票品的票务系统已经不能再满足市场需求，而当前国内尚没有一个功能完善的在线电子票务管理系统，因此开发一个电子票务管理系统是非常有意义和可行的。

电子票务管理系统是一个基于网络环境的综合性服务平台，专门为文艺演出、体育比赛等票务项目的管理而设计。电子票务管理系统从客户的角度出发，提供了多种购票方式，其中有代理商客户端售票、代理商网上售票和个人网上售票（包括 Web 售票和手机电子票售票）。电子票务管理系统采用先进的二维条码制作识别技术和票务信息管理相结合，支持可视化选择座位，具有国际化支持的功能。电子票是一个以二维条码作为验证形式的票品，电子票的优点有安全可靠和携带方便。本论文以电子票务管理系统 C/S 部分的设计和实现展开研究和分析，详细介绍了系统的开发过程。电子票务管理系统 C/S 部分由两部分组成，即电子票务后台管理子系统和电子票务代理商售票子系统，前者是用来实现后台管理功能的系统，后者是用来实现代理商售票功能的系统。这两个系统都采用了分层的体系结构设计方法。后台管理则是一个典型的三层应用程序，而代理商售票系统是一个基于 N 层的分布式应用程序。同时在系统的总体设计过程中，采用.Net Remoting 框架来构建分布式应用服务器，运用一个数据访问包装类（类名为：DAO，DAO 封装了连接数据库、执行 SQL 命令和启动事务等操作）来设计数据访问层，并采用了实现传递数据的机制(DataSet，强类型 DataSet，自定义实体类)来实现层间传递数据的问题。

从 2006 年 1 月份本系统投入使用至今，后台管理系统和代理商售票系统基本处于正常运行状态。其间，本系统参与运营了多个演出和娱乐项目，得到了演出主办方和多个代理商的肯定，基本上能够满足演出主办方和代理商的功能需求。

关键词：电子票务管理系统；电子票；分布式应用服务器；国际化

Designation and Implementation of Large Culture and Sports Performance electronic ticketing system

Abstract

Along with theatrical performances and sports competitions increasing, the traditional ticketing system has been unable to meet market demand, and there is not a perfect on-line electronic ticketing management system in domestic market, so the development of an electronic ticketing management system is very meaningful and feasible.

Electronic ticketing management system is a Web-based software platform; it is designed for theatrical performances and sports competitions. Electronic ticketing management system provides a wide variety of ticketing methods, including agent ticketing, agent Web ticketing and personal Web ticketing (including Web ticketing and mobile phone ticketing). Electronic ticketing management system use advanced a two-dimensional bar code identify technology combining with ticketing information management, sustaining visual-seating and international supporting function. Electronic ticket is a two-dimensional bar code for the certification ticket and the advantages of electronic ticket are safe, reliable and convenient carrying. This paper introduces the design and realization of electronic ticketing management system's C/S part. Electronic ticketing management system's C/S part includes electronic ticketing background management system and electronic ticketing agent ticketing system, the former is used to realize background management functions, the latter is used to realize functions of agent ticketing. The two system all use the layered system structure method. Background management is a typical three-layered application program, the agent ticketing system is a based-N layered distributing application program. In the collectivity design, using .Net Remoting Frame conceive distributing application server,and using a data access packing class(class name: DAO,DAO packing the link database、executing SQL command and handling affair) design data access layer, and realize data transfer mechanism(dataset, strong class dataset, user-defined entity-class) realizing the data transfer problem in the layered.

From January 2006 to now ,the background management and agent ticketing system is running stabilization basically. In the period, the system participate in many performance items, achieving affirmation and contenting the requirment of the front side and agent side.

Key words: Electronic ticketing management system; Electronic ticket; Distributed application server; Internationalization

独创性说明

作者郑重声明：本硕士学位论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得大连理工大学或者其他单位的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

作者签名：周建宇 日期：2006.6

大连理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“大连理工大学硕士、博士学位论文版权使用规定”，同意大连理工大学保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权大连理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：

导师签名：

2006年 6月 10日

1 绪论

1.1 课题的背景和意义

当前，随着我国票务市场的不断发展和成熟，传统的印刷票和使用物理防伪技术的票品已经不能再满足需求，电子票这一新生事物开始走向市场，走入寻常百姓的生活。电子票与传统的票品不同，它采用先进的经过数字加密的二维条码技术，不但可以有效降低票面印制成本，而且完全可以使时下票务市场十分头疼的倒票、假票得到彻底地杜绝；同时电子票的保存和携带非常方便，它可以被打印在纸张上，也可以存放在手机和计算机上。

1998 年前后，伴随着商业娱乐演出的兴起，全国的票务业务开始市场化和正规化，并呈快速发展的趋势，专业的票务公司开始出现。当前，市场上主流的票务系统的功能比较单一，对演出项目的整体流程定义不好，采用的是传统的印刷票和使用物理防伪技术的票品。而随着计算机网络、二维条码等为代表的现代技术的成熟，同时识别技术和机器传动控制方面的芯片开发和发展也达到一个新的水准，这些技术为电子票务管理系统的发展提供了坚实的基础。

生活中的许多场合都需要使用各种票证（如文艺娱乐演出、体育比赛、各类旅游风景区、铁路、公路、航运等行业的收费票证），传统的售票及票务管理都是以手工作业为主，导致在某些特定的高峰时段，售票人员的劳动强度剧增，工作效率下降，而管理人员也无法由准确的统计数据来科学合理地安排售票人员的工作，最终的结果就是导致客户抱怨和不满。

电子票务管理系统就是为了解决这个问题而出现的。高效率、低成本是电子票在电子票务管理系统应用中的显著特点，用二维条码技术实现电影票、门票、车票等票据的制作及自动化管理，结束了传统手工售票及统计的历史，它的意义不仅是用票据打印机代替了手工售票，而是使得票务管理工作走向全面自动化、规范化，从根本上解决了票据查询难、售票劳动强度大的现状，提高了票据管理效率和对客户的服务质量。

电子票务管理系统采用先进的二维条码制作识别技术和票务信息管理相结合，具有形象现代化，管理一体化，信息实时性，防伪可靠性，核算严密性的特点。电子票务系统集网上发布售票信息、网上浏览、网上预订、网上支付、实时出票、本地打印、网下验票等功能为一体，在票务营销和管理、降低票务经营成本的同时为广大顾客提供了方便。

1.2 国内、外电子票务的发展和现状

在我国，电子票务的发展也非常快，并没有太落后于英、美等国家，许多行业开始应用电子票和电子票务管理系统，尤其是航空业对电子票务表现出了极大兴趣。

近几年来，国内的航空公司开始大力推行电子票，因为电子票方便携带和安全保密，受到了广大客户的热烈欢迎。2000 年 3 月 28 日，中国南方航空公司正式推出我国第一张电子客票服务。2001 年 11 月初，深圳航空公司在深圳至北京、海口和桂林的航线上，开始试行电子客票。而在 2003 年的 8、9 月间，中国国际航空公司、中国东方航空公司也陆续推出电子客票服务。截至目前，南方航空公司的电子客票销售网络遍布北京、上海、天津、广州、重庆等 48 个城市，覆盖航线超过上百条。据不完全调查，目前广州旅行社中至少有 2 / 3 在使用或正准备使用电子票务。另据南方航空公司电子商务部预测，在将来，电子票务的比例会大幅度提高，至团体出行高峰时，电子客票的比例可占到 40%。预计 2005 年，南航电子客票销售额将达到 40 亿元到 50 亿元，占到客运收入的 25%左右。中国南方航空公司副总经理甚至表示，到 2007 年，南航国内电子客票销售额将达到国内客票销售总额的 40%。

但是，电子票在文艺演出和体育比赛等票务中的运用仍处于萌芽阶段。中演公司和美国达特康公司联手于 2002 年 4 月正式推出中演票务通售票平台，中演票务通在北京的娱乐类票务市场上占有重要地位，不过中演票务通不支持电子票和不支持多种渠道方式售票是它的最大缺点。随着二维条码技术的广泛运用和普通客户对电子票的认同，开发一个基于电子票的娱乐类票务管理系统是非常有意义和发展前景的。

在国外，电子票务在近几年发展非常迅速，欧美国家的很多票务公司纷纷推出电子票务管理系统。据统计在英国与美国的各类娱乐演出和体育比赛中，电子票务约占所有票务市场的 30%，其中体育比赛也占了 15%的份额；在美国，各个航空公司的飞机票大部分采用的都是电子票，如美国联合航空公司、美国西北航空公司的电子票销售已占公司全部票务收入的 70%左右。亚洲的日本和新加坡等国家也大力发展电子票务，在多个行业中电子票务得到了广泛应用。

根据最新消息报道，国际航空运输协会总干事皮西亚尼先生在新加坡发放一份官方文件中透露，国际航空运输协会正计划以 2007 年为目标，实施全球航空业无纸化机票系统，以高科技和设定配套措施规定，采用电子票务来进一步降低航空企业的经营成本。

1.3 课题的主要工作

电子票务管理系统是一个基于网络环境的综合性服务平台，专门为文艺演出、体育比赛等票务项目管理而设计；它是一个集演出项目管理，票品管理，数据统计，订单查

询为一体的综合票务管理系统。实现了全部日常票务经营运作的智能化管理，具有数据图形化，界面友好，操作便捷等特点，使用户简单上手，票务状况一目了然。

电子票务管理系统支持代理商客户端售票、代理商网上售票和个人网上售票（包括Web售票和手机电子票售票）等多种售票功能。本系统具备从多种不同的终端设备售票的能力，并结合成熟的电子票技术，使用户可以同时从浏览器、手机、基于C/S的销售终端实时选择座位和完成购票。本系统打破现有传统售票方式的地区局限性，保障用户随时随地都可以轻松完成购票。

图1.1是电子票务系统的物理结构图。

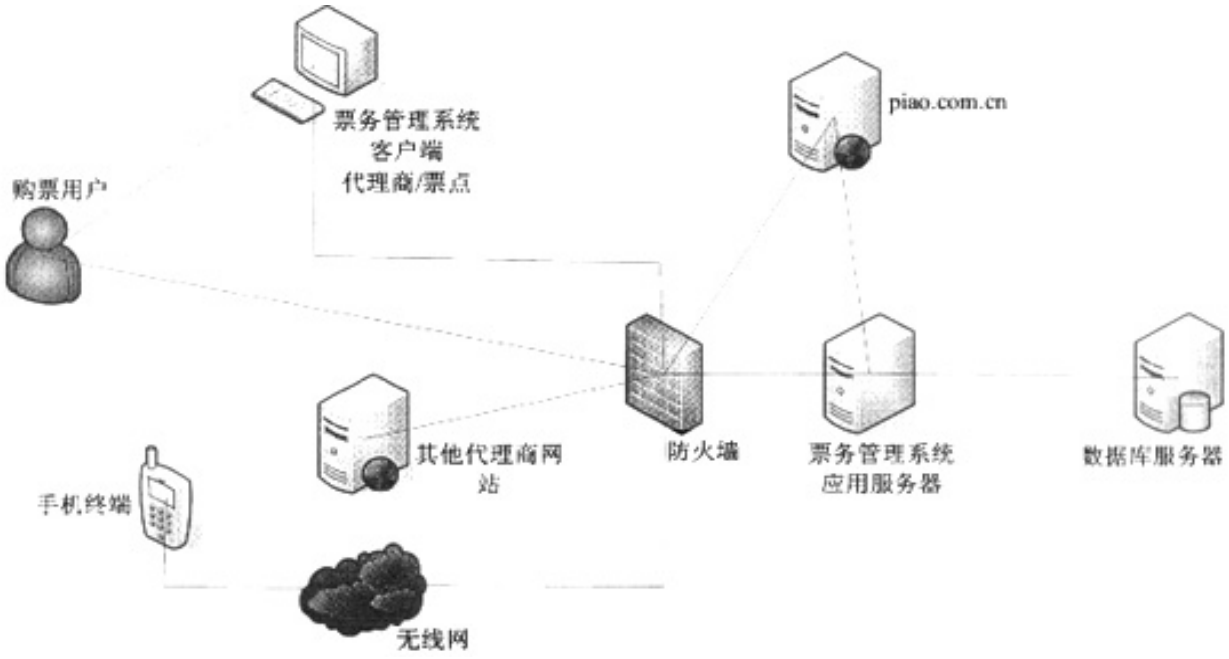

图1.1 系统的物理结构图

Fig. 1.1 System physics structure chart

由图可见，电子票务管理系统可为客户提供四种购票方式：

1. 客户到票务管理系统代理商客户端售票点完成购票，打印的票品为带有二维条码的纸质票，它是本系统提供的最主要的一种购票方式。此代理商客户端与图1.1中的票务管理系统应用服务器一起构成一个基于C/S结构的分布式应用系统；

2. 客户登录www.piao.com.cn网站系统，通过选择票品和进行网络交款直接完成购票，客户得到可打印的带有二维条码的电子票；

3. 客户登录其他代理商网站（被称为合作方），通过选择票品和进行网络交款直接完成购票，客户得到可打印的带有二维条码的电子票；

4. 客户通过手机终端登录无线网络，通过选择票品和进行网络交款直接完成购票，客户得到带有二维条码的电子票，此电子票返回到客户的手机终端。

电子票务管理系统基于微软的.NET Framework 平台，以 SQL SERVER2000 为后台数据库和自行开发的应用服务器（采用.Net Remoting 框架技术实现），以超强数据平台保证系统安全、稳定，具有超高扩展性、安全性、系统整合性，为第三方代理商提供统一、顺畅的售票接口。

在电子票务管理系统的开发中，本人主要从事系统 C/S 部分的设计和实现工作，所以本课题仅仅包含电子票务管理系统的 C/S 部分。注：在下文的所有章节中，“系统”都是指电子票务管理系统的 C/S 部分，而不是整个电子票务管理系统。

1.4 论文的组织结构

本论文由六个部分组成：

第一章：绪论。阐述研究工作的背景和意义，分析了电子票务在国内外的发展现状，并说明课题的主要工作和目标以及本论文的组织结构。

第二章：关键技术及难点研究。介绍了在系统的总体设计过程中，项目小组解决的一些关键技术和难点，主要包括两个部分：分布式应用服务器的构建，设计数据访问层和实现层间传递数据。

第三章：需求分析。介绍了电子票务系统 C/S 部分的功能性需求和非功能性性需求，并且通过用例模型进行了需求分析。

第四章：系统的总体设计。介绍了系统的体系结构风格和体系结构设计；把系统分为后台管理系统和代理商售票系统，并介绍了这两个系统的体系结构设计；之后介绍了基于组件的软件开发方法和系统中其它方面的设计工作。

第五章：系统的详细设计与实现。介绍了后台管理系统和代理商售票系统中的几个功能模块的详细设计与实现；并详细分析了系统国际化版本的实现和水晶报表的实现。

第六章：系统的测试与实施。介绍了系统的测试和部署方面的工作，并分析了系统的运行情况和维护工作。

结论对本论文进行了简单的总结，并对下一步的工作进行了展望。

2 关键技术及难点研究

2.1 分布式应用服务器的构建

在系统的实现过程中，最大的关键技术就是构建一个稳定、健壮的分布式应用服务器（即代理商售票子系统中的服务器端子系统），它是代理商客户端子系统正常运行的保障。

2.1.1 实现分布式应用的几种技术比较

在.Net Framework 中，Web Services 和.Net Remoting 都可以用来实现分布式应用系统，但是它们的应用范围是不同的，下面介绍一下 Web Services 和.Net Remoting 的优缺点。

(1) Web Services 的互操作性好，完全支持 HTTP 上的 WSDL 和 SOAP 协议，可以实现跨平台的访问，但是当传输的数据量很大时，SOAP 协议系统的开销就太大了[1]。

(2) .Net Remoting 可实现公共语言运行时（CLR）的类型系统的高保真，支持其他数据格式和通信通道，更具有通用性和扩展性，并且支持大数据量的传输，唯一的缺点是不能跨平台[2]。.Net Remoting 的目标可以通过应用程序类型和所提供的协议来描述，还可以通过 CLR Object Remoting 来描述。CLR Object Remoting 是.Net Remoting 的一个重要方面，所有的语言结构（例如：构造方法、委托、接口、方法、属性和字段等）都可以与远程对象一起使用。.Net Remoting 通过网络扩展了 CLR 对象的功能，CLR Object Remoting 可以处理激活、分布式验证、生存期和调用环境等方面的工作。

(3) 在 Web Services 中，对象是抽象的，客户端不需要知道服务器端的对象类型；而在.Net Remoting 中，对象是有具体类型的，客户端需要知道服务器端的对象类型。

因为系统必须满足以下一些非功能需求：代理商客户端子系统需要从和代理商服务器端子系统之间相互传输大量数据，并且需要迅速响应各种请求；代理商客户端子系统的操作界面美观，方便使用，用户体验好。所以，本系统将采用.Net Remoting 框架来实现分布式系统。

2.1.2 .Net Remoting 技术介绍

.Net Framework 应用程序是在应用程序域中运行的，应用程序域可以看作进程中的子进程。传统上，进程通常用作隔离的分界线，在一个进程中运行的应用程序不能访问和破坏另一个进程中的资源和内存，对于相互通信的应用程序来说，需要跨进程的通信。在.Net Framework 下，应用程序域就成为进程中新的安全分界线了。在同一个应用程序

域中的对象可以直接进行交互，但是在访问不同应用程序域中的对象时，必须实现跨应用程序域的通信，这就是.Net Remoting 技术所解决的问题[3]。下面列出了.Net Remoting 体系结构的主要元素：

应用程序域：可以把它当作一个逻辑进程，因为单个 Win32 进程可以容纳多个应用程序域。在某一应用程序域中执行的代码和对象不能直接访问另一应用域中的代码和对象。

.Net Remoting 边界：两个.Net Framework 应用程序域之间的分界线就构成了一个.Net Remoting 边界。

穿越边界：.Net Remoting 让在应用程序域和上下文的逻辑子部分中执行的对象可以越过.Net Remoting 边界相互交互。.Net Remoting 基础设施把对象分为两类：不可远程的和可远程的。顾明思义，可远程对象可以穿越.Net Remoting 边界，或者通过.Net Remoting 边界被访问。.Net Remoting 定义了三种可远程化的类型：按值编列、按引用编列和上下文绑定。

通道：通道用于客户端和服务器之间的通信。通道包括客户端的通道部分和服务器端的通道部分。.Net Framework 1.1 提供了两种基本通道类型 HTTP 和 TCP 通道，它们分别通过 HTTP 和 TCP 进行通信。此外，还可以创建自己订制的通道，使用不同的协议进行通信。

消息：消息被发送到通道中。消息是为客户端和服务器端之间的通信而创建的。消息包括远程对象的信息、被调用的方法的名称和返回信息。

代理对象：客户端是通过代理对象来访问远程对象的，它分为透明代理对象和真实代理对象。对客户端来说，透明代理对象看起来与远程对象类似，它作为远程对象在客户端的代理。真实代理对象接收透明代理对象创建的消息并将其发送到.Net Remoting 基础设施用于传递给远程对象，最终远程对象执行操作并返回结果。

远程对象：远程对象是运行在服务器端的对象。客户端不能直接调用远程对象的方法，而要使用代理对象。

图 2.1 是.Net Remoting 的远程调用示意图，它描述了客户端对象是怎样与远程对象进行通信的：

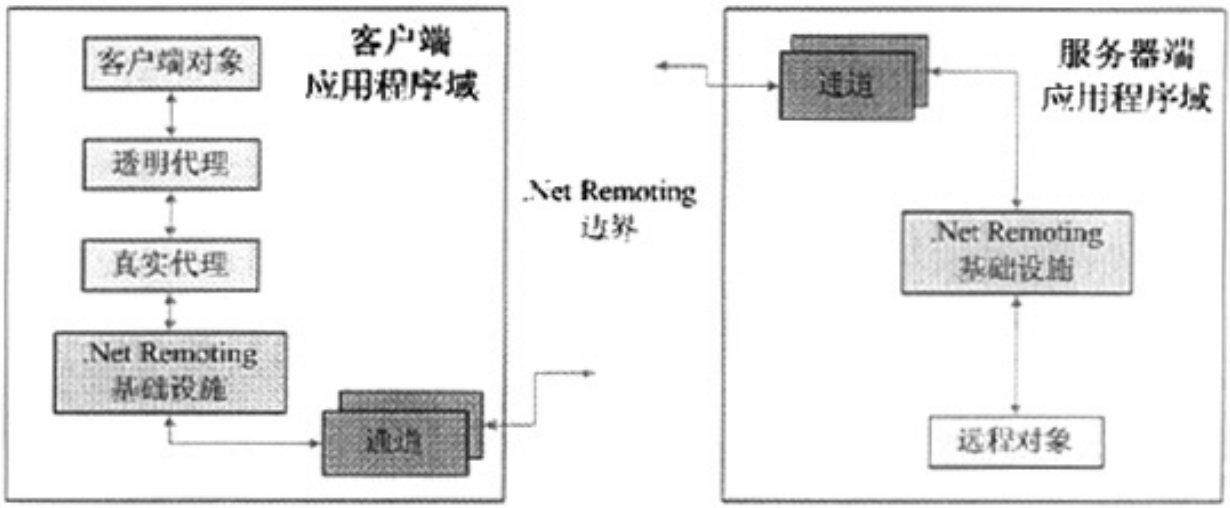

图 2.1 .Net Remoting 的远程调用

Fig. 2.1 .Net Remoting long-distance transfer

由图 2.1 可见，当客户端应用程序域中的客户端对象调用远程对象的方法时，客户端对象实际上调用的是透明代理对象的方法。透明代理对象使用反射机制，从程序集中读取元数据，获得真实代理对象的公共方法的信息。然后，透明代理对象调用真实代理对象，真实代理对象通过.Net Remoting 基础设施把消息发送到通道中。当消息到达后，服务器端应用程序域通过.Net Remoting 基础设施接受消息和发送信息给远程对象，远程对象执行请求方法，再次通过通道传送返回结果给客户端应用程序域。客户端应用程序域接收返回结果，最终返回给客户端对象。至此，一个完整的.Net Remoting 的远程调用就完成了。

2.1.3 用.Net Remoting 构建分布式应用服务器

运用.Net Remoting 技术来设计一个分布式系统需要一些必要的步骤，在本节下面的内容中，作者将结合.Net Remoting 在本系统中的实现去说明这些步骤。

(1) 使类型可远程化

按值编列:即 marshal by value（MBV）。使用这种方式所传递的是某个 type 类型的对象，是将此对象的 value 值拷贝到远程应用程序域。要做到这一点，用户自定义的 types 必须可序列化(其定义必须标以 Serializable)。在一个远程调用中，当上述 types 的对象被作为参数或返回值传递时，这些对象会被自动序列化，并被传递到远程应用程序域；一旦到达远程应用程序域，数据会被自动逆序列化为对象[4]。

按引用编列:即 marshal by reference（MBR）。它指对象跨越应用程序域边界，以 reference 的方式传递一个 reference type 对象。只有当这个 reference type 继承于

MarshalByRefObject 才可以这么做。当一个 MBR 对象跨越应用域边界进行传递时，被传递的只是对象的 reference[4]。本文中，仅仅把按引用编列的可远程化的对象称为远程对象。

代理商客户端子系统需要和代理商服务器端子系统交互以实现代理商售票员登陆客户端子系统、查询演出信息、可视化售票和订单查询等诸多功能。一般而言，一个小功能都需要与之相对应的一个远程对象来实现，但是随着业务逻辑越来越复杂，这些远程对象也会越来越多，管理和维护这些远程对象也越来越困难。为了降低管理和维护这些远程对象的困难度，在本系统中我们只发布了两个远程对象（即：AgentLogin 和 RMIProxy），这两个远程对象都是按引用编列的。AgentLogin 远程对象只用来实现代理商售票员登陆客户端子系统和退出客户端子系统的功能，而 RMIProxy 用来实现其他所有与业务相关的逻辑。

(2) 选择一个宿主应用程序域

控制台应用程序和 Windows 应用程序都可以作为服务器端宿主应用程序域。由于控制台应用程序的简单性，使用它作为宿主环境可以进行快速测试和原形的开发。Windows 应用程序通常都被认为是客户端应用程序，而不是服务器端应用程序，为.Net Remoting 宿主使用 GUI 应用程序强调的是如何模糊客户端和服务器端之间的界限。

在代理商售票系统中， 我们采用了一个 Windows 窗体应用程序作为宿主应用程序域。采用这种方式的好处有：通过鼠标可以可视化的启动、重启和停止应用服务器；可视化的显示应用服务器的状态（如：启动时间、被请求次数等）。

(3) 选择一个远程对象的激活模型

设计远程技术时，最大的挑战之一是决定选用什么方案来支持远程对象的创建（即激活，activation），.Net Remoting 提供了三种解决方案[3]。

① Single-call object

对来自每一个客户端的每一次请求，都创建出远程对象类型的一个崭新对象，并于调用结束时被销毁。Single-call 意味着服务器端不用为远程对象保存状态，所以这种模式在服务器端的运行效果是非常好的。Single-call 是服务器端激活方式的一种实现。

② Singleton object

在给定的远程服务器上只创建远程对象类型的唯一对象，所有客户端对远程服务器上的这个远程对象所做的调用，都由这唯一对象来处理。一般情况下，如果要在所有的客户端之间共享一些资源，就可以使用这种远程对象的激活方式。Singleton 也是服务器端激活方式的一种实现。

③ Client-activated object

为每一个意欲使用这一远程对象的客户端都创建对应的唯一对象，只有当客户使用完毕，对应实体才被销毁。如果远程对象要保存某一特定客户端的状态，可以使用客户端激活的对象。Client-activated 是客户端激活方式的一种实现。

在代理商售票系统中，项目小组采用了 Single-call 的服务器端激活方式。因为这种方式极大的满足了本系统的业务需求，同时采用服务器端激活方式可以使在代理商客户端部署尽可能少的逻辑组件。

(4) 选择一个通道和端口

.Net Remoting 提供了两种主要的通道：HttpChannel 和 TcpChannel。影响通道的选择因素有：

① 通道是否穿越防火墙传输；

② 数据作为纯文本发送是否会引发安全问题；

③ 是否需要 IIS 的.Net Remoting 安全特性。

在 HttpChannel 中，数据是作为纯文本发送的，会引发安全问题；同时，代理商客户端子系统和代理商服务器端子系统之间需要交换大量的数据，所以本系统将采用 TcpChannel。TcpChannel 使用二进制格式化程序，此格式化程序以二进制格式进行数据的序列化并使用原始套接字在网络中传输数据[3]。因为，代理商客户端子系统和代理商服务器端子系统可能不在同一个受防火墙保护的封闭环境中，因此在服务器端所处的网络中需要开放一个公共端口。

(5) 确定客户端如何获得服务器端的元数据

客户端必须能够获取描述远程对象类型的元数据，元数据有两个用途：

① 让引用远程对象类型的客户端代码能够成功编译；

② 让.Net 框架生成客户端用于同服务器端远程对象进行交互的代理对象类型。

实现本步骤的最简单方式是：在客户端部署包含远程对象的完全实现的部件。但是这样做的缺点是违背了面向对象的封装原则，更严重的是若客户端知道了远程对象的业务逻辑实现，会引起安全性问题。

在代理商售票系统中，采用了.Net Framework 提供的一个工具 SoapSuds.exe，它支持生成一个“空”类，此“空”类仅包含远程对象类的定义和所有类的成员的定义，此“空”类中没有具体方法的实现代码，这样就保证了安全性问题。从另一种意义上说，这种“空”类只包括远程对象类的元数据，而不包括远程对象类的逻辑实现[5]。为此，把用 SoapSuds.exe 工具生成的远程对象类的对应“空”类部署在代理商客户端子系统中就可以了。

(6) 为.Net Remoting 配置服务器端和客户端

可以通过编程方式和配置文件方式实现为.Net Remoting 配置服务器和客户端。

因为代理商服务器端子系统只公布了两个远程对象，通过简单的编程方式就可以实现.Net Remoting 的服务器端的配置了。在代理商客户端子系统中，通过简单的编程方式也可以实现对远程对象进行远程请求调用了。

(7) 其他的辅助机制

在.Net Remoting 的实现过程中，上述几个步骤是必须的。除此之外，在代理商售票系统的设计中，项目小组还增加了其他一些很有意义的技术：

① 在代理商售票系统中，远程对象的激活模型采用的是 Single-call 方式。因此，服务器端对客户端发来的每一次远程请求都会生成一个新的远程对象，此远程对象运行于服务器端为执行本次请求而创建的独立线程当中，这个独立线程只用于执行本次请求，当本次请求结束时这个独立线程的生命也随之终止。.Net Framework 提供了保存和获得执行线程上下文（Context）信息的技术。代理商服务器端子系统专门提供了一个模块，用于封装执行线程上下文的信息，此模块被称为代理商远程请求上下文。在代理商远程请求上下文中会保存一些很有用的信息，如：代理商信息、售票员信息和代理商客户端的文化信息等，这些信息在执行线程中可能是有需要的。

② 为了保证从客户端子系统向服务器端子系统发送来的远程调用请求是合法的，基于令牌的身份验证方式被加入进来。一个令牌是一种典型的由一长串字符组成的字符串，并能以某种方式解除引用，成为一个唯一的用户 ID[6]。

③ 因为基于 TcpChannel 的通道是很耗资源的，因此在服务器端子系统增加了一个会话（Session）机制。本会话机制规定了许多远程调用的规则，如：若客户端子系统在发送上次请求 20 分钟之后没有向服务器端发送下次请求，本次会话直接结束；代理商售票员只有重新登陆之后才可以重新发送下一次请求了。

④ 客户端和服务器端之间交互的数据是需要在网络上传输的，为了提高传输性能，压缩机制被引入近来。实现如下：数据从应用程序域传入网络之前，先进行压缩；数据从网络中传入应用程序域之后，先进行解压缩。

图 2.2 是一次完整的远程调用的顺序图，它描述了客户端对象是怎样与服务器端远程对象进行通信的。远程对象 RMIProxy 提供了一个公共的远程方法，此远程方法的签名有：服务器端远程业务类的完全限定名（包括程序集名称和业务类名称），服务器端远程业务类的方法名和方法所需的参数列表；此远程方法有一个 Object 类型的返回值。远程对象 RMIProxy 在接收到客户端的请求后，分析请求中的信息，找到远程业务类并且通过反射动态生成远程业务类的一个实例对象，之后执行业务对象的方法，最终返回调用结果了。类 RMIProxy 可以通过单件模式（Singleton）来实现，因为利用它的

一个实例就可以解决问题了。单件模式要求一个类有且仅有一个实例，与此同时，提供了一个全局的访问点[7]。

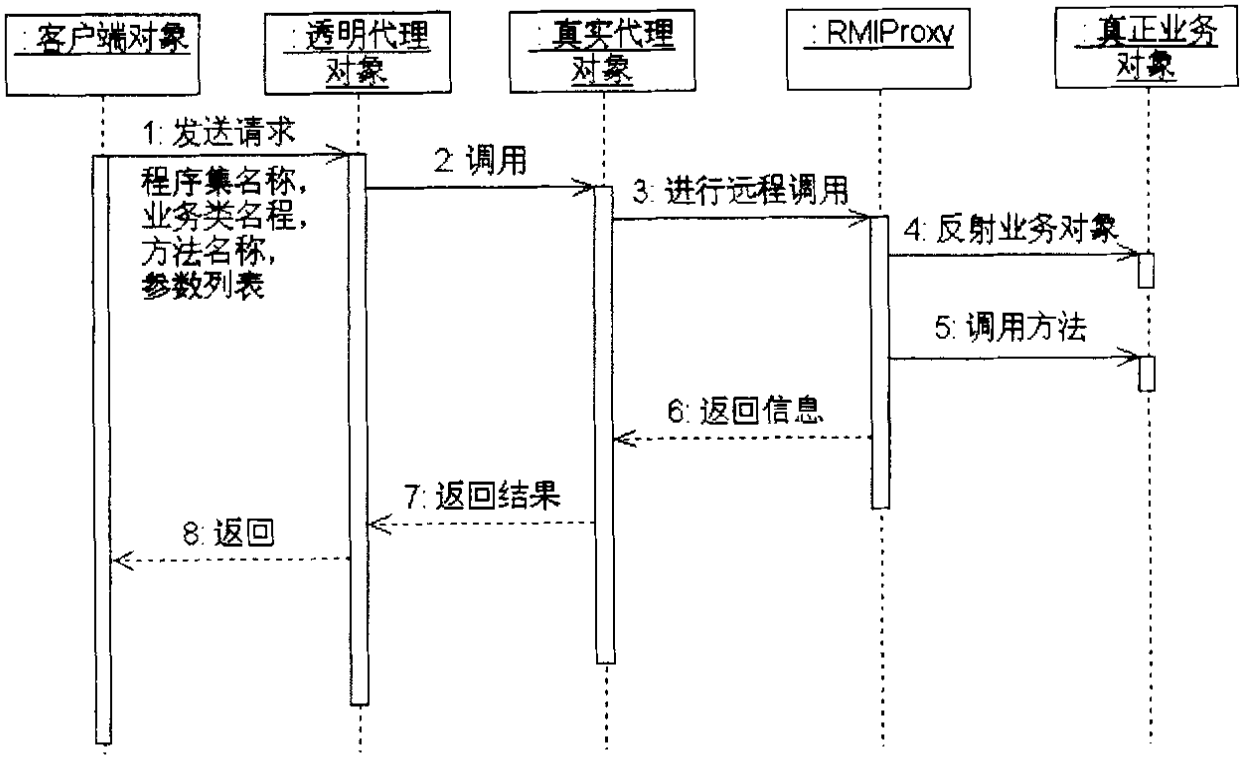

图 2.2 远程调用的顺序图

Fig. 2.2 Long-distance transfer sequence chart

2.2 设计数据访问层和实现层间传递数据

2.2.1 数据访问层的介绍

在.Net 平台下，数据访问层是指通过 ADO.Net 完成对数据库中数据的 CRUD 操作[8]。当前，关系型数据库仍然占有绝对的统治地位（注：面向对象数据库还不成熟），而面向对象编程语言的应用越来越广泛，为了解决这两者之间的交互，许多优秀的技术和框架被引入软件开发中，例如：Hibernate 框架在 J2EE 中的成功应用。在.Net 平台的开发环境下，也有一些优秀的数据层框架，著名的就有：微软的 Data Access Application Block(即:DAAB)和 NHibernate。简单分析一下这两种框架技术，Data Access Application Block 是微软开发的一套企业级组件库，它包含优化的数据访问代码，可以帮助用户调用存储过程以及向关系型数据库发出 SQL 文本命令。它在.NET 应用程序中被作为构造块来使用，可以减少需要创建、测试和维护的自定义代码的数量。但是，Data Access

Application Block 的整体功能比较简单，它采用过程化方法实现对数据库的操作，导致编程枯燥；而 NHibernate 是 Hibernate 框架在.Net 平台上的移植，它采用 O/R Mapping 技术实现对数据库的操作，但是它仍然不稳定，至今还没有发布正式版本。考虑到本系统的数据库比较复杂，项目小组打算开发一个轻量级的基于面向对象编程的数据访问层，其中我们将借鉴和运用 Data Access Application Block 和 NHibernate 中的一些好的思想。作者 Framework 的项目小组的目标是：开发一个简单、易扩展和封装底层数据库操作的数据访问层，用来构建一个低耦合、高内聚的系统，以提高系统的复用性和健壮性。

2.2.2 系统的数据访问层设计

本系统运用了一个数据访问包装类（类名为：DAO），DAO 封装了连接数据库、执行 SQL 命令和启动事务等操作。在系统中，不管是对数据库表进行增加、删除和修改操作，还是进行简单查询或复杂查询，都必须得到 DAO 的支持。DAO 是 Adapter 模式在系统中的一个成功运用，因为在本系统中只使用了 SQL Server 数据库，所以 DAO 仅仅包括对 SQL Server 数据库进行操作的类的封装。

在 ADO.NET 中，SqlConnection 用来连接 SQL Server 数据库，SqlCommand 用来向 SQL Server 数据库发送查询和其它操作指令，SqlDataReader 用来快速、未缓冲、只向前移动的访问数据，SqlDataAdapter 作为数据适配器，SqlTransaction 用来启动、提交事务。本系统中的 DAO 类对上述所有类的操作进行封装并且对外部公共了很多方法，这些方法用来启动事务，提交事务，执行增加、删除和修改操作，进行各种查询工作。

在.Net 平台下采用的是垃圾收集器机制，通常不需要担心不再有用的对象，只要让这些对象的所有引用都超出作用域，垃圾收集器就会开始自动释放这些对象所占有的资源。但是，垃圾收集器释放资源的时间是随机的，对于那些占用了宝贵而重要资源的对象（特别是 DAO 对象，一个 DAO 对象会与数据库建立一个连接），需要尽可能快的释放这些资源，为此类似于 DAO 的类需要实现 IDisposable 接口。IDisposable 接口定义了一个模式，为释放非托管的资源提供了确定的机制。IDisposable 接口声明了一个方法 Dispose()，Dispose()的执行代码显式释放由对象直接使用的所有非托管的资源，并在所有实现 IDisposable 接口的封装对象上调用 Dispose()方法[9]。

对关系型数据库的操作可以分为以下三类：

(1) 对单个数据表的增加、删除和修改操作

在这种情况下，一般仅对一个单独的数据库表进行操作，为此本系统采用了一种面向对象的编程模型。下面通过一个简单的示例来说明数据表是如何与数据访问实体类实现映射的。

见图 2.3 数据库表结构示例图所示，表 Order 与表 Agent、表 Order 与表 Performance 之间存在一对多的关系。以表 Order 与表 Agent 为例，一对多的关系有两层含义：表 Order 的一条记录在表 Agent 中肯定有一条记录与之相对应；表 Agent 的一条记录在表 Order 中有一组记录（可能有多个，也可能没有）之相对应。

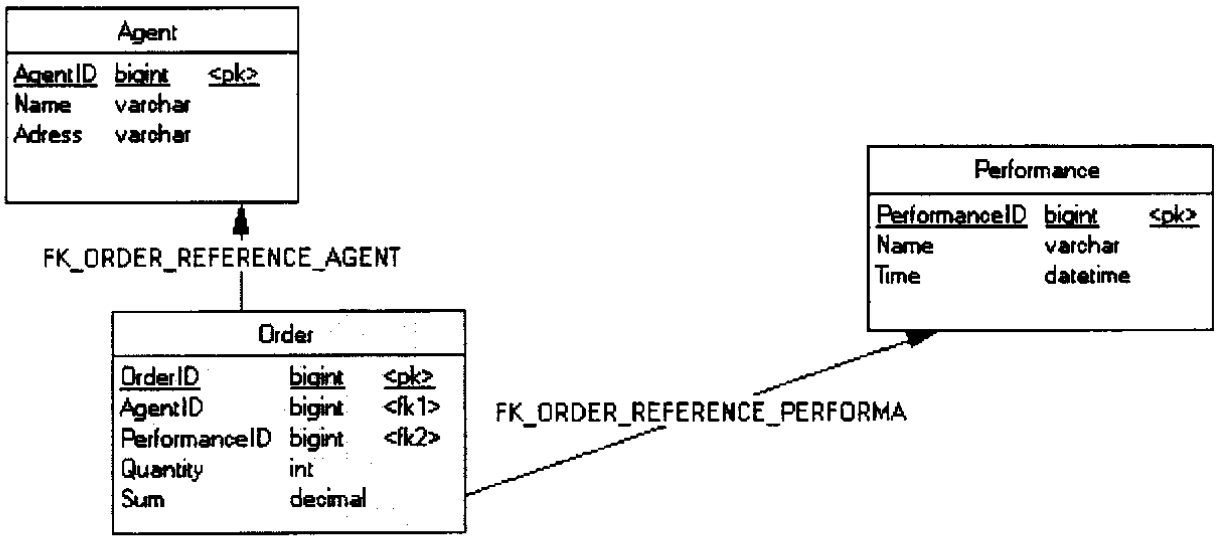

图 2.3 数据库表结构示例图

Fig. 2.3 Example of database chart structure

见图 2.4 数据访问实体类结构图所示，BaseDataObject 是一个抽象数据访问实体类，它拥有一些可被子类重写的受保护和公共类型的方法。而类 Agent、类 Performance 和类 Order 是抽象类 BaseDataObject 的具体实现类，它们是真正的数据访问实体类，它们分别是表 Agent、表 Performance 和表 Order 的映射类。它们拥有自己的属性，并且各自重写了 BaseDataObject 的数据操作方法，这些方法封装了对数据库的增加、删除和修改的操作。

一般情况下，数据库中的一个表映射为一个数据访问实体类，表中的一个字段映射为实体类中的一个属性。在数据库模型中，表 Order 与表 Agent、表 Order 与表 Performance 之间都是多对一的关系；但是在数据访问层中，只存在类 Order 依赖类 Agent、类 Order 依赖类 Performance 的单向依赖关系，而一对多的关系被去除了，这样设计是为了降低数据访问层的复杂度和编程的难度，避免了类 Order 与类 Agent 之间的双向依赖关系。

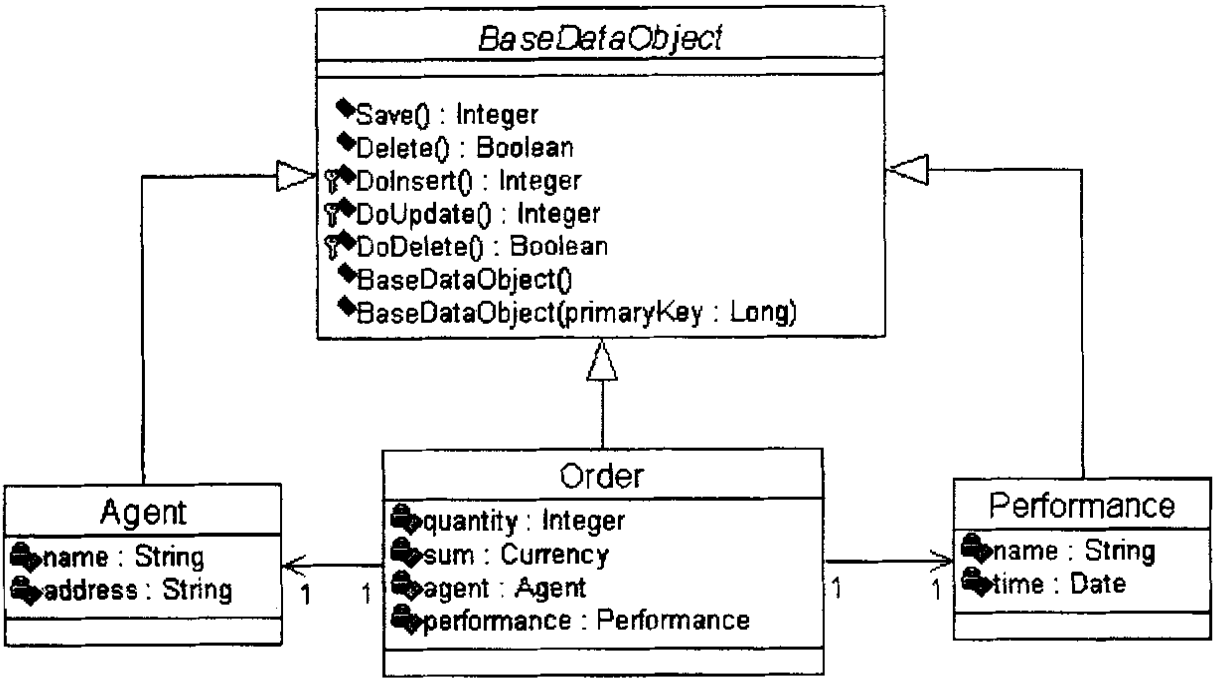

图 2.4 数据访问实体类结构图

Fig. 2.4 Data access entity class structure chart

(2) 对单表或多表进行查询

因为查询的复杂性和多变性，本系统采用直接向数据库传递 sql 命令的方式实现查询，使用类 DataSet 作为查询结果的容器。类 DataSet 是.Net Framework 提供的一种数据容器，它可被当作数据访问实体的集合。数据访问包装类 DAO 提供了一些方法，这些方法的返回类型是 DataSet。在.Net 平台下，类 DataSet 被简称为数据集。

数据集是数据在内存中的缓存容器，它不包含数据库连接的概念。数据集由一组数据表组成，每个表都有一些数据列和数据行。除了定义数据之外，还可以在数据集中定义数据表之间的关系[10]。从某种意义上来说，一个数据集就相当于一个缓存在内存中的脱机数据库。

(3) 在查询的基础上进行增加、删除和修改功能

在这种情况下，需要在查询数据之后对结果数据进行多次增加、删除和修改操作。为了保证数据的完整性，这些操作必须在一个事务中完成，结合上述两种解决方法可以实现这种功能。数据访问包装类 DAO 提供了启动事务、回滚事务和提交事务的方法，这些方法是对 ADO.Net 中事务操作的封装。

2.2.3 实现层之间的数据传递

有四种方式实现传递数据的机制：DataSet，强类型 DataSet，自定义实体类和普通 XML。在企业级应用程序的上下文环境中，XML 是弱类型、难于维护、效率低下的[11]。在本系统中将采用其它三种方式实现传递数据的机制。

在后台管理系统中，表现层与逻辑层之间、逻辑层与数据层之间采用上节所述的数据访问实体类和 DataSet 作为传递数据的工具。

但是考虑到代理商售票系统是一个分布式应用系统，数据要在网络中传输，尽量降低数据量的大小是很有必要的，因此在代理商客户端子系统和代理商服务器端子系统之间传递数据的对象是：业务实体类和 DataSet 数据集。业务实体类是数据访问实体类的简化类或封装类，一般情况下，业务实体类只拥有属性而没有业务逻辑。为了降低数据传输的代价，业务实体类是越简单越好。注意：在代理商客户端子系统和代理商服务器端子系统都需要部署业务实体类的组件。关于业务实体类的设计方法可见 2.2.4 节的相关内容介绍。

如图 2.5 代理商售票系统中的数据传输对象所示：

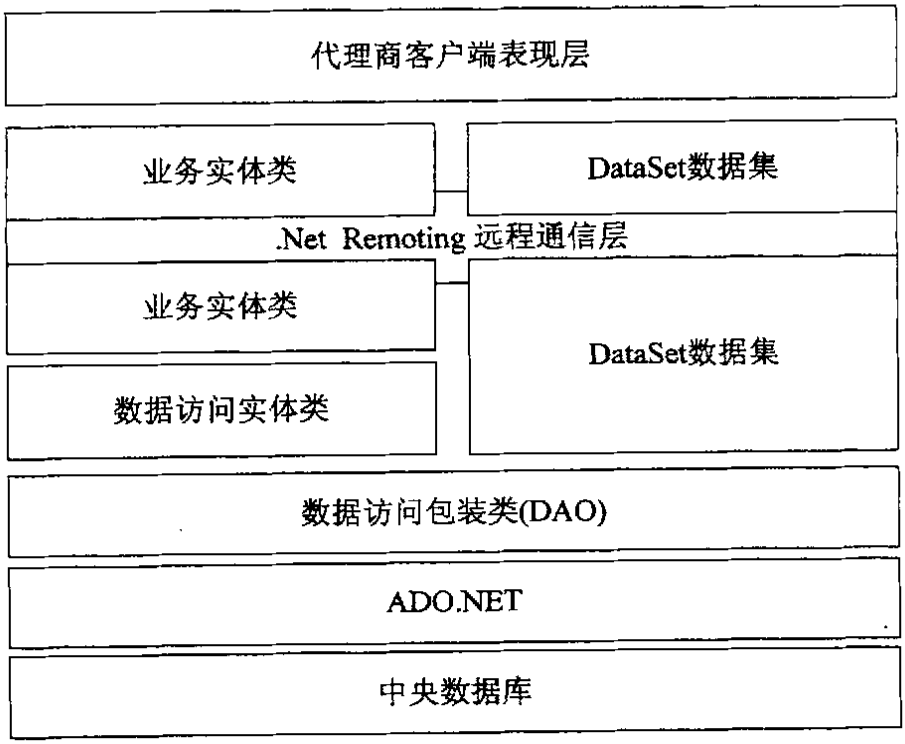

图 2.5 代理商售票系统中的数据传输对象

Fig. 2.5 Data transfer object of agent ticketing system

(1) 若对单个表的一条记录信息进行操作，则以业务实体类为传输对象；

(2) 若对单个表的多条记录信息进行操作，则以 DataSet 数据集为传输对象；

(3) 若对多个表的信息进行操作，则以业务实体类或 DataSet 数据集为传输对象，这要视具体情况而定。

2.2.4 Data Transfer Object 在远程调用中的应用

在2.2.3节中所述的业务实体类是代理商服务器端子系统和代理商客户端子系统之间传递数据的工具，按照微软的说法，它们可被称为 DTO（Data Transfer Object），中文翻译的名称为“数据传输对象”[12]。DTO 是一个为了减少远程方法调用的次数而在应用程序域间传递数据的对象，它是降低远程方法调用引起资源耗费的有效解决方法之一[13]。下面详细说明为何使用 DTO。

在一次远程调用中，首先客户端要找到远程对象所在的服务器地址，接着客户端和服务器建立连接并且发送请求信息，服务器执行请求后开始返回结果。在此过程中，传输数据所占用的时间份额是比较小的，并且随着传输数据的量变得很大时，它耗用的时间也增长不大，而寻找服务器地址和进行连接所花费的时间是很大的。因为每一次远程调用的代价都非常大，所以需要减少调用的次数，这就意味着每一次调用中都会传输大量数据。大量数据的传输的方法可以通过使用大量参数来实现，然而，这对大部分编程语言都是难以实现的，例如：Java 和 C#就只能返回一个值。解决的办法是创建一个数据传输对象，这个对象将保留所有调用需要的数据。DTO 需要被序列化以便能在网络中传输，在本系统中，DTO 类是按值编列的。

假定代理商客户端子系统要获取“代理商”、“剧场”和“演出”信息，现在考虑下面两种情况：

(1) 客户端发出三次远程调用请求，分别获得“代理商”、“剧场”和“演出”信息；

(2) 客户端只需要发出一次远程调用请求，一次性获得“代理商”、“剧场”和“演出”的所有信息。

对前一种情况，在每次请求中传输返回的数据较少，但是要进行三次远程调用请求；在后一种情况中，在一次远程调用请求中传输返回了所有的数据。经过测试，采用后一种情况所花费的时间几乎只有前一种情况所花费时间的三分之一。为此要构建一种新的数据容器，用它来封装“代理商”、“剧场”和“演出”信息，这种对象类型即为 DTO。

3 需求分析

3.1 需求概要

经过调研项目小组发现当前市场上的票务系统的功能有以下一些缺点：

(1) 售票功能简单，一般不支持可视化选择座位并完成售票；出票方式单一，一般包括客户端售票或个人 Web 售票；

(2) 报表统计功能简单，一般需要通过手工去统计售票情况；

(3) 业务流程定义不完善，很难运营大型文艺演出或体育比赛项目；

(4) 一般只能出售印刷票或使用物理防伪技术的票品，不支持电子票的出售；

(5) 一般不支持多语言的国际化版本。

本系统将保留传统票务系统的优点，并为上述缺点提供解决方案。

经过分析票务管理系统的功能得出系统的用户有以下两类：

(1) 代理商售票员。通过本系统，他们可以实现可视化选择座位并完成售票。

(2) 系统管理员。通过本系统，他们可以输入和发布演出项目信息，并对剧场信息和座位信息等进行设置，他们不负责售票。

为此，必须把整个系统的功能分为两个部分：后台管理功能和代理商售票功能。后台管理功能部分的执行者为系统管理员，代理商售票功能的执行者为代理商售票员。

3.2 功能性需求

在下面的 3.2.1 和 3.2.2 小节中分别简单描述了后台管理功能和代理商售票功能的需求情况。注：因为系统国际化支持的需求比较特殊，所以并没有在这里叙述，关于国际化支持的需求可见 5.3 小节。

3.2.1 后台管理功能的需求

(1) 后台用户输入用户名和登录密码进入代理商售票系统；后台用户有多种角色，如：普通管理员、剧场管理员、财务管理员和超级管理员等，不同角色的管理员拥有不同的权限；一个后台管理员用户可以拥有多种管理员角色。

(2) 普通管理员能实现对代理商信息、演出项目信息、订单管理、基本票品等级信息和演出主办方信息的维护；剧场管理员可以输入和编辑基本剧场信息，根据剧场的简略图设计出可视化的看台和座位信息；财务管理员可以登记所有代理商的结算信息和察看财务结算报表；超级管理员的权限最大，他除了拥有普通管理员的权限之外，还可以执行后台用户管理、系统模块定义和用户角色权限分配。

(3) 一个代理商可以拥有多个售票点，一个售票点可以有多个售票员，售票员是运用代理商售票系统的用户；不同的代理商对不同的项目的操作权限可以在系统中进行设定，当一个项目新建之后，任何代理商都不能对它进行售票，代理商只有查询项目信息的权限；代理商必须通过管理员授权后才能出售该项目的票，系统的授权包括两部分，一是对演出场次的授权（即同一个项目，代理商拥有不同场次的操作权，这样可以让业务可以更加灵活地控制代理商的打票），二是对不同等级票的授权（即代理商只能出售经过授权的售票等级的票）。

(4) 在一个演出项目登记之后，普通管理员可以新建项目和演出场次信息，并且选择与演出场次相关联的剧场座位信息、设计场次票面设计、设置票品等级的价格和定义场次套票信息等。当以上所有信息都被设置完成之后，一个场次就可以发布了，被成功发布之后的场次就可以进行售票了。当本场次的售票结束时间到了之后，售票员将不能进行卖票了。在演出结束之后，本场次就无效了，若此演出项目没有其它场次了，那么演出项目也就结束了，至此一个演出项目的生命周期就完成了。

(5) 系统提供各个演出场次和各个代理商的售票统计报表（包括汇总和明细报表信息）；系统还提供给主办方观看的演出项目的销售情况的统计报表；系统提供自动升级功能，避免了为安装新的版本而必须先删除旧版本的软件；

(6) 系统为一个演出项目的生命周期提供全面的支持和跟踪，为演出主办方、承办方和代理商的职责和权力提供强大支持。

3.2.2 代理商售票功能的需求

(1) 售票员输入用户名和登录密码进入代理商售票系统；根据客户机的文化环境，售票系统启动为相应的 UI 层界面；登录成功之后，售票员可以修改自己的用户注册信息；

(2) 在出票的过程中，每次都要查询硬件加密狗的状态，如果发现没有插加密狗,系统将拒绝出票，这样可以防止代理商使用一个加密狗多处出票的情况发生。

(3) 售票员可以通过选择分类，项目开始、结束时间，项目名称，场次名称等进行查询目前系统里有哪些正在运行的演出项目，并可以显示该项目的详细信息，包括各个场次的可售票数量，已售票数量等；

(4) 售票员选择项目和场次以及要购买的票品所在的看台及楼层之后，系统将该区域的座位、售票状态和售票等级以图形的形式在窗体中表现出来；售票员可以选择客户要购买的票（选择的票被放到“购票车”中），完成选票以后，售票员按“打印”按钮提交购票信息，服务器收到信息、确认购票信息中所选的票全部都没有被别人抢先打印后,直接为用户锁定全部票品,生成订单信息，并返回可打印的信息，之后系统直接调用

打印程序，使用每个项目场次预设的票品版面信息把票输出到打印机；如果购票信息中的部分票已被其它用户抢先打印了,服务器返回被别人已抢先打印的票的信息，并提示售票员哪些票已经被别人先打印了，并从当前“购票车”中删除这些冲突的票，售票员可以直接重新提交剩余的票打印或者重新选择其它位置的票增加到“购票车”后提交到服务器端确认打印。在确认出票时，售票员可以填入收款方式（如现金，支票，刷卡和欠费等）。

(5) 售票员在出票的过程中，可能由于打印机或者网络的原因导致没有正确打印出票，此时售票员可以提出申请重打请求；在后台管理员审核通过申请重打的票品之后，售票员就可以进行重新打票操作了。

(6) 系统拥有自动升级的功能，系统可为售票员提供快速查询订单信息的功能；同时，系统能提供代理商和当前登录售票员的售票汇总和明细统计报表信息。

3.3 非功能性需求

(1) 系统的界面美观、可操作性好，必须使用 Windows 的操作系统；窗体上的各种可视化控件拥有 XP 或 office 2003 的显示风格；系统支持对大集合数据信息的显示，并且能对之进行简单过滤和排序等操作。

(2) 售票员在代理商客户端进行操作时的响应时间最长不能超过 4 秒；

3.4 系统的用例模型

3.4.1 后台管理功能的用例模型

图 3.1 是后台管理功能的用例图，因为本部分的功能非常多并且复杂，本用例图只包括了后台管理的部分功能。

下面简单说明图 3.1 所描述的用例图。系统中有多种执行者（Actor），如：“后台用户”、“财务管理员”、“剧场管理员”、“普通管理员”和“超级管理员”等。“后台用户”只拥有登录、退出系统和管理个人信息等最基本的一些功能；“财务管理员”、“剧场管理员”是“后台用户”的泛化角色，它们除了拥有“后台用户”的功能外，各自还有其他的一些专有功能，如：“财务管理员”拥有代理商结算管理和结算统计报表等功能，“剧场管理员”拥有剧场信息管理和剧场座位管理等功能；“普通管理员”也是“后台用户”的一种泛化角色，它拥有本系统提供的绝大部分的功能，如：演出信息管理、代理商信息管理、场次票面设置管理和订单信息管理等；超级管理员是“普通管理员”的泛化角色，它们除了拥有“普通管理员”的功能外，各自还有其他的一些特殊的高级功能，如：后台用户管理和用户角色管理等功能。

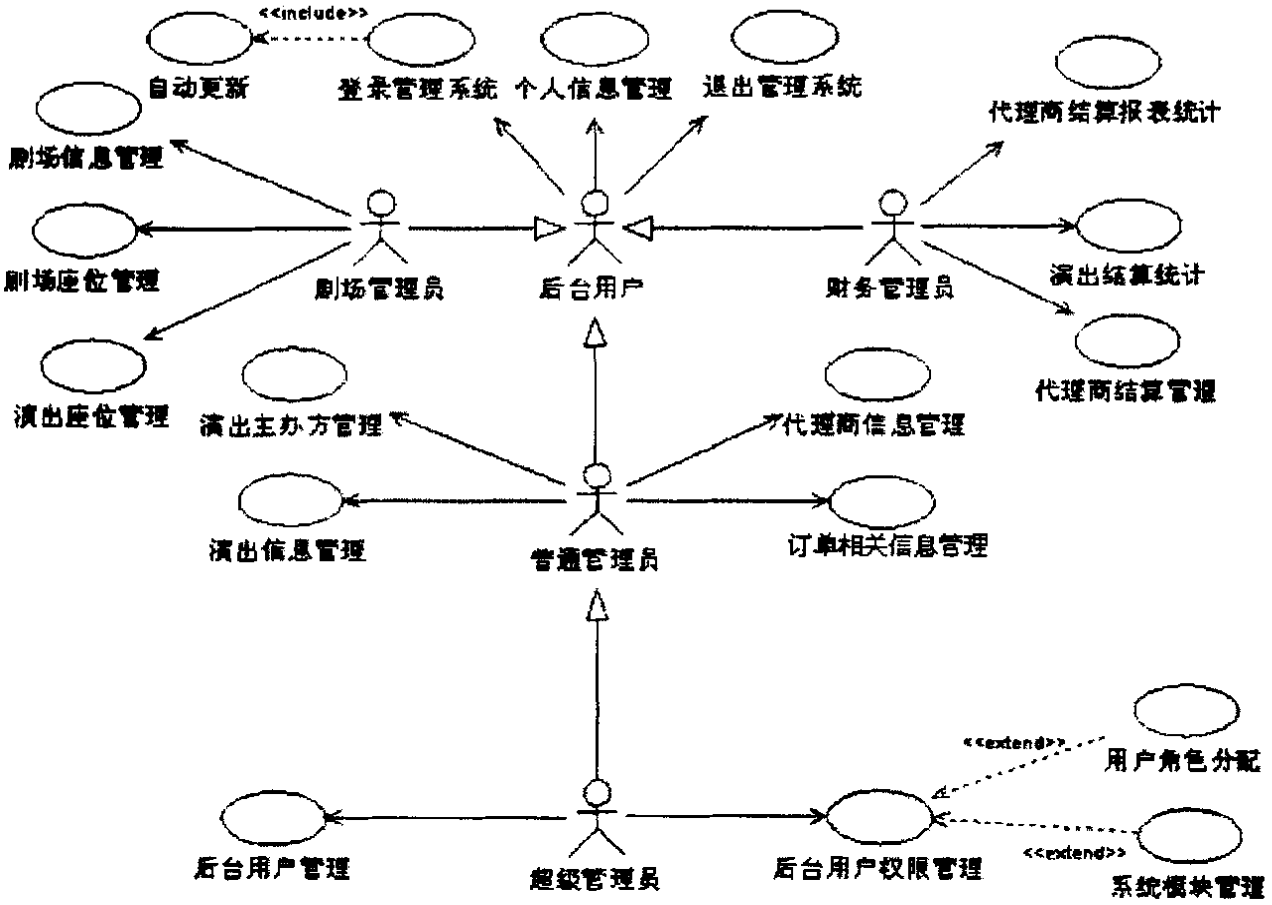

图 3.1 后台管理功能的用例图

Fig. 3.1 Background management function model chart

3.4.2 代理商售票功能的用例模型

为了保证系统的数据业务逻辑的安全性，不能把进行数据业务逻辑的具体实现部署放在代理商客户机上，为此需要一个服务器来完成那些数据业务逻辑的执行，所以需要把代理商售票功能的表现层逻辑和数据业务逻辑分别部署到客户端和服务器端。代理商售票功能可以用两个用例图来描述，即代理商客户端用例图和代理商服务器端用例图。

见图 3.2 是代理商客户端用例图。由图可见，在本用例图中只存在 “代理商售票员”这一种执行者。“代理商售票员”拥有本用例图所描述的所有功能，这些功能包括：售票员登录管理、个人信息管理、售票管理、订单查询、报表统计、申请重打和重新打印票品等。

在本用例图中，售票管理是核心功能，它由以下用例组成：“可视化售票”、“无座位售票”、“批量打印”、“票品打印”和“收取票品费用”。系统需要支持有座位

项目和无座位项目；对有座位项目的而言，为了加快售票速度，系统提供了批量打印的功能；打印出票品之后，售票员要向客户收取费用。

为了使客户端能打印出票品，用例“打印机设置”提供了配置打印机的功能。这里的订单管理功能与后台管理功能用例图中的订单管理功能不同，运用后台管理功能中的订单管理，“普通管理员”可以查询所有订单信息；运用图 3.2 中所示订单管理，“代理商售票员”只能查询本售票员所出售的订单信息。

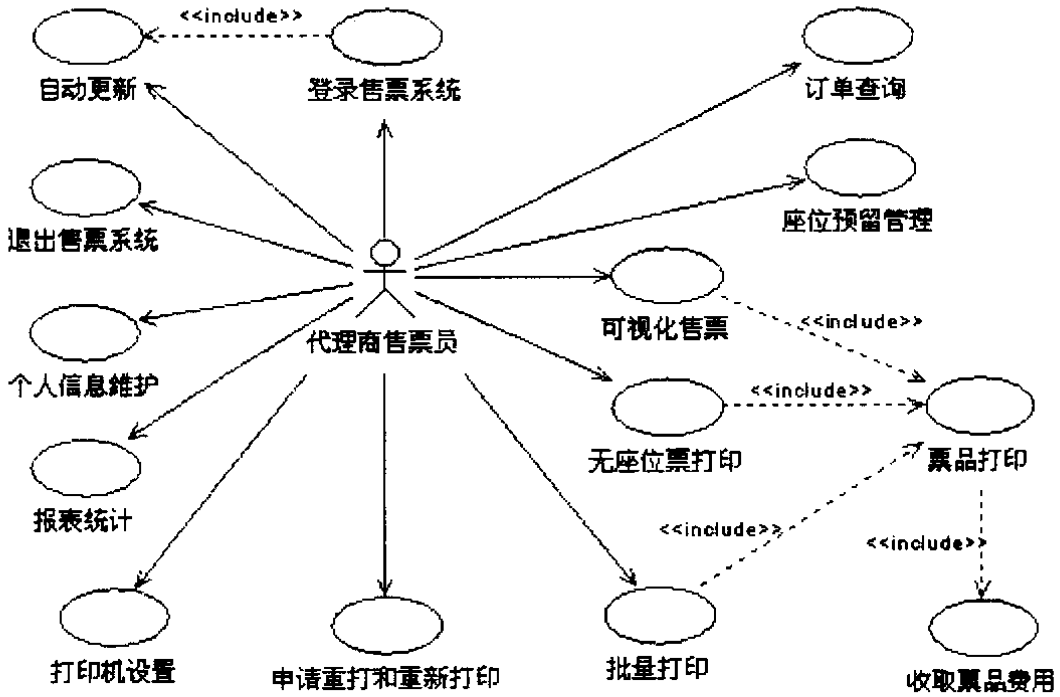

图 3.2 代理商客户端用例图

Fig. 3.2 Agent client model chart

见图 3.3 是代理商服务器端用例图。经过分析代理商售票功能的需求，项目小组决定增加本用例图，本用例图主要用来描述代理商应用服务器端的功能。

由图可见，在本用例图中只存在 “超级管理员”这一种执行者，这里的“超级管理员”与后台管理功能用例图中的“超级管理员”是同一个执行者。“超级管理员”拥有本用例图所描述的所有功能，这些功能包括：管理应用服务器和维护应用服务器等。

管理应用服务器的功能是本用例图中的核心功能，它由以下用例组成：“启动应用服务器”、“停止应用服务器”和“重启应用服务器”。

维护应用服务器的功能由以下用例组成：“查看日志情况”、“修改配置信息”和“查看服务器端状态”。用例“查看日志情况”是为了记录应用服务器的异常情况；用

例“查看服务器端状态”是为查看应用服务器的工作情况；用例“修改配置信息”是为了方便超级管理员修改服务器端的配置文件。

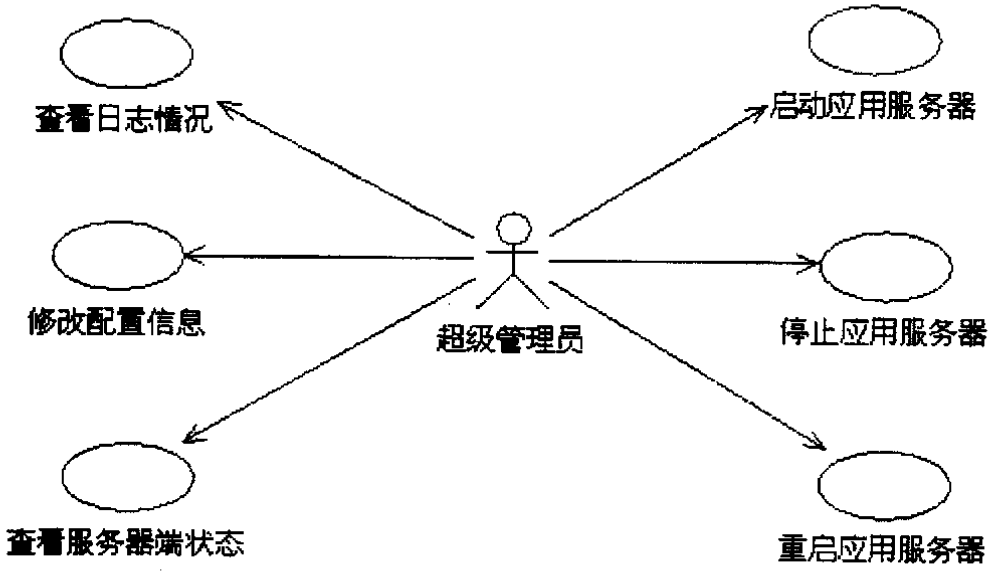

图 3.3 代理商服务器端用例图

Fig. 3.3 Agent server model chart

4 系统总体设计

4.1 系统的体系结构风格

系统采用客户端/服务器的体系结构，为了满足第三章所述的功能需求，系统分为后台管理系统和代理商售票系统两个部分，其中代理商售票系统包括代理商客户端子系统和代理商服务器端子系统。后台管理系统用来实现 3.2.1 小节中所描述的后台管理功能，代理商售票系统用来实现 3.2.2 小节中所描述的代理商售票功能。这两个系统都采用了分层的体系结构设计方法，后台管理系统是一个典型的三层应用程序，而代理商售票系统是一个基于 N 层的分布式应用程序。一个分层风格的系统按照层次结构组织，每一层向它的上层提供服务，同时又是它的下层的客户[14]。在下文的 4.2 节中，首先介绍系统的体系结构设计，接着分别详细描述后台管理系统和代理商售票系统的体系结构设计。

4.2 系统的体系结构设计

4.2.1 总体设计

系统采用关系数据库作为存储可持久化信息的方式。关系数据库提供的数据抽象要比普通文件提供的高级，在关系数据库中，数据以表的形式存储在按照预先定义好的称谓 Schema 的类型中。UML 提供了包（Package）的机制来说明元素或者子系统，一个包可以是任何种类的一组模型元素[15]。图 4.1 是用 UML 的包图表示的系统体系结构图。

可以是任何种类的一组模型元素[15]。图 4.1 是用 UML 的包图表示的系统体系结构图。

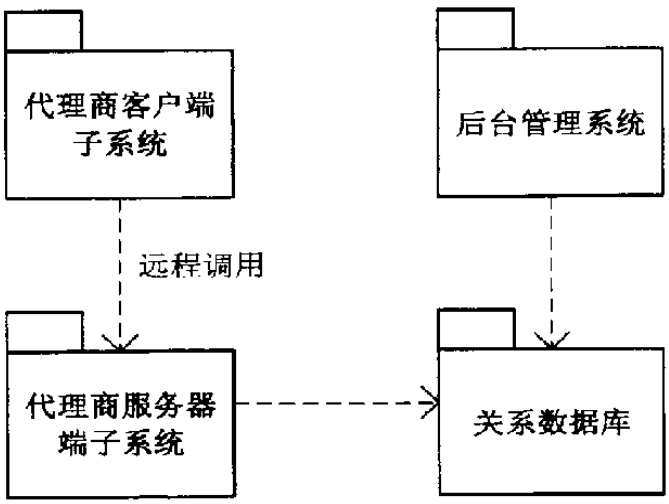

图 4.1 系统体系结构图

Fig. 4.1 System structure chart

4.2.2 后台管理系统的体系结构设计

后台管理系统是用来实现后台管理功能的系统，从分层的角度来看，它可以分为后台管理表现层、后台管理业务逻辑层和数据访问层，图 4.2 是后台管理系统的分层体系结构图。

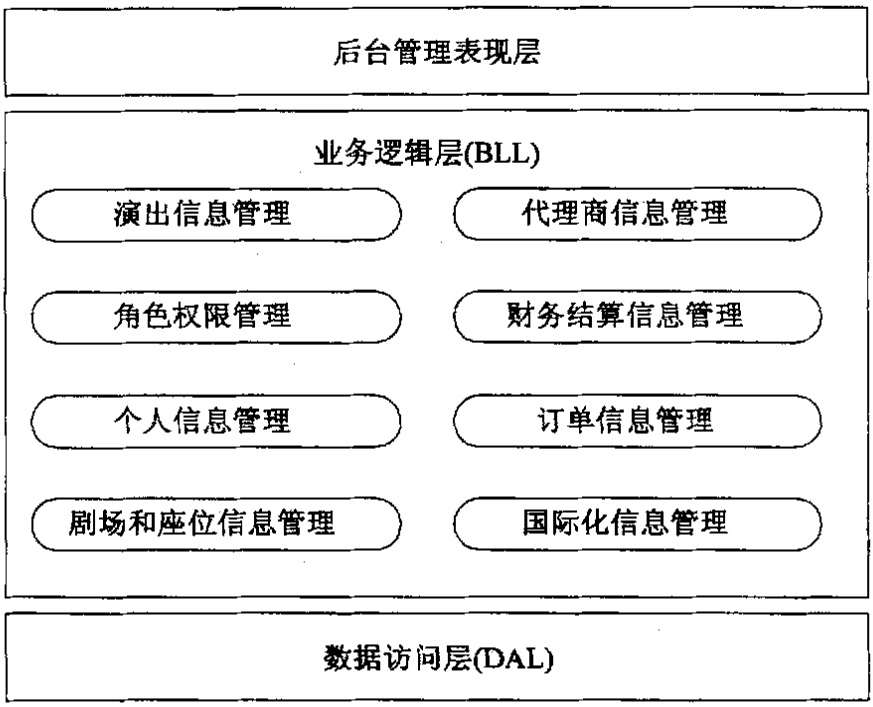

图 4.2 后台管理系统的分层体系结构图

Fig. 4.2 Delamination system chart of background management system

后台管理表现层：表示层提供应用程序的用户界面 IGUI)，表示层通常包括 t indows 窗体（用于智能客户端应用程序）和 ASmNET 技术（用于基于浏览器的交互）的使用[16]。在后台管理表现层中只有 t indows 窗体，这些窗体是后台管理员进行操作数据的可视化界面。

后台管理业务逻辑层：业务逻辑层用来实现应用程序的业务功能，它通常由使用一种或多种支持.NET 的编程语言实现的大量组件组成[16]。后台管理业务逻辑层是后台管理系统中的核心层，它用来实现后台管理的诸多复杂逻辑，在图 4.2 中只画出了后台管理系统中的部分业务逻辑。

数据访问层：数据访问层提供对外部数据系统（如数据库系统）的访问和操作，该层涉及到的主要.NET 技术是 ADO.NET。为了降低开发难度和加快编程速度，系统采用了一套高效的数据访问层技术，关于数据访问层的详细信息可见 2.2 节的内容。

4.2.3 代理商售票系统的体系结构设计

代理商售票系统是用来实现代理商售票功能的系统，它是一个分布式应用程序，它分为代理商客户端子系统和代理商服务器端子系统。从分层的角度来看，代理商客户端子系统可以分为代理商客户端表现层、业务逻辑层、.Net Remoting 远程通信层；代理商服务器端子系统可以分为.Net Remoting 远程通信层、代理商服务器端控制台层、远程对象逻辑层和数据访问层。为了使代理商客户端子系统和代理商服务器端子系统进行通信和交互，在它们之间增加了一个通信层（即.Net Remoting 远程通信层）。图 4.3 是代理商售票系统的分层体系结构图。

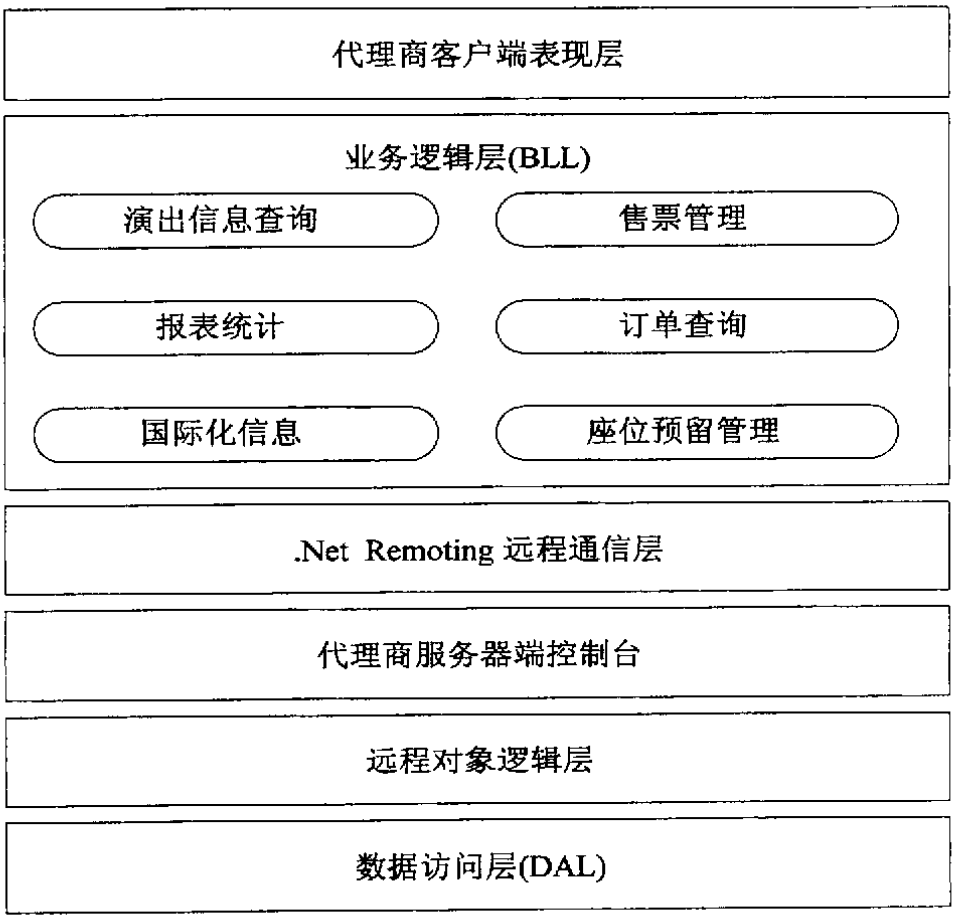

图 4.3 代理商售票系统的分层体系结构图

Fig. 4.3 Delamination system structure chart of agent ticketing system

代理商客户端表现层：在本层中只有 Windows 窗体，这些窗体是代理商售票员进行售票和其他操作的可视化界面。

代理商客户端业务逻辑层：业务逻辑层用来实现应用程序的业务功能，它通常由使用一种或多种支持.NET 的编程语言实现的大量组件组成代理商客户端业务逻辑层，它是代理商客户端子系统中的核心层，它用来实现代理商客户端子系统中的诸多复杂逻辑，在图 4.3 中只画出了部分的业务逻辑。

.Net Remoting 远程通信层：代理商客户端子系统和代理商服务器端子系统都拥有远程通信层，但是远程通信层在这两个子系统中的用处是不相同的。在代理商服务器端子系统中，远程通信层用来监听和接收代理商客户端子系统发来的远程方法调用请求，之后远程通信层传递请求信息给远程对象逻辑层，远程对象逻辑层调用服务器端的数据访问层并且获得结果数据，最后远程通信层返回结果数据给代理商客户端子系统；在代理商客户端子系统中，远程通信层用来发送远程方法调用请求给代理商服务器端子系统，它最终得到代理商服务器端子系统返回的结果数据。关于.Net Remoting 远程通信层的详细信息可见 2.1 节的内容。

代理商服务器端控制台层：本层作为代理商服务器端子系统的宿主应用程序域。通过它，系统的超级管理员可以实现启动和停止代理商服务器端子系统的运行。

远程对象逻辑层：放置代理商服务器端子系统中的远程对象和相关业务逻辑的层次。本层次是代理商服务器端子系统中的核心层，它是实现远程方法调用的执行者。

代理商服务器数据访问层：代理商服务器端子系统的数据访问层与后台管理系统中的数据访问层的构建方法一样的。

4.3 基于组件的软件开发方法

面向对象技术为软件工程的基于组件的过程模型提供了技术框架。面向对象技术强调类的创建，类封装了数据和用于操作该数据的方法（即算法）。如果经过合适的设计和实现，面向对象的类可以在不同的应用及基于计算机的系统的体系结构中复用[17]。组件和类之间的一个很大差别在于前者可以由众多后者组成。组件是类的进一步完善，因为组件是从简化软件开发的出发点来进行设计的，可以通过支持基于工具的组合模型来简化软件开发。基于组件的开发（CBD）模型融合了螺旋开发方法的许多特性，它的开发模型是利用预先包装好的软件组件来构造应用的。通过.Net Framework 框架，设计和开发组件非常容易，一般情况下一个程序集就是一个组件。

本系统采用了分层的软件体系结构，按照层次来划分，组件可以分为表现层组件（即前端组件）、中间层组件和后端组件；同时每个层可由多个组件组成，组件之间可能存在各种依赖关系。表现层组件包括胖客户端的 Windows 窗体组件和瘦客户端的 Web 页面组件，中间层组件包括 Web 控件、Web 服务组件和 Windows 服务组件等，后端组件包括

数据库服务组件、文件服务组件和网关服务组件等[18]。在本系统中大量用到了表现层组件和后端组件，在这里后端组件主要是数据层组件。所以在本节中主要介绍表现层组件和数据访问层组件的设计方案。

4.3.1 表现层组件的设计

系统中用到了自定义表现层组件和第三方表现层组件。第三方表现层组件是其它公司开发的基于.Net 平台的商业组件；自定义表现层组件是本项目小组开发出来的，它以.Net 框架自带的表现层组件和第三方表现层组件为基础进行二次开发而来。

(1) 第三方表现层组件的应用

因为本系统的表现层界面要满足以下的一些要求：

① 界面美观，可操作性好；各种可视化控件拥有 XP 或 office 2003 的显示风格；

② 支持对大集合数据的显示，并且能对之进行简单过滤和排序等操作。

当前有两种.Net 的 GUI 第三方组件占有很大的市场，即：Infragistics NetAdvantage 和 DevExpress .Net 等。经过比较和测试，最终项目小组决定采用 Infragistics NetAdvantage 组件库。NetAdvantage 是目前为止最为完整的表示层组件集，它所提供的组件集可用于构建基于 Windows 应用程序、XML web services、和 web 解决方案的界面。无论是创建 Microsoft 环境下精美、强壮的 GUI（图形用户界面），还是在 COM 或是 ASP.NET 环境下开发，NetAdvantage 都能提供适用的工具集。它的功能组件主要有：数据网格（Grids）、图表、进度条、导航条和工具条等。

在下文的(2)小节介绍的自定义组件中的窗体基类 BaseMainFrm 和 BaseChildFrm 中用到了大量的 Infragistics NetAdvantage 的控件，这些控件的结构模型、编程方式与 VS.Net 设计器中的控件完全一样，这为可视化设计表现层界面提供了便利。

(2) 自定义表现层组件的设计

任何应用程序的前端就是显示用户界面的地方，表现层组件基本上都是 GUI 组件，在本系统中，表现层组件是 Windows 窗体组件。

本系统的后台管理系统和代理商客户端子系统都是基于 MDI 的应用程序，所有的主窗体都有很多共用性，如：菜单栏，工具栏等；所有的子窗体也有很多共用性，如：进度条、界面风格和可浮动工具栏等。为此，项目小组建立了一个.Net 工程，本组件中包含了所有的表示层基类（如：窗体，对话框和进度条等）和一些辅助类。窗体基类有 BaseMainFrm 和 BaseChildFrm，它们都是对.Net 的基本类 Form 的扩展类，其中，BaseMainFrm 是 MDI 应用程序中所有主窗体的基类，BaseChildFrm 是 MDI 应用程序中所

有子窗体的基类。表现层组件的作用是：保证系统界面风格、布局的统一性，同时最大程度降低模块的藕合性、提高模块的可重用性。

4.3.2 数据访问层组件的设计

(1) 自定义数据访问层组件的设计

多层应用程序的后端是关于传送数据、处理事务以及管理企业信息的所有问题。在本系统中，数据访问层组件构成了后端组件，下文将运用上文的 2.2 节的相关内容来介绍数据层组件，这里用到了三种数据访问层组件。

① 数据访问逻辑组件：数据访问逻辑组件从数据库中检索数据并把实体数据保存回数据库中，数据访问逻辑组件还包含实现数据相关操作所需的所有业务逻辑。逻辑层是通过数据访问逻辑组件来实现对数据库数据的操作的。

② 数据实体组件：数据实体用来表示演出、座位和订单等现实世界中的实体对象。在应用程序中表示这种数据实体的方法非常多，例如 XML、DataSet、面向对象的自定义类等，这取决于应用程序的物理和逻辑设计限制。本系统中的数据实体组件包括：DataSet、强类型 DataSet、数据访问实体类和业务实体类。

③ 数据访问帮助组件：如果应用程序包含多个数据访问逻辑组件，可以使用数据访问助手组件来简化数据访问逻辑组件类的实现。该组件可以帮助管理数据库连接、执行 SQL 命令以及缓存参数。数据访问逻辑组件仍然封装访问特定业务数据所需的逻辑，而数据访问助手组件则专注于数据访问 API 的开发和数据连接配置，从而帮助减少代码的重复。本系统中的数据访问帮助组件即 DAO（数据访问包装类）和其它一些相关辅助类。

数据访问逻辑组件代表调用程序提供对数据库执行以下任务的方法：在数据库中创建记录；读取数据库中的记录并把实体对象数据返回给调用程序；使用调用程序提供的修改后的实体对象数据更新数据库中的记录；删除数据库中的记录[19]。

通常，数据访问逻辑组件访问一个单一数据库，并封装了针对该数据库中一个表或一组相关表的数据相关操作。例如，“代理商订单”数据访问逻辑组件（由 AgentOrderBO 及相关类构成）封装了对代理商订单表、代理商订单明细表的数据操作。

下面的图 4.4 描述了这三种组件的关系。

(2) 解决数据访问中的并发问题

并发是软件开发中最为棘手的问题之一。无论什么时候，用多进程或多线程操作同一数据时，都可能会引起并发问题。在一个多客户端的分布式应用程序中，并发问题的发生是很频繁的，如本系统中的代理商售票系统。

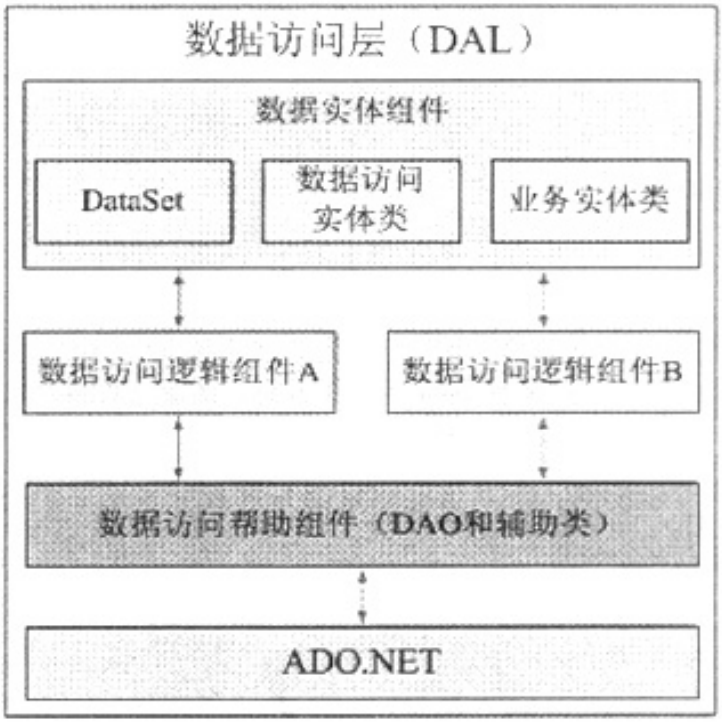

图 4.4 数据访问层组件结构图

Fig. 4.4 Data access layer component structure chart

可以考虑下面一种情形，现在有两个代理商客户端 A 和 B 同时进入可视化售票窗体，假设 A 和 B 向服务器端发送购票请求，而且很不幸的是 A 和 B 请求购买的是同一个座位 S。因为在 A 和 B 在加载可视化售票图时，座位 S 是尚未售出的，所以客户端逻辑是允许 A 和 B 提交请求购买同一个座位的。若服务器端先接收到了 A 的请求，并且同意了 A 的购票请求，接着设置 S 的状态为“已出售”；此时，服务器端接收到了 B 的购买座位 S 的请求，若服务器端在不先判断 S 的状态的当前状态情况下也同意 B 的请求，这样就会导致座位 S 会被出售两次。当前有几种解决这种并发问题的解决方案，下面将进行分析并且介绍在本系统中采用的是何种解决方案。分析完之后，再接着讨论解决 A 和 B 购买座位 S 引起的并发问题。

数据访问逻辑组件中实现管理锁定和并发的代码，管理锁定和并发的方法有两种：

① 保守式并发：为进行更新而读取某行数据的用户可以在数据源中对该行设置一个锁定。在该用户解除锁定之前，其他任何用户都不能更改该行[19]。

② 开放式并发：用户在读取某行数据时不会锁定该行。其他用户可以在同一时间自由访问该行。当用户要更新某行数据时，应用程序必须确定自该行被读取以来其他用户是否进行过更改。尝试更新已经过更改的记录会导致并发冲突[19]。

保守式并发主要用于数据争用量大以及通过锁定来保护数据的成本低于发生并发冲突时回滚事务的成本的环境。如果锁定时间很短（例如在编程处理的记录中），则实现保守式并发效果最好。保守式并发要求与数据库建立持久连接，并且因为记录可能被锁定较长时间，因此当用户与数据进行交互时，不能提供可缩放的性能。

开放式并发适用于数据争用量低或要求只读访问数据的环境。开放式并发可以减少所需锁定的数量，从而降低数据库服务器的负荷，提高数据库的性能。开放式并发在.NET中被广泛使用以满足移动和脱机应用程序的需要。在这种情况下，长时间锁定数据是不可行的。此外，保持记录锁定还要求与数据库服务器的持久连接，这在脱机应用程序中是不可能的。

通过上面的介绍可以知道，在本系统中采用开放式并发是非常合适的。实现开放式并发的方法有下面几种：

① 使用分布式时间戳。分布式时间戳适用于不要求协调的环境。在数据库的每个表中添加一个时间戳列或版本列。时间戳列与对表内容的查询一起返回。当试图更新时，数据库中的时间戳值将与被修改行中的原始时间戳值进行比较。如果这两个值匹配，则执行更新，同时时间戳列被更新为当前时间以反映更新。如果这两个值不匹配，则发生开放式并发冲突。

② 保留原始数据值的副本。在查询数据库的数据时保留原始数据值的一个副本。在更新数据库时，检查数据库的当前值是否与原始值匹配。

③ 使用集中的时间戳。在数据库中定义一个集中的时间戳表，用于记录对任何表中的任何行的更新。

为了降低复杂度和难度，采用第二种方法是比较合适的。现在继续讨论上面所述的客户端 A 和 B 购买座位 S 引起的并发问题。现在规定，要在更新座位表（SeatInfo）行时采用开放式并发冲突，可以发出以下 UPDATE 语句：

```
UPDATE SeatInfo Set Status= @NewValue WHERE id = 'S' and Status = @OldValue
```

在上面的 SQL 语句中，“@NewValue”表示更新的值（即为：“已出售”），“@OldValue”表示在客户端存放的原始值（即为：“未出售”）。利用这种方法就可以解决并发问题了，假定服务器端先接收了客户端 A 的请求，并把座位 S 的状态改为“已出售”；此时，当服务器端 B 的请求后，因为座位 S 的状态已经是“已出售”了而不是原始值“未出售”了，所以 B 的购买请求失败。

这种解决并发问题的方法的思想为：如果原始值与数据库中的值匹配，则执行更新。如果某个值被修改，WHERE 子句将无法找到相应匹配，从而更新将不会修改该行。

4.3.3 系统的组件视图

根据上面几个小节所述的内容，可以用图 4.5 来表示系统的组件视图。组件“Adin”是后台管理系统的表现层组件，“Agent”是代理商售票子系统的表现层组件，“Component”即 4.3.1 的(1)小节中提到了表现层基类组件。

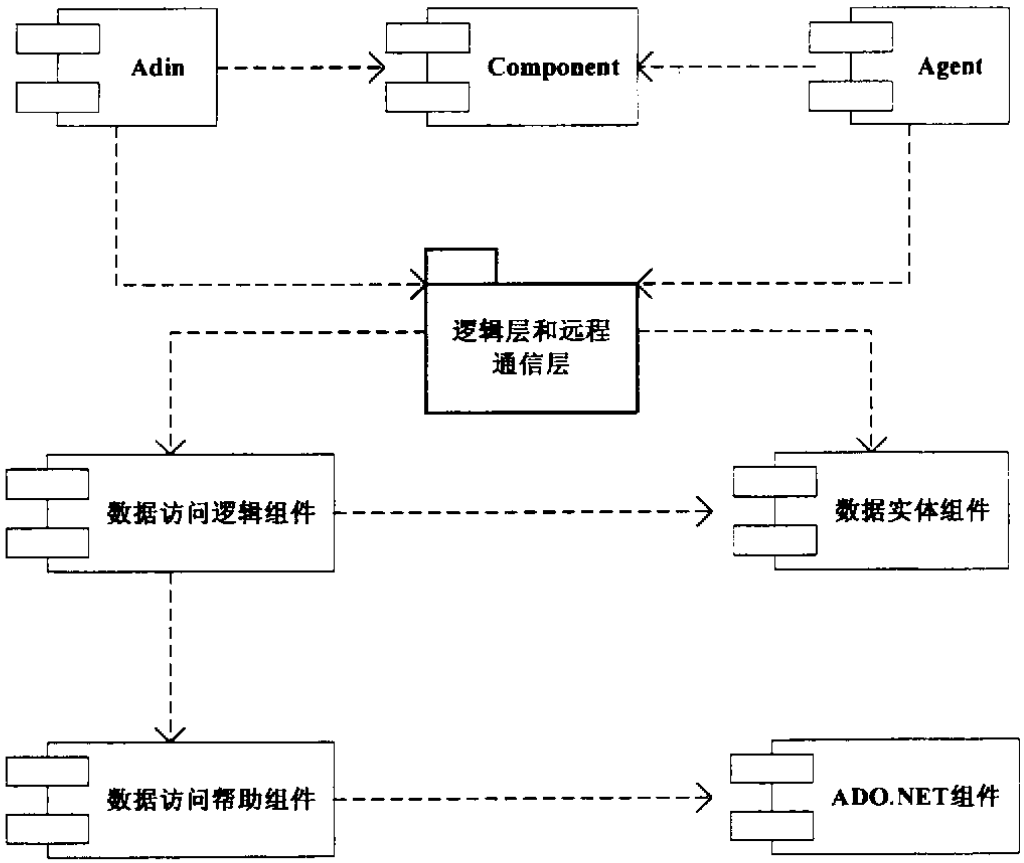

图 4.5 系统组件结构图

Fig. 4.5 System component structure chart

4.4 系统中其它方面的设计

4.4.1 二维条码技术在电子票中的运用

二维条码是实现电子票的核心技术，目前实现二维条码的编码技术主要有 PDF417 码、Code49 码、Code 16K 码、Data Matrix 码和 QR code 码等，它们主要可分为堆积和矩阵式两大类别。为了保证打印票品的安全性和唯一性，本系统的二维条码将包括尽量多的信息。项目小组最终决定采用 QR code 码，QR code 码是日本 Denso 公司于 1994

年9月研制的一种矩阵二维码，它具有其它二维条码所具有的信息容量大、可靠性高，同时还可表示图像和汉字多种文字信息、保密防伪性强等优点。目前市场上的大部分条码打印机都支持QR code条码，其专有的汉字模式更加适合我国应用。

票品中的二维条码是唯一的，因为在二维条码中包含了本票品所拥有的一些唯一信息（如：订单信息、演出场次信息、场馆信息和座位信息等）。并且，电子票采用的是经过特殊加密算法得到的二维条码，这使得每个电子票的数据均不可人为制造，即使使用相同的二维条码生成方法，创造了一个新的二维条码，它的数据也不可能和本系统的电子票上的数据完全一样。用户购买的票品及座位信息都包含到该二维条码中，入场时，用户只需要出示带有二维条码的电子票/手机票进行检验即可换票入场。虽然电子票可以被多次打印或复制，但是只有第一次成功扫描的电子票被视为有效门票。

4.4.2 可视化售票功能的设计

系统的可视化售票功能将采用灵活的座位分区设计，支持任意大的演出场所通过可视化的图形界面可直观明了地选择座位，同时提供对座位图的多种查看选择方式；具有座位类别颜色、座位状态标识、座位信息实时统计、鼠标座位信息气球等多种人性化功能；预定票品数量及相应金额即时体现。

在.Net Framework中要运用GDI+的相关技术来实现上述功能。GDI是Windows API中处理图形的部分，GDI+是基于GDI，GDI+把GDI的功能包装起来并集成到.Net Framework中[20]。

可视化售票功能主要包括以下方面：

(1) 在后台管理系统中，提供一个可视化的剧场的座位设计模块，用于设置座位的坐标位置、座位对应的价格等级名称等信息，设置过的座位及相关信息被持久到数据库中；

(2) 在后台管理系统中，提供一个可视化的演出场次的座位设计模块，用于设置座位的价格、对应的价格等级颜色、套票座位、赠票座位和工作票座位等信息，设置过的座位及相关信息被保存到数据库中；

(3) 在代理商客户端子系统中，提供一个可视化的演出场次的座位售票模块。在可视化座位售票模块中，一个售票员可以通过点击鼠标选择和取消座位，被选中座位的打印需要票品打印模块来实现。

4.4.3 设计系统的异常管理框架

(1) .Net框架的异常管理机制

在.Net Framework 环境下，异常通常由应用程序或运行时（公共语言运行时和应用程序运行库）引发的。当发生异常时，系统或当前正在运行的应用程序通过引发包含关于该错误的信息的异常来报告异常信息。异常发生后，将由运行的应用程序或默认的异常处理程序进行处理。所有异常的类型都直接或间接从 Exception 类继承而来，其中包括两种主要的异常类：

① ApplicationException，它是用户自定义的应用程序异常类型的基类，它直接继承于 Exception，但是不提供扩展功能，必须开发 ApplicationException 的派生类，用来实现自定义异常的功能。如果自定义的异常覆盖了应用程序所独有的错误情况，就应该使自定义异常直接或间接派生于 ApplicationException 类。

② SystemException，它是预定义的公共语言运行时异常类的基类。通常由.Net 运行时抛出，或者有着非常一般的本质、可以由几乎所有的应用程序抛出。

ApplicationException 和 SystemException 这两个异常类型构成了几乎所有的应用程序和运行时异常类的基础。.Net Framework 是通过异常类型来辨认异常的，应用程序建立源于 ApplicationException 的层次结构的异常体系。层次结构的自定义异常体系对应用系统带来以下好处：易于开发，因为通过继承派生异常可以扩充自己的属性；当系统部署完毕时，仍然可以通过继承扩展新的自定义异常。无须更改已经写好的异常处理代码，因为扩展的异常派生于基类异常，对基类异常的处理也就是对它派生的异常处理[12]。

要架构一个结构良好、维护性高、富有弹性的应用程序系统就必须采用一套适当的异常管理策略。一般情况下，应用程序系统的异常管理框架应该包含以下功能：

① 探测和捕获异常

.Net 公共语言运行库时提供一种异常处理模型，该模型基于对象形式的异常表示形式，它把程序的相关代码分成三种不同类型的代码块，即：try 块、catch 块和 finally 块。try 块包含的代码组成程序的正常运行部分，但可能遇到某些严重的异常情况；catch 块包含的代码用于处理各种异常情况，而这些异常是由 try 块中的代码执行时遇到的；finally 块包含的代码用于清理资源或执行要在 try 块或 catch 块末尾必须执行的其它操作，无论是否发生异常，都会执行 finally 块。对一个 try 块，可以有一个或多个 catch 块，每个 catch 块都设计为处理一种特定类型的异常，或者将一个 catch 块设计成捕获比其他块更具体的异常。

② 记录异常日志、发送异常信息

当一个异常发生时，一般需要要把此异常信息记录下来以便以后的改进工作。同时也需要想上面的调用方进行异常传播，有三种途径来传播异常：

让异常自动传播

捕获抛出的异常

捕获、包装、抛出已包装的异常

③ 产生异常事件，使表现层能够监测和引发响应

一旦发生了异常情况，正常流程就不能完成了，操作用户就不能看到期望的结果了，因此需要向用户发出提示信息，表明操作已经完成，但是发生了错误，并且提示错误类型。

(2) 系统的异常管理机制的设计

在本小节中，将描述系统的异常管理框架的设计：

① 在系统中的逻辑层和数据访问层中可能会产生异常，在所有可能发生异常的代码处都要用 try 块、catch 块和 finally 块探测和捕获异常。

② 当捕获到异常信息之后，需要做以下两部分的工作：

把异常信息以及原因和发生异常的源代码位置记录下来，系统运用了一个开源工具 log4net 作为异常日志记录工具。log4net 的异常日志记录了所有的异常信息，这些信息对系统的维护和升级工作提供支持。

因为在逻辑层和数据访问层中产生的异常一般都是关于数据库和其它逻辑方面的，这些异常信息是非常专业化的，不应该直接展现给用户，所以需要通过包装异常之后再向上抛出异常。这是就需要自定义异常类了，自定义异常类包含的信息应该是比较人性化的，是能被用户所能接受和理解的。

③ 在表现层中，也用 try 块、catch 块和 finally 块探测调用了逻辑层和数据访问层的相关代码块。所以在表现层中捕获到的异常都是自定义的包装类了，所以一旦捕获到了异常，就可以直接向用户展现异常信息了。

5 系统的详细设计与实现

5.1 后台管理系统的设计与实现

5.1.1 概述

后台管理系统中的所有功能模块主要分为三类：

(1) 有一大部分功能模块是对数据库表进行查询及显示查询结果、增加、删除和修改操作，如：代理商信息管理模块、售票员信息管理模块、主办方演出场次信息管理模块、剧场信息管理模块、财务结算管理模块和场次座位信息管理模块等；

(2) 有一些功能模块是对数据库表进行复杂查询及显示查询结果、删除操作，如：订单信息管理模块；

(3) 还有一些特殊功能的模块，如：演出票面设置信息管理，后台用户角色分配管理模块、自动升级模块和报表统计模块等。

后台管理系统的模块很多，这里不可能对所有的模块进行详细分析，下面仅仅通过两个模块的具体实现作为例子来说明。用售票员信息管理模块的实现描述第一类模块，用演出票面设置管理模块的实现描述第三类模块。

5.1.2 售票员信息管理模块的设计与实现

(1) 表现窗体的设计

可见图 5.1 所示，在本模块的窗体上，需要两个表格（.Net 中称为 DataGrid）来显示代理商信息和售票员信息，因为在数据库中代理商数据表和售票员数据表之间存在一对多的关系，所以窗体上的两个 DataGrid 之间也存在父子关系。在窗体上提供“查询”、“增加”、“删除”和“修改”按钮。

(2) 业务逻辑类的实现

业务逻辑类的作用是：控制窗体及控件的显示，通过调用数据访问类的接口获得数据并绑定数据到 DataGrid 上去。

(3) 数据访问类的实现

数据访问类的作用是：封装对售票员数据表进行的查询、增加、删除和修改操作。一般情况下，查询的返回结果是一个数据集合，所以要用 DataSet 作为存储数据的容器；每次只能对一条售票员信息进行增加、删除和修改操作，此时就可以通过 2.2.2 小节中所述的数据访问实体类“AgentUser”提供的接口实现了。本模块的数据访问类被放在数据访问逻辑组件中，数据访问实体类“AgentUser” 被放在数据实体组件中。

图 5.1 售票员信息管理模块的窗体

Fig. 5.1 Window of conductor information management module

下面通过“查询售票员信息”的顺序图来进行描述，可见图 5.2 所示，图中类“AgentUserBO”为数据访问类。

5.1.3 演出票面设置的管理模块的设计与实现

本模块的功能是为演出场次的票品设置票面信息，票面设置的详细内容包括三类：

文本型元素。它在一个票面设置上作为固定文字的信息存在，如：“座位”，“演出时间”、“场馆”和“票价”等文字标签。

图形元素。它在一个票面设置上作为固定图形的信息存在，如：票品的背景图形。

系统型变量元素。它在一个票面设置上作为动态文字或图形的信息存在，如：“座位”，“演出时间”、“场馆”、“座位”和“票价”等信息，它们必须从数据库中获取。在票面设置中，“二维条码”是一个图形系统型变量，它是由一套算法计算获得的图形。

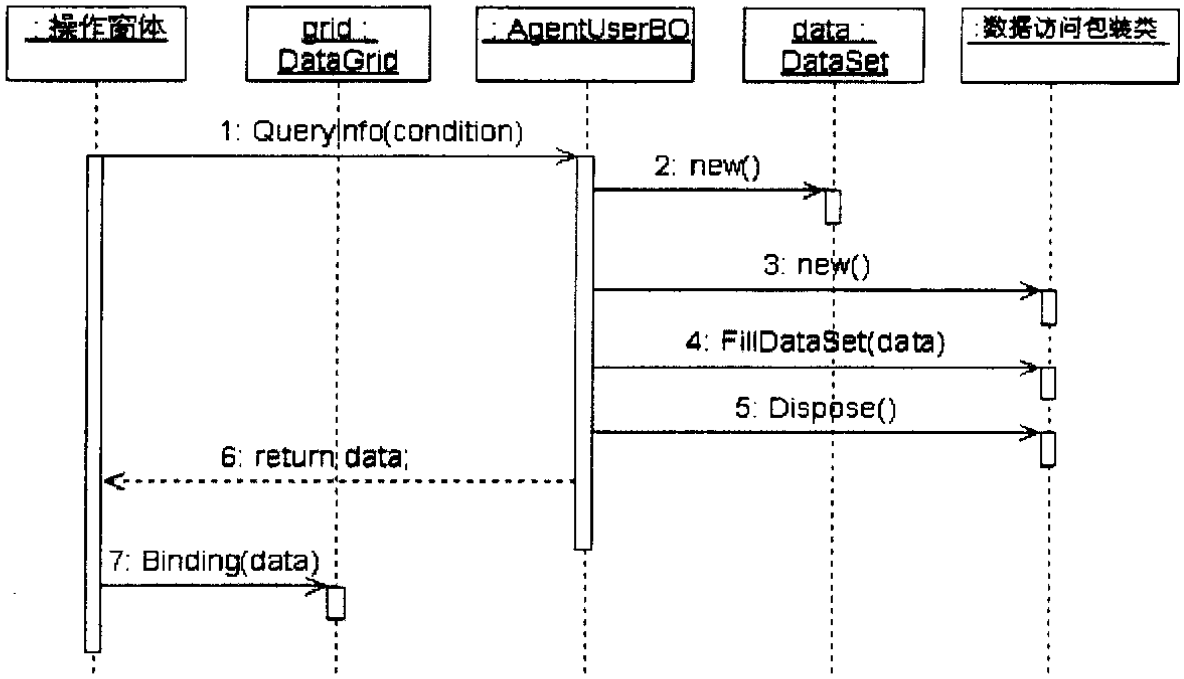

图 5.2 查询售票员信息和进行数据绑定

Fig. 5.2 Inquire conductor information and data binding

(1) 表现窗体的设计

可见图 5.3 所示，在本模块的窗体上有三个区域，左边区域为票面设置的详细内容的类型（即上文所说的三种类型），中间区域为票面设置内容的可视化显示，右方区域用于设置各个类型的票面信息的“大小”、“高度”和“字体”。

(2) 业务逻辑类的实现

业务逻辑类的作用是：控制窗体及控件的显示，通过调用数据访问类的接口对数据库表进行操作。

(3) 数据访问类的实现

数据访问类的作用是：封装对票面设置数据表进行的查询、增加、删除和修改操作。本模块所操作的数据库有表“PerformTicketSetting”和表“TicketSettingDetail”。表“PerformTicketSetting”中存放的是场次票面设置信息，即票面的大小、高度和打印方向等，一个演出场次只能有一条场次票面设置信息；表“TicketSettingDetail”中存放的是票面设置的详细内容信息（即上文所说的三种类型）。当进行查询、增加、删除和修改时，要对表“PerformTicketSetting”和表“TicketSettingDetail”同时操作。

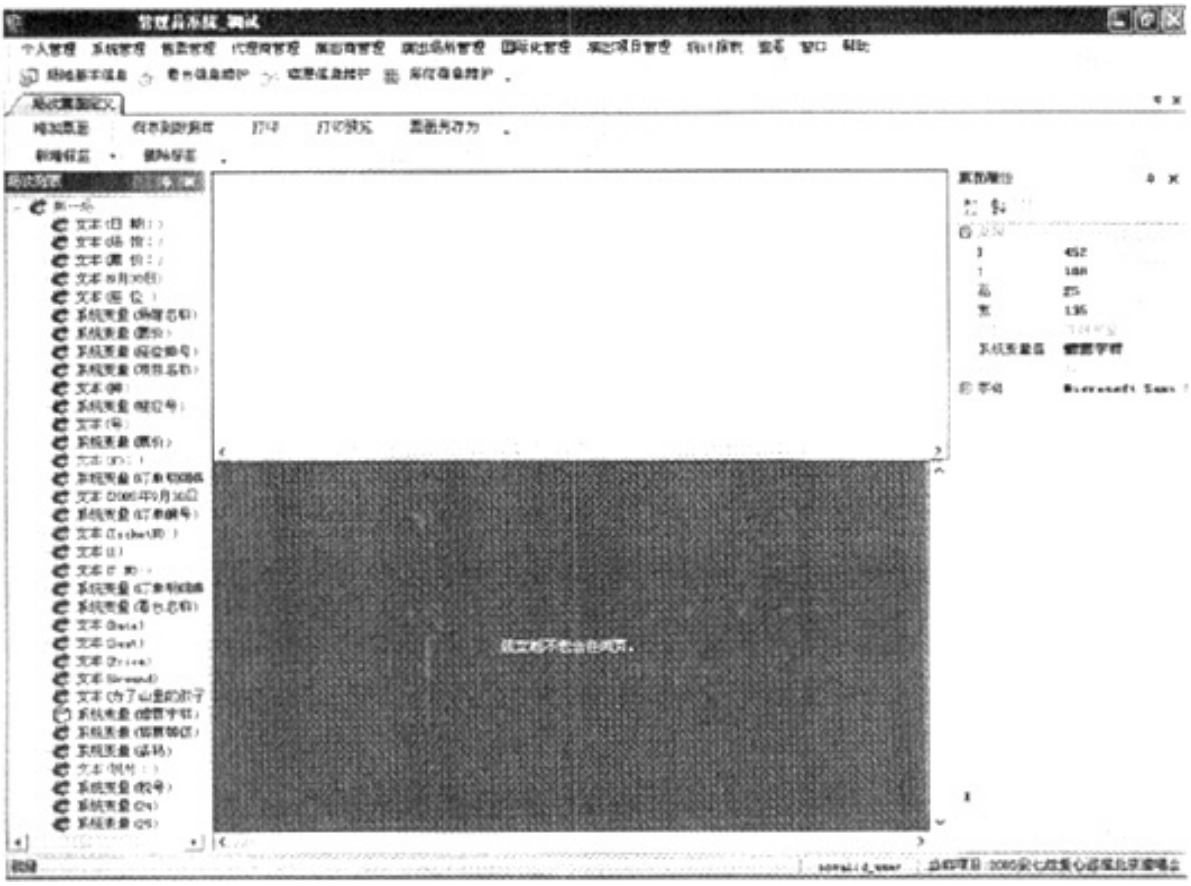

图 5.3 演出票面设置模块的窗体

Fig. 5.3 Window of perform ticket setup module

5.2 代理商售票系统的设计与实现

5.2.1 概述

代理商售票系统分为代理商客户端子系统和代理商服务器端子系统，在 2.1 小节中已经把代理商服务器端子系统作为一个关键技术进行了详细分析，在本节中主要介绍代理商客户端子系统的设计和实现。

代理商客户端子系统的所有功能模块主要分为三类：

(1) 一些模块是用于满足代理商售票相关功能的，如：可视化售票模块、售票打印机设置模块、座位预留模块、无座位票打印模块、批量打印模块、申请重打和重新打印模块、票品打印模块、收取票品费用模块等；

(2) 部分功能模块是对数据库表进行查询及显示查询结果，如：代理商订单查询模块和报表统计模块；

(3) 还有一些特殊功能的模块，如：售票员个人信息管理、自动升级等。

代理商客户端子系统的模块比较多，这里不可能对所有的模块进行详细分析。代理商客户端子系统的核心功能是实现选定票品的打印，因此下面详细介绍票品打印模块的设计与实现。

5.2.2 票品打印模块的设计与实现

在 5.1.3 小节中介绍了演出票面设置管理模块的功能是设置演出票面的信息，而票品打印模块的功能就是在票纸上打印出演出票面设置的信息。

图 5.4 是票品打印模块的实现类图。“TicketPrinter”是处理打印票品的控制类，“TicketSetting”是表示票品票面设置的类，“TicketPrinter”类与“TicketSetting”类的关系为一对多；“PrintContent”是一个抽象类，它表示票面设置上的一个元素，它有三个继承类：“ImageContent”、“SystemVariableContent”和“TextContent”，这三个类对应于 5.1.3 小节中所述的图形元素、系统变量元素和文本型元素；“PerformanceInfo”类是要打印票品所从属的演出场次信息，“SeatInfo”类表示票品所对应的座位信息（对无座位的票品而言，其对应的“SeatInfo”的对象为空）；在票品的打印过程中，票品上的所有元素必须被绘画出之后才送到打印机中，因为其中有些元素的信息是关于演出场次或座位的信息，所以“PrintContent”类必须和“PerformanceInfo”类、“SeatInfo”类存在关联关系。

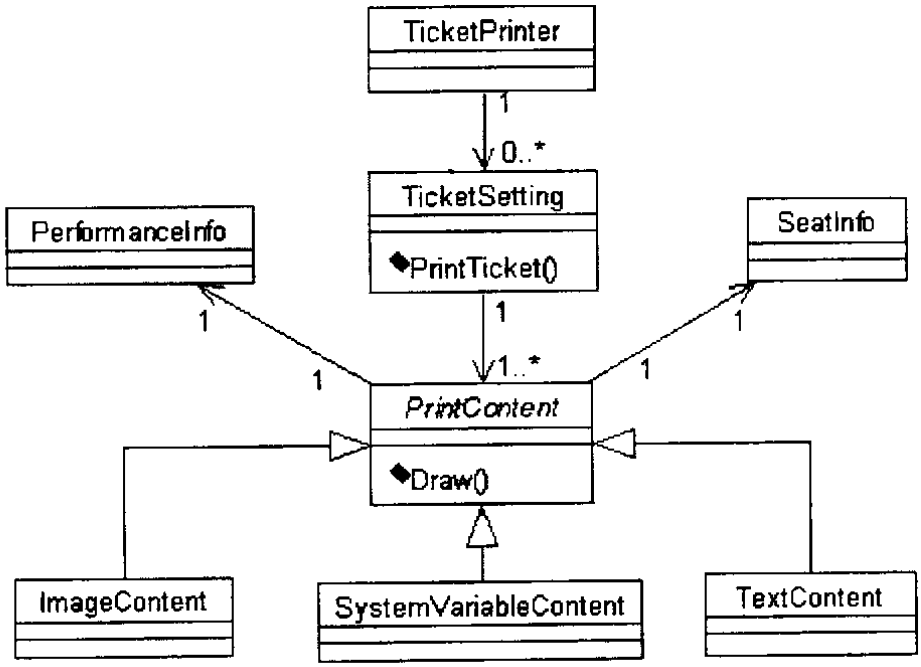

图 5.4 票品打印模块的类图

Fig. 5.4 Ticket print module class chart

图 5.5 介绍了“TicketPrinter”类的对象是如何与其它类的对象进行交互完成一个票品的打印的。

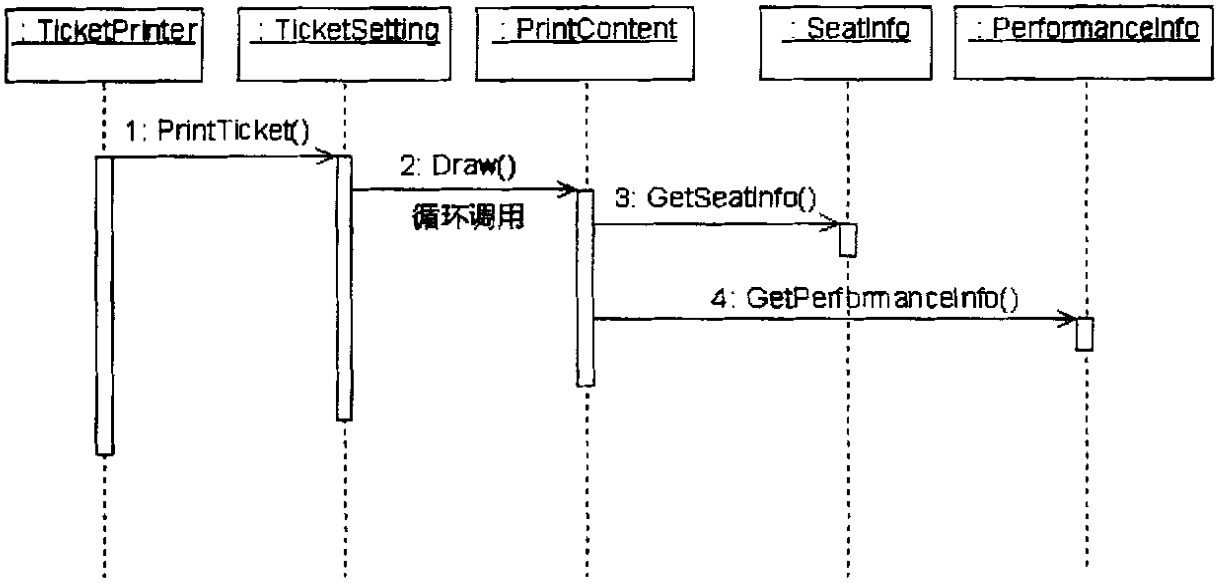

图 5.5 票品打印模块的时序图

Fig. 5.5 Ticket print module sequence chart

5.3 系统国际化版本的设计与实现

5.3.1 系统国际化的需求

目前，香港的几家公司对电子票务系统表现出极大的兴趣，他们要求尽快推出系统的中文繁体版本。为了适应形势将来的发展，电子票务系统的国际化版本将包括：中文简体、中文繁体、英文、韩文和日文版本。

本系统的国际化支持将包括以下几个方面：

(1) 当代理商售票员登录时，代理商客户端子系统能根据客户端操作系统的文化环境，运行适应当前文化环境的表现层版本；在代理商客户端子系统中提供接口，此接口用来实现各个国际化版本表现层的相互转换；客户端子系统向服务器端子系统获取数据时，服务器端子系统返回符合客户端子系统文化环境的数据信息。

(2) 在后台管理系统中，增加演出场次、剧场和座位等级等信息的多语言版本的翻译和编辑的功能模块。

5.3.2 .Net Framework 对国际化的支持

世界分为若干个文化和区域，为了支持国际化应用程序必须知道这些文化和区域的区别。文化是基于用户的语言和文化习惯的一组特性，RFC 1766 定义了文化的名称，这些名称根据语言和国家或者区域的不同在世界各地使用[4]。例如：en-AU、en-CA、en-GB 和 en-US 分别用于表示澳大利亚、加拿大、英国和美国的英语。

在.Net Framework 中包含了很多的文化和区域类，最重要的类是 CultureInfo，它表示文化，该类定义了日历、数字和日期的格式、以及和文化一起使用的排序字符串。另一个重要的类是 RegionInfo，它表示区域设置（例如货币）。一个区域可以使用多种语言，一个语言也可以在多个区域被使用。

在.Net Framework 中使用文化，必须区分 3 种类型：特定文化、中立文化和不变的文化(invariant culture)。特定的文化与真正存在的文化相关，这种文化用来 RFC 1766 定义。特定的文化可以被影射为中立的文化。例如，de（注：de 是德语的简写）是特定文化 de-AT、de-DE 和 de-CH 等的中立文化，AT、DE 和 CH 分别是澳大利亚、德国和瑞士等国家的缩写。在翻译应用程序时，通常不需要为每个区域翻译，因为澳大利亚和瑞士等国使用的德语没有多大区别。所以可以使用中立的文化来国际化应用程序，而不需要使用特定的语言。不变的文化独立于真正的文化。

图 5.6 为文化类型层次关系简单示意图：

(1) Invariant 是不变的文化，它可以理解为最高层次的文化类型；

(2) de、zh-CHS、zh-CHT 和 en 是中立的文化，它们处于文化的第二层次，zh-CHS 表示的是简体中文的中立文化，zh-CHT 表示的是繁体中文的中立文化；

(3) de-AT、de-DE、de-CH、zh-CN、zh-SG、zh-TW 和 zh-HK 等文化是特定的文化，它们处于文化的第三层次，de-AT、de-DE 和 de-CH 是 de 的特定文化，zh-CN 是 zh-CHS 的特定文化，而 zh-SG、zh-TW 和 zh-HK 是 zh-CHT 的特定文化，en 的特定文化非常多，这里就没有详细说明了。

设置文化时必须区分用户界面的文化和数字、日期格式的文化。在.Net 的编程环境中，文化与线程相关，有了这两种文化类型，就可以把这两种文化设置应用于线程。用线程设置文化时，线程类 Thread 提供了 CurrentCulture 和 CurrentUICulture 两个属性[4]。CurrentCulture 属性用于设置与格式化和排序选项一起使用的文化，而 CurrentUICulture 属性用于设置用户界面的语言文化。通过编程的方式，可以很容易的改变这两种文化的格式。

在国际化过程中，图片或字符串这样的资源可以放在资源文件或辅助程序集中。在国际化应用程序时，这些资源非常有用。.Net 对国际化资源的搜索提供了内置支持。资

源文件在软件的国际化中将起到非常重要的作用，资源文件中包含图片、字符串等资源。在下文的 5.3.4 小节中，作者会结合系统的国际化实现来说明资源文件的运用。

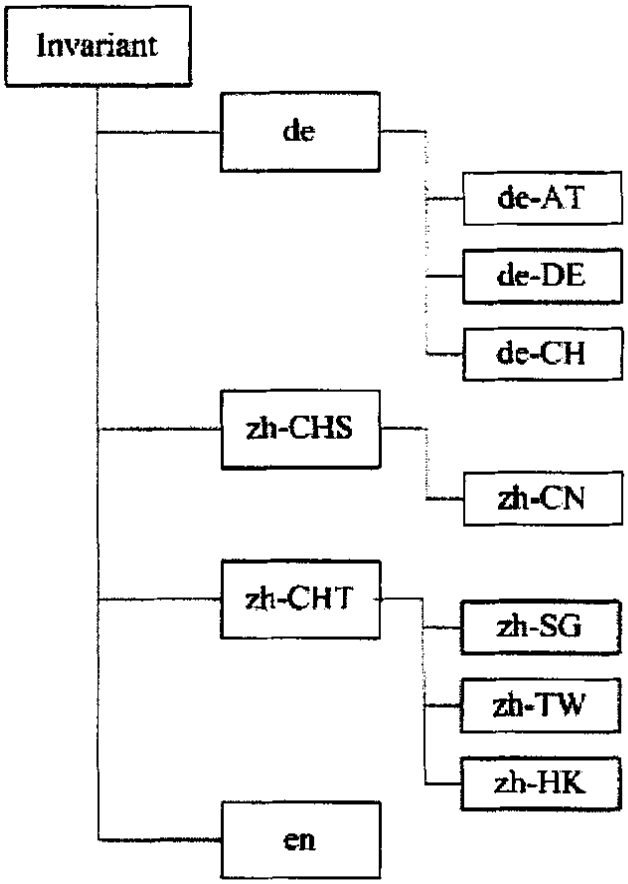

图 5.6　文化类型层次关系简单示意图

Fig. 5.6 Simpleness example of culture type level relation

5.3.3 管理后台系统的国际化设计

系统的国际化支持功能是在本系统的 1.1 升级版本中开发的，因此要满足以下几点要求：在保证实现国际化的条件下，要尽量少的影响系统的原有的功能模块；除非万不得已，尽量不要修改以前的源代码，尽量不要修改以前数据表的结构；系统的国际化功能与之前版本的兼容性好。

经过详细分析，得出结论：通过增加数据表的形式可以很好的满足上面所述的几点要求。首先，要增加一个“文化信息”表，“文化信息”表用来记录本系统所支持的文化版本种类；接着，为所有要增加国际化信息的数据表分别增加一个“资源信息”数据表，“资源信息”表中存储某信息（以简体中文为原始值）针对何种文化的对应信息形

式。下面以前文提到的表 Performance 为例来说明，可见图 5.7 表 Performance 的资源信息数据表结构示意图。

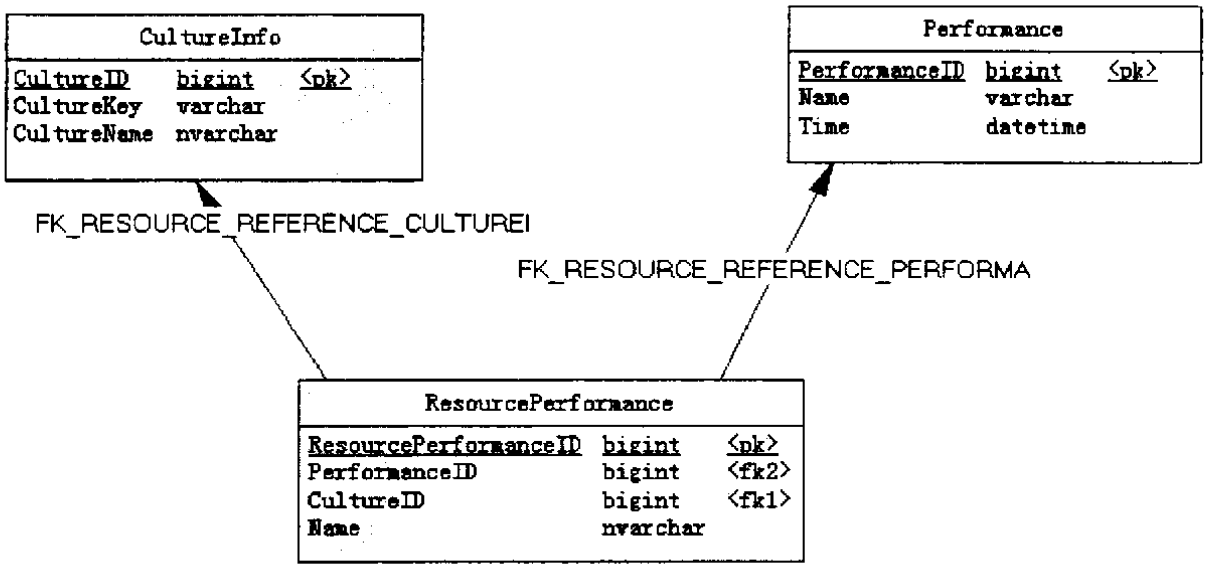

图 5.7　资源信息数据表结构示意图

Fig. 5.7 Example of resource information data form structure

在上图中，表 CultureInfo 即为“文化信息”表，表 ResourcePerformance 即为表 Performance 的相对应资源信息数据表。在表 Performance 中有个字段 Name，此字段表示演出场次的名称，因为此字段的原始信息为中文，所以要对此字段进行国际化版本信息编辑。表 ResourcePerformance 和表 CultureInfo、表 ResourcePerformance 和表 Performance 之间为一对多的关系。表 ResourcePerformance 中的字段 Name 即为资源信息，此字段对应于表 Performance 的字段 Name。

在下面的表 5.1 中为表 CultureInfo 的所有文化版本信息，字段 CultureKey 为语言的中立文化，字段 CultureName 为文化的中文名称。关于表 CultureInfo 的字段 CultureKey 中为何存放中立文化将在 5.3.4 小节中介绍。

在后台管理系统中要提供一个模块用来编辑和维护所有类似于表 ResourcePerformance 中字段 Name 的资源信息。

5.3.4 代理商售票系统的国际化设计

(1) 代理商客户端子系统的国际化支持的设计

① 当代理商售票员成功登录客户端子系统后，客户端子系统首先获得客户机的操作系统的文化环境（此文化信息是语言的特定文化，即：客户端子系统线程的

CurrentCulture 信息），之后设置客户端子系统的 CurrentUICulture 信息与 CurrentCulture 信息一致，然后客户端子系统运行适应当前文化环境的表现层版本。

表 5.1 数据表 CultureInfo 的信息

Tab. 5.1 Data form CultureInfo information

CultureID	CultureKey	CultureName
1	zh-CHS	中文简体
2	zh-CHT	中文繁体
3	En	英语
4	Ko	日文
5	Ja	韩文

在.Net 中的桌面应用程序中，每一个表现层中的 windows 窗体都有一个资源文件，资源文件的名称为“窗体名.resx”，它包括了本窗体的所有资源信息，这些资源信息仅仅包括哪些可视化控件的“标题”、“显示内容”、“大小”和“字体”等，它们都是静态资源，关于动态资源的国际化需要别的方法来解决。为了支持国际化版本，需要为每一个 windows 窗体建立其他文化的资源文件，如：英文版本的资源文件为“窗体名.en.resx”，繁体版本的资源文件为“窗体名.zh-CHT.resx”，日文版本的资源文件为“窗体名.ja.resx”。在表现层的项目被编译之后，除了生成项目的程序集（此程序集包括了本地化的资源信息）之外，会为每种其它语言创建一个辅助程序集（此程序集中仅包含此文化的特定资源信息），并把辅助程序集放在对应的语言子目录下，如：辅助程序集“项目名.en.dll”放在子目录“en”下，辅助程序集“项目名. zh-CHT.dll”放在子目录“项目名. zh-CHT.dll”下。假定一个操作系统的 CurrentCulture 为“zh-HK”的客户端子系统启动之后，客户端子系统开始加载主窗体的资源信息时，首先它会寻找名称为“项目名. zh-HK.dll”辅助程序集，一旦它发现没有这个辅助程序集之后，就会寻找父程序集，在这里即为辅助程序集“项目名.zh-CHT.dll”，之后它就会在辅助程序集“项目名. zh-CHT.dll”中找到需要的资源信息。若在辅助程序集“项目名.zh-CHT.dll”人没有找到需要的资源信息时，商客户端子系统会继续查询“项目名.zh-CHT.dll”的父程序集，这里即为“项目名.dll”，在任何情况下在程序集“项目名.dll”中都会有本项目中所有的资源信息。

对于那些在源代码中硬编码的动态资源信息（如：MessageBox 中显示的信息），要通过下面的方法实现国际化。为客户端子系统的表现层项目建立多个动态资源信息的资

源文件，如："项目名.string.resx"为本地化的资源文件，"项目名.string.en.resx"为英文的资源文件，"项目名.string.zh-CHT.resx"为本地化的资源文件等，对于要国际化的信息在上述所有资源文件中都要有对应的资源信息。再通过.Net 提供的资源管理类：ResourceManager 来动态获得满足客户机的特定文化的资源信息。

② 在代理商客户端子系统中提供所有的国际化版本种类的菜单项，通过点击菜单项用来实现各个国际化版本表现层的相互转换。当一种语言文化被选择之后，通过编成方式修改客户端子系统线程的 CurrentCulture 和 CurrentUICulture 为被选择的语言文化。这样通过 CurrentUICulture 信息，不管是静态资源信息还是动态资源信息都能在窗体或 MessageBox 中正常显示了。

(2) 代理商服务器端子系统的国际化支持的设计

客户端子系统向服务器端子系统发送远程请求以获取数据时，客户端子系统会把线程的 CurrentCulture 信息也传送过去。服务器端子系统会把 CurrentCulture 信息存放到请求客户端的线程上下文中。CurrentCulture 为语言的特定文化，而数据库中的表 CultureInfo 中存放的是中立文化，为此要把发送过来的特定文化转化为其对应的中立文化，之后服务器端子系统就查询数据库返回中立文化的资源信息。

这样设计的好处有：

① 世界上的特定文化种类很多，不可能为所有的特定文化都进行国际化版本设计。

② 属于同一个中立文化的所有特定文化的语言之间相差不大，因此可以用中立文化代替属于它的特定文化，如：用 zh-CHT 代替 zh-SG、zh-TW 和 zh-HK 等文化。这样一来，若有两个操作系统文化分别为 zh-TW 和 zh-HK 的客户端发送相同查询条件到服务器端以获取数据库的信息时，它们得到的国际化结果信息（繁体中文形式）将是完全一样的。

比如：一个 CurrentCulture 信息为"zh-HK"的客户端向服务器端发送请求，要求获得所有演出场次的名称信息，那么服务器端向数据库服务器提交的 SQL 指令为：

```
Select ResourcePerformance.Name from Performance left join ResourcePerformance on Performance. Performance ID = ResourcePerformance. Performance left join CultureInfo on ResourcePerformance. CultureID = CultureInfo. CultureID where CultureKey = 'zh-CHT'
```

5.4 用水晶报表实现系统的统计报表

系统要能提供实时销售报表与详细数据，可为演出主办方、后台管理员及代理商提供演出项目各个阶段的详细数据和销售情况报表，又要支持统计报表的 Excel，PDF 文

件格式的导出功能和直接打印功能。经过查阅资料和比较当前流行的几个表报工具(如：Crystal Reports 和 Active Report)，项目小组最终选取 Crystal Reports（即：水晶报表）作为报表统计工具。水晶报表的.Net 版本是一个与 VS.Net 完美集成的报表设计工具，它允许开发人员从数据库或其他数据源检索并且格式化结果集的报表。水晶报表还可以用自己的公式语言来创建算式和其他许多功能，这些功能通过图形、表格、汇总等方法把原始数据转转化为清晰的报表。

Crystal Reports .Net 通过数据库驱动程序与数据库连接。每个驱动程序都被编写为可处理特定数据库类型或数据库访问技术。Crystal Reports .Net 可以通过“Pull”模型和 “Push”模型来实现与数据库的连接[21]。

在“Pull” 模型（即拉模型）中，驱动程序将连接到数据库并根据需要将数据“拉”进报表中来。使用这种模型时，与数据库的连接和为了获取数据而执行的 SQL 命令都同时由 Crystal Reports .Net 本身处理，不需要开发人员编写代码。相反，在“Push”模型（即推模型）中，需要开发人员编写代码以连接到数据库，执行 SQL 命令以创建与报表中的字段匹配的记录集或数据集，并且将该对象传递给报表。该方法使开发人员在应用程序中共享数据库连接，并可以在 Crystal Reports .Net 接收到数据之前将数据进行筛选。推模型在三层架构中可以得到很好运用，它完全符合分层的体系结构的设计思想。我们可以这么考虑，报表只是表现层的一种表现形式，它不应该直接和数据库进行连接。

在本系统的报表设计中，使用了推模型。为了实现可视化的设计报表，必须引入一个新的概念：强类型数据集（即强类型 DataSet），强类型数据集是对普通数据集的继承和扩展。我们知道，普通数据集可以存放多个数据库表的信息，因此它被称为缓存数据库。当检索普通数据集的某个表的某列的值时，得到的值的类型都是 object，为了得到此列得真正类型，必须进行强制类型转化，一旦类型转化失误会引起运行时错误。而强类型数据集则解决了这个问题，因为它提供了一些作为“包装器”的属性和方法，把基于 object 的存取器包装起来，这样，我们就可以通过简单的属性来访问某个特定的值了[8]。有了强类型的数据集之后，我们就可以通过简单的“拖拽”强类型数据集的某个表的某个特定字段实现数据绑定了。依照业务要求，完成对报表的数据绑定之后，报表的设计基本上就算完成了。同时，因为强类型数据集中各个表之间可能存在各种关系，为了使报表能显示出完整的数据，我们要加上分组条件（可以这么看，一个分组条件就是两个数据表之间的外连接）。水晶报表提供了强大的汇总、自定义公式和各类图表的功能，在本系统中这些都得到了很好的运用。

下面以一个简单的例子来说明怎样用“推模型”实现水晶报表的数据绑定，可见图5.8 所示的顺序图：

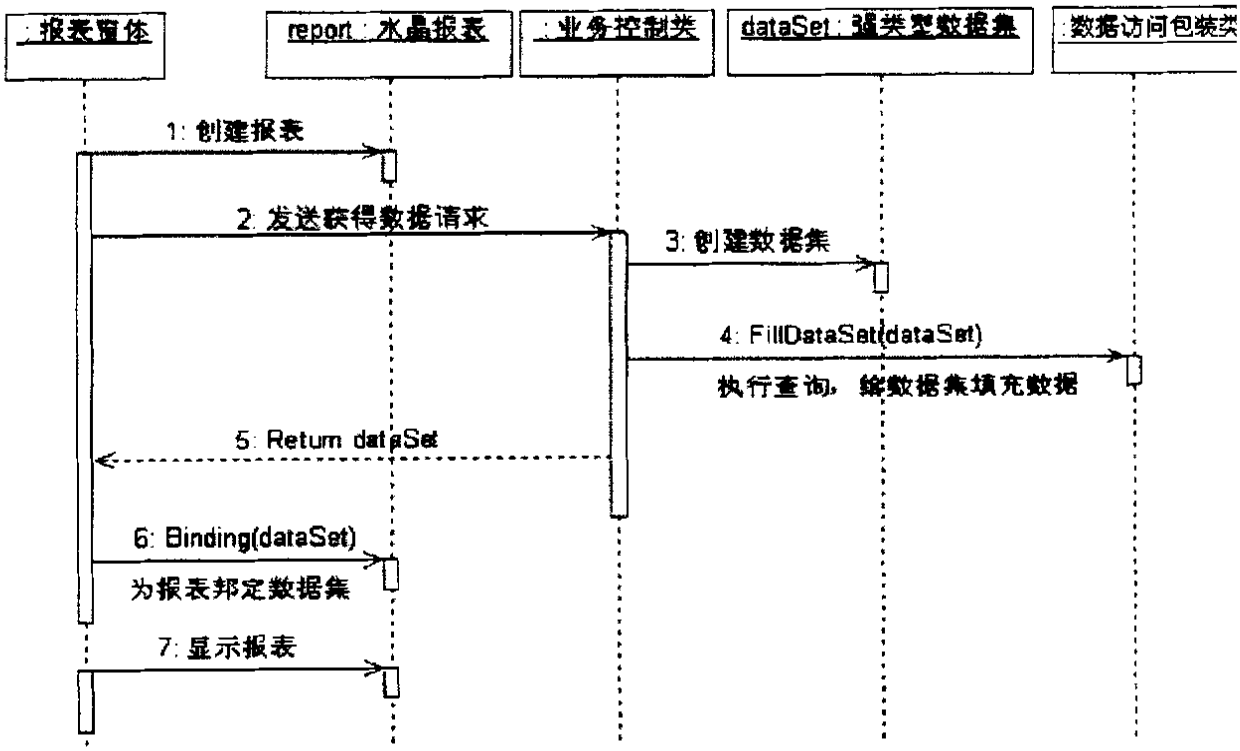

图 5.8　推模型实现水晶报表数据绑定的顺序图

Fig. 5.8 Sequence chart of illation model realizing crystal report forms data binding

在图 5.8 中，“报表窗体”是用来显示水晶报表对象的窗体容器，首先“报表窗体”向“业务控制类”对象发送获得数据的请求，“业务控制类”对象通过调用“DAO”对象的方法得到数据并把数据填充到 dataSet 对象中，之后“报表窗体”得到 dataSet 对象，最终“报表窗体”把 dataSet 对象绑定到水晶报表对象上并且显示水晶报表。

通过系统的报表统计功能，代理商、演出主办方和管理员等角色可以快速了解演出的售票汇总和明细情况。在本系统中，不同的角色所能浏览的报表种类是不一样的。

在后台管理系统中的统计报表有：代理商售票统计的汇总和明细报表、演出项目销售统计的汇总和明细报表、演出主办方的销售统计报表、财务结算和登记统计报表等。在代理商售票系统中的统计报表有：代理商售票统计的汇总和明细报表，售票员售票统计的汇总和明细报表。

6 系统的测试与实施

6.1 系统的测试

软件测试是软件质量保证的关键元素，并代表了规约、设计和编码的最终评审[22]。软件作为系统元素的可见性不断增加而且软件故障带来的代价太高使得人们注重于规划良好的彻底测试，软件开发组织将30%～40%的项目工作量花在测试上并不为怪。已经出现了大量的软件测试案例设计方法，这些方法为开发人员进行测试提供了系统的方法。能够采用以下两种方法对软件产品进行测试：

黑盒测试：若了解产品已被设计要完成的规定功能，则测试的进行要去证实各个功能完全可执行，同时在各个功能中寻找错误；

白盒测试：若了解产品的内部原理，则测试的进行要确保“所有齿轮吻合”，即内部操作依据规约执行，并且所有的内部组件被充分利用。

在本系统的开发过程中将主要进行单元测试和系统测试，单元测试是一种白盒测试，系统测试为一种黑盒测试。

6.1.1 单元测试

发现软件错误的情况主要有下面几种：

(1) 由编写代码的开发人员发现；
(2) 由尝试运行代码的开发人员发现；
(3) 由开发小组中的其他开发人员或测试人员发现；
(4) 作为产品大规模测试的一部分被发现；
(5) 由最终用户发现。

如果在第一种情况下发现软件错误，则修复错误比较容易，成本也很低。发现的情况越靠后，修复软件错误的成本就越高。修复一个由最终用户发现的软件错误可能要耗费 第一种情况所耗费的 100～1000 倍的成本。更不用说用户通常因为软件错误导致工作无法继续，而一直等到下一个版本才能解决问题。因此在本系统的开发过程中，由编写代码的开发人员进行单元测试。

本系统的单元测试工作主要分为两个步骤：人工静态检查和动态执行跟踪。人工静态检查是单元测试的第一步，这个阶段工作主要是保证代码算法的逻辑正确性（尽量通过人工检查发现代码的逻辑错误）、清晰性、规范性、一致性、算法高效性。第二步是通过设计测试用例，执行待测程序来跟踪比较实际结果与预期结果来发现错误。用人工

静态检查法能够有效的发现30%～70%的逻辑设计错误和编码错误，但是代码中仍会有大量的隐性错误无法通过视觉检查发现，必须通过跟踪调试法细心分析才能够捕捉到。

数据访问层是系统的核心部分，对它要进行完善的单元测试。在本系统的测试中使用 Nunit 作为单元测试工具，Nunit 是一种用于.NET 的开放源代码的测试框架，使用 Nunit 可以快速检测数据访问层中的问题。通过设计测试用例，并运行测试用例来动态执行跟踪数据访问层的逻辑[23]。因为采用 Nunit 进行单元测试非常繁琐，因此对于系统其它的部分的测试采用的是人工静态检查。每当一个功能模块完成之后，开发人员必须通过检查代码和动态执行来检查错误。

6.1.2 系统测试

软件系统测试是基于一定的计算机就硬件环境，对整个软件进行一系列的测试。系统测试应该根据软件项目系统级的有关文档（如系统设计文档、软件需求规格说明书等）来开展软件系统的测试工作，主要是检查新开发的软件系统是否满足系统设计文档、软件需求规格说明书等规定的功能和性能要求[24]。

本系统的系统测试的测试内容分要包括以下几个部分：功能性测试和性能测试。。

在本系统的测试过程中采用了 TestDirector 作为测试管理工具。TestDirector 通过在一个整体的应用系统中集成了测试管理的各个部分，包括需求管理，测试计划，测试执行以及错误跟踪等功能，TestDirector 极大地加速了测试过程。TestDirector 将测试过程流水化，从测试需求管理，到测试计划，测试日程安排，测试执行到出错后的错误跟踪。

本系统的测试过程如下：

(1) 在系统的需求分析之后，测试人员制定测试计划；

(2) 系统的概要设计结束之后，测试人员依据各个功能模块设计制定测试用例；

(3) 系统开发完成之后，测试人员开始进行测试。若某个功能出现 bug，就认为此功能测试不合格，要求开发人员改正错误，在开发人员修改 bug 之后，测试人员继续测试，直到完全测试通过；

(4) 通过 TestDirector，为系统的测试制定测试计划和测试用例，并对 bug 进行全程跟踪。

功能性测试结束之后，测试人员提交测试报告，测试报告可见表 6.1 所示：

表 6.1 功能测试报告

Tab. 6.1 Function testing report

测试项目	实际结果	遗留问题	解决计划
代理商售票员登录客户端	成功登录	无	
售票员更改个人注册信息	成功	无	
可视化选择座位完成售票	成功	无	
售票员订单查询	成功	无	
申请重打和重新打印		状态为“取消”的订单不可以申请重打	在系统的 1.1 升级版本中解决
售票打印机配置	成功	无	
座位预留管理	成功	无	
代理商报表统计（汇总和明细报表）	成功	无	
售票报表统计（汇总和明细报表）	成功	无	
代理商售票员退出	成功退出	无	
超级管理员登陆服务器	成功登录	无	
启动服务器	成功	无	
重启服务器	成功	无	
停止服务器	成功	无	
超级管理员退出服务器	成功退出		
管理员登陆后台管理系统	成功登录	无	
代理商信息管理	成功	无	
售票员信息管理	成功	无	
演出项目信息管	成功	无	

续表 6.1

表 6.1 功能测试报告

Tab. 6.1 Function testing report

测试项目	实际结果	遗留问题	解决计划
剧场信息管理	成功	无	
剧场基本座位设置		删除选定座位时的警告提示不合理	在系统的 1.1 升级版本中解决
场次座位设置		删除选定座位时的警告提示不合理	在系统的 1.1 升级版本中解决
场次票面设置	成功	无	
订单管理	成功	无	
财务结算管理	成功	无	
后台用户管理	成功	无	
场次销售报表统计	成功	无	
代理商报表统计	成功	无	
主办方报表统计	成功	无	
管理员权限管理	成功	无	
自动升级		在二级目录中的文件不能完成升级	在系统的 1.1 升级版本中解决
管理员退出后台管理系统	成功退出	无	

系统的性能测试：

模拟在售票系统最大负荷情况下，客户端请求远程调用的响应时间，经测试得出售票系统的平均响应时间在 4 秒以内，基本满足了系统的非功能需求。

性能测试方案的构建如下：开发一个可以部署在代理商客户端的测试程序，程序以 0.1 秒为时间间隔不间断循环向代理商服务器端发送 5 种远程调用请求，同时将响应时间记录到日志文件中。将该测试程序部署到 5 台客户机上，同时运行 10 分钟，该测试用例基本能够模拟 500 个代理商客户端同时访问服务器，在 10 秒内同时发出访问请求的情况。

测试结果显示最快响应时间为 0.61 秒，最慢响应时间为 4.97 秒，平均响应时间为 3.86 秒，基本满足系统的性能指标。

6.2 系统的部署

6.2.1 .Net 平台下的部署方式

编译源代码并完成测试后，开发过程并没有结束。在这个阶段，需要把应用程序提供给最终用户。无论是 ASP.NET 应用程序、桌面应用程序还是使用 Compact Framework 构建的应用程序，系统都必须部署到目标环境中去。.Net Framework 使部署工作比以前的开发平台要容易得多，因为不再需要注册 COM 组件，编写新的注册表项了[4]。

.Net 提供了两种部署应用程序的方式：xcopy 部署和 Windows Installer 安装程序部署。Xcopy 部署就是把一组文件复制到目标计算机上的一个目录中，无论文件的数目是多少，只要所有文件都被复制到同一个文件夹中，应用程序就可以运行了，不需要编辑配置或注册表。Windows Installer 是一个服务，它负责管理在大多数 Windows 操作系统上安装、更新、修复和删除应用程序，通过 Visual Studio .Net 中的部署项目可以很容易的创建 Windows 安装软件包。

6.2.2 实现系统的部署

后台管理系统、代理商客户端子系统和代理商服务器端子系统要分别进行部署。下面详细介绍部署方式：

(1) 管理后台系统和代理商客户端子系统的部署

因为 xcopy 部署方式有一些缺点：它不能把程序集放到全局程序集缓存（GAC）中，不能在操作系统的“开始”菜单中添加图标，也不能在“桌面”上添加启动系统的快捷方式。上述的缺点会使得在目标计算机上进行第一次部署非常麻烦，同时使最终用户操作系统很不方便。

因此，在目标计算机上第一次部署管理后台系统或代理商客户端子系统时，要通过 Windows Installer 安装程序进行部署。使用 Windows Installer 进行部署时，一些第三方控件组件（如 5.2 节所述的 Infragistics NetAdvantage 的组件库中的程序集）可以被直接注册到全局程序集缓存中，同时也会在“开始”菜单中添加图标等。

在管理后台系统或代理商客户端子系统有了升级版本之后，通过简单的 xcopy 部署方式把升级的组件部署到应用程序的运行目录下覆盖以前的版本就可以了，这个功能由系统的“自动更新”模块来实现。此时，不用 Windows Installer 进行升级部署是有道理的。若用 Windows Installer 进行升级部署，必须先通过 Windows Installer 卸载之

前的版本，再安转新的版本；在一次升级过程中，一般只有很少的组件被升级，因此用 Windows Installer 进行升级部署是麻烦的、不明智的。

(2) 代理商服务器端子系统的部署

因为代理商服务器端子系统直接被部署到服务器端，而且它所拥有的组件也不是很多，所以直接使用 xcopy 方式进行部署就可以了。

6.2.3 系统的部署视图

UML 的部署视图考虑应用程序的物理部署情况，如：网络布局和组件在网络上的位置的问题。部署视图包含处理器、设备和进程，也包含两个处理器、两个设备或处理器与设备之间的连接线[25]。

系统的部署视图如图 6.1 所示：代理商服务器端子系统、后台管理系统和数据库服务器被部署在一个局域网内；代理商客户端子系统被部署在外网，因为代理商客户端子系统负责售票，所以它拥有售票打印机。

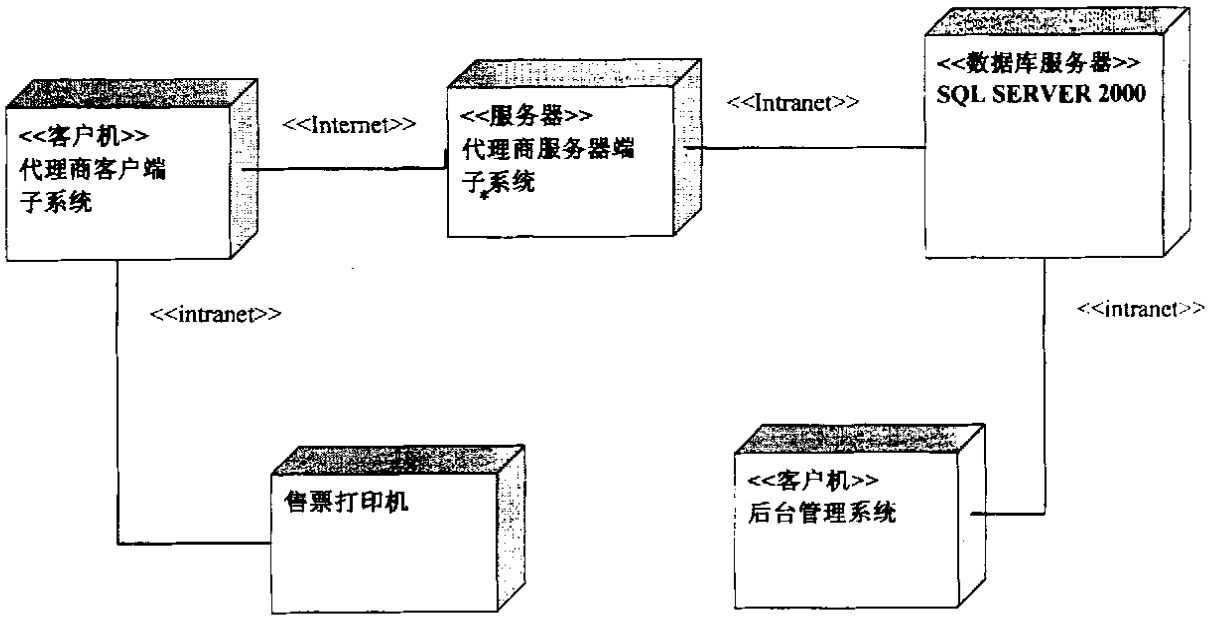

图 6.1 系统的部署视图

Fig. 6.1 System dispose view

6.2.4 系统的运行环境

因为后台管理系统、代理商客户端子系统和代理商服务器端子系统要分别部署，所以它们的运行环境可能是不同的，下面的三个表 6.2、6.3、6.4 分别描述了它们的运行环境的条件：

表 6.2 后台管理系统的运行环境

Tab. 6.2 Background management system running environment

项目名称	名称/版本
操作系统	Windows 98/2000/xp/ Server 2003
运行平台	.Net Framework 1.1
关系数据库系统	SQL Server 2000
CPU	奔腾III 300 以上
内存	128M 以上
硬盘	剩余空间 300M 以上
显示器	17 寸彩色显示器, 支持分辨率 800×600 以上

表 6.3 代理商客户端子系统的运行环境

Tab. 6.3 Agent client subsystem running environment

项目名称	名称/版本
操作系统	Windows 98/2000/xp/ Server 2003
运行平台	.Net Framework 1.1
CPU	奔腾III 300 以上
内存	128M 以上
显示器	彩色 VGA、分辨率 800×600 及以上，最好分辨率为 1024×768
硬盘	剩余空间 300M 以上
票品打印机	TEC B-419 型打印机 或 Boca 直热式高速打印机
其它设备	USB 或并口加密狗

6.3 系统的运行与维护

从 2006 年 1 月份至今，后台管理系统和代理商售票系统基本处于正常运行状态。其间，本系统参与运营了多个演出和娱乐项目，得到了演出主办方和多个代理商的一致好评。

据实践证明，分布式应用服务器被启动之后，它在运行的 3、4 月间不会发生任何异常情况（通过查看服务器端的异常日志文件和代理商客户端的请求异常日志），同时它占用的内存和 CPU 资源也没有多大增加。这些都说明分布式应用服务器的设计是比较成功的。

后台管理系统和代理商售票客户端子系统的功能一直处于完善之中，一旦发布了新版本之后，后台管理系统和代理商售票客户端子系统会启动“自动更新”模块完成系统的升级。

表 6.4 代理商服务器端子系统的运行环境

Tab. 6.4 Agent server subsystem running environment

项目名称	名称/版本
操作系统	Windows Server 2000 / Server 2003
运行平台	.Net Framework 1.1
关系数据库系统	SQL Server 2000
CPU	奔腾III 300 以上
内存	512M 以上
硬盘	剩余空间 500M 以上

结　论

在中国，电子票务正处于发展阶段，而电子票务在各个行业的发展现状也很不一致。在航空、民航业中电子票务的运用已经非常普遍，但是在文艺演出和体育比赛等行业中电子票务的发展还处于萌芽阶段。电子票务系统正是为了满足当前市场需求而开发的，它专门为文艺演出和体育比赛等票务项目管理而设计，它为票务项目的完整生命周期提供全面的解决方案。

本论文的研究工作开始于 2005 年 8 月，于 2006 年 5 月份基本结束，本论文就电子票务系统 C/S 部分的设计和实现展开分析。主要工作成果包括以下几个方面：

(1) 本论文收集了国内外有关电子票务的发展和研究现状成果，这些对电子票务系统的开发起了很大的帮助。

(2) 本论文的主要讲述了需求分析、系统的总体设计、系统详细设计和实现、测试和实施方面的工作。本论文对在电子票务系统的设计和开发中遇到的关键问题提供了比较全面的解决方案。

在系统的开发过程中，本人参与了分布式应用服务器的构建和自定义数据访问层的设计工作；独立设计和开发了系统的多个模块，如：票品打印模块的设计、软件国际化版本的设计和实现、统计报表的实现。

因为时间和资金限制，电子票务系统仍然有一些不完善的地方，以下是可以进一步展开研究工作的几个方向：

(1) 在后台管理系统中，客户机是直接连接数据库服务器的，随着客户机的增加，系统的效率将会大受影响。因此为了满足大型演出的需求，需要把后台管理系统也设计为一个分布式应用系统。

(2) 在电子票务系统中还有一个检票子系统，因为检票子系统尚未完善，所以在本论文没有对它介绍，完善检票子系统也将是进一步的工作之一。

参考文献

[1] Robinson Simon, Nagel Christian. 李敏波. C#高级编程（第三版）. 北京：清华大学出版社，2005.

[2] 王正桓;蔡明. MS.NET Remoting的分布式技术应用研究.计算机应用与软件.2005,3:21-24.

[3] McLean Scott, Naftel James, Williams Kim. 张坤琪. Microsoft .Net Remoting 权威指南. 北京：机械工业出版社，2003.

[4] Srivatsa, V. Remoting in.NET Framework 2.0. C C PLUS PLUS USERS JOURNAL. 2005, 23(11): 29.

[5] Sharma, N.; Sharma, D. A Multi-agent Framework for.NET. LECTURE NOTES IN COMPUTER SCIENCE. 2005, 3681: 219.

[6] Seidmann, T. Architecture of a Business Framework for the.NET Platform and Open Source Environments. LECTURE NOTES IN COMPUTER SCIENCE. 2005, 3381: 47.

[7] Metsker Steven John. 颜炯. C#设计模式. 北京：中国电力出版社，2005.

[8] Otey, Michael. ADO.NET 2.0. SQL Server Magazine. 2005, 7(2): 48.

[9] Richter Jeffrey. 李建忠. Applied Microsoft .NET Framework Programming. 北京：清华大学出版社，2003.

[10] Ardestani K., Hoffman K., Xie Donald. 张哲峰. 高效掌握ADO.Net. 北京：清华大学出版社，2003.

[11] Chakravorti, Bhaskar. Bulk-Insert Options for ADO.NET. SQL Server Magazine. 2004, 6(8): 48.

[12] Ma, T.; Luo, J. Optimizing Large File Transfer on Data Grid. LECTURE NOTES IN COMPUTER SCIENCE. 2005, 3795: 455.

[13] Fowler Martin. 王怀民，周斌. 企业应用架构模式. 北京：机械工业出版社，2004.

[14] 冯冲，江贺，冯静芳. 软件体系结构理论与实践. 北京：人民邮电出版社，2004.

[15] Larman Craig. 姚淑珍，李虎. UML 和模式应用——面向对象分析与设计导论. 北京：机械工业出版社，2002.

[16] Chawathe, Y.; Ramabhadran, S.; Ratnasamy, S.; LaMarca, A.; Shenker, S.; Stoica, I.; Hellerstein, J. A Case Study in Building Layered DHT Applications. COMPUTER COMMUNICATION REVIEW. 2005, 35(4): 97.

[17] Pressman Roger s.. 梅宏. 软件工程：实践者的研究方法. 北京：机械工业出版社，2002年.

[18] Faison Ted. 战晓苏. Visual C#基于组件的开发. 北京：清华大学出版社，2003.

[19] Crocker Angela, Olsen Andy, Jezierski Edward. Design data layer module and transfer data. LECTURE NOTES IN COMPUTER SCIENCE. 2005, 3517: 356.

[20] Garcia, Andrew. Stanching GDI+ overflow. eWeek. 2004, 21(42): 59.

[21] McAmis David. 李万红，王军. Crystal Reports for Visual Studio .Net 高级编程. 北京：清华大学出版社，2003.

[22] Bruegge.B, Dutoit.A.H. 吴丹，唐忆，申震杰. 面向对象的软件工程——构建复杂且多变的系统. 北京：清华大学出版社，2002.

[23] Smith Steven. Use NUit help testing: Cell testing for data access. LECTURE NOTES IN COMPUTER SCIENCE. 2004, 3749: 427.

[24] 林宁，孟庆余. 软件测试实用指南. 北京：清华大学出版社，2004.

[25] Boggs Wendy, Boggs Michael. 邱仲潘. UML 与 Rational Rose2002 从入门到精通. 北京：电子工业出版社，2002.

致　　谢

本人在该论文撰写过程中，得到了我的导师的悉心指导和帮助。导师渊博的知识，严谨的治学态度以及针对科学研究的孜孜探求给了我很大的启示，让我在以后的工作中受益匪浅。在此，谨向导师表示衷心的感谢和崇高的敬意！

同时，在公司工作期间，很多同事们给予了我很多帮助和支持，论文的成功与他们对我的帮助是分不开的。

最后，感谢所有曾经给予我关心和帮助的人！