TPC-C 基准测试系统的设计与实现

学　　院　　软件学院

专　　业　　软件工程

年　　级　　2004 级

姓　　名　　康　　强

指导教师　　张坤龙

2008 年 6 月 15 日

摘　要

随着数据库管理系统的高速发展，其技术日趋成熟，产业规模也已十分庞大。而在国内，也对数据库的发展给予了高度的重视。在这种情况下，无论是开发数据库，还是选择采购数据库，都需要有一个参考来衡量各个 DBMS 的性能。所以，对 DBMS 的系统性能测试是十分必要的，同时也具有重大的意义。

本论文选择了由 TPC 组织提出一个针对 OLTP 应用环境的基准：TPC-C 基准。通过对 TPC-C 测试模型的详细分析，以及对其 TPC-C 规范内容的深入了解，得出 TPC-C 基准测试的基本方法和流程。在此基础上对 TPC-C 基准测试系统 TpccTesting 进行设计，重点阐述了基本 TPC-C 模型的设计思路、实现方法以及 TpccTesting 测试程序的体系结构。然后使用 C#语言实现了基于 TPC-C 测试模型的测试工具 TpccTesting。在整个过程中，主要工作如下：

1. 介绍 DBMS 的相关内容，然后阐述了系统性能测试的内容和方法，从而引出论文的选题背景和意义。
2. 重点的学习了 TPC-C 基准性能测试的规范，对 TPC-C 的测试模型进行了详细的分析，从中得出 TPC-C 测试的基本方法和流程，并依此进行设计。
3. 对系统进行设计实现时，详细说明事务的一些细节，重点解决了 3 个 TPC-C 模型的关键点：数据库连接池管理，维护连接资源；OLTP 应用环境的模拟；多用户并发访问的模拟。

关键字：TPC-C；C#；OLTP；TpccTesting

ABSTRACT

With the rapid development of the database management system, its technology become more mature, and the industrial scale has been huge. In China, the development of the database also gave high attention. In this case, both the development of the database and choose the procurement database need to have a reference to measure the performance of various DBMS. Therefore, the DBMS system performance testing is necessary, at the same time also of great significance.

This paper selected TPC-C benchmark which supported by the TPC organizations OLTP application for an environmental baseline. Through the detailed analysis for the TPC-C test model of TPC-C and its norms in-depth understanding of the content, that TPC-C benchmark tests the basic methods and processes. Base on these, the TPC-C benchmark TpccTesting system is designed, focusing on the basic TPC-C model of the design, implementation method and test procedures TpccTesting architecture. Then use the C # language based on the TPC-C test model of the test tools TpccTesting. Throughout the process, the main work is as follows:

1. Introduce the content of the DBMS, and then we show content and methods for the performance testing of the system. Among these topics we leads to the background and significance.
2. Focusing on the study of the TPC-C benchmark norm. the TPC-C test model carried out a detailed analysis, drawn from the TPC-C test the basic methods and processes, and so on the design
3. For the system design and implementation, detailed descript the details of the services, focused on solving the three key points of the TPC-C model: the database connection pool management, maintenance of connecting resources; OLTP application environment simulation, multi-user access with the simulated

Key words: TPC-C；C#；OLTP；TpccTesting

目　　录

第一章　　绪论

1.1　数据库管理系统

数据库管理系统(database management system)是一种操纵和管理数据库的大型软件，是用于建立、使用和维护数据库，简称 DBMS。它对数据库进行统一的管理和控制，以保证数据库的安全性和完整性。用户通过 DBMS 访问数据库中的数据，数据库管理员也通过 DBMS 进行数据库的维护工作。它提供多种功能，可使多个应用程序和用户用不同的方法在同时或不同时刻去建立，修改和询问数据库。它使用户能方便地定义和操纵数据，维护数据的安全性和完整性，以及进行多用户下的并发控制和恢复数据库。

按功能划分，数据库管理系统大致可分为 6 个部分：

(1)模式翻译：提供数据定义语言(DDL)。用它书写的数据库模式被翻译为内部表示。数据库的逻辑结构、完整性约束和物理储存结构保存在内部的数据字典中。数据库的各种数据操作(如查找、修改、插入和删除等)和数据库的维护管理都是以数据库模式为依据的。

(2)应用程序的编译：把包含着访问数据库语句的应用程序，编译成在 DBMS 支持下可运行的目标程序。

(3)交互式查询：提供易使用的交互式查询语言，如 SQL。DBMS 负责执行查询命令，并将查询结果显示在屏幕上。

(4)数据的组织与存取：提供数据在外围储存设备上的物理组织与存取方法。

(5)事务运行管理：提供事务运行管理及运行日志，事务运行的安全性监控和数据完整性检查，事务的并发控制及系统恢复等功能。

(6)数据库的维护：为数据库管理员提供软件支持，包括数据安全控制、完整性保障、数据库备份、数据库重组以及性能监控等维护工具。

基于关系模型的数据库管理系统已日臻完善，并已作为商品化软件广泛应用于各行各业。它在各户服务器结构的分布式多用户环境中的应用，使数据库系统的应用进一步扩展。随着新型数据模型及数据管理的实现技术的推进，可以预期 DBMS 软件的性能还将更新和完善，应用领域也将进一步地拓宽。

数据库管理系统是随数据库系统发展而发展的。自数据管理进入数据库管理系统后，上世纪六七十年代，先后发展了产生了层次数据库系统、网状数据库系统和关系数据库系统。这三个系统都是建立在相应的数据库模型理论上的，数据模型是一种限制世界数据特性的抽象，分别在现实世界、信息世界和机器世界表达描述，分别以数据、信息或记录等表示的。数据模型主要是数据结构、

数据操作和数据约束等。随着面向对象概念的产生以及发展。进而出现了面向对象数据库。目前市场技术上占主流的数据库系统是关系数据库，其产品种类多、门类齐、参与开发的公司多，并且应用十分广泛。

1.1.1 主流数据库管理系统发展现状

数据库管理系统经历了 30 多年的发展演变，已经取得了辉煌的成就，发展成了一门内容丰富的学科，形成了总量达数百亿美元的一个软件产业，并且已经发展成为一个规模巨大、增长迅速的市场。 目前，市场上具有代表性的数据库产品包括 Oracle 公司的 Oracle、IBM 公司的 DB2 以及微软的 SQL Server 等。在一定意义上，这些产品的特征反映了当前数据库产业界的最高水平和发展趋势。因此，分析这些主流产品的发展现状，是我们了解数据库技术发展的一个重要方面：关系数据库技术仍然是主流、产品形成系列化、支持各种互联网应用、向智能化集成化方向扩展

1.1.2 我国数据库管理系统发展现状

国外数据库发展较早，并且经历了十几年的技术拓展和实践应用，已经变的十分成熟，其生产的数据库也作为主流数据库产品，代表了现在数据库产业的发展。相对而言，国内数据库起步晚，市场拓展也刚刚起步。

随着国内信息化产业的加速发展，数据量越来越大，使得我国开始重视数据库管理系统的发展，并将其作为国家信息基础建设设施的重要组成部分，是民族 IT 产业以及软件产业发展的支撑技术。

多年来我国一直注重开发自主知识产权的数据库产品，并取得了长足的进步，已经具备普遍应用所需的基本数据库管理功能，也具有一定的扩展性，如 DM4 和 Kingbase ES 等产品，已得到了很好应用。但相比国外的主流数据库仍有一定的差距：易用性、可管理性、系统的稳定性、执行效率等方面还有很大不足。并且，国内数据库起步晚，其技术领域涉及面比较窄，从中国的数据库市场来看，大部分数据库系统的建立是用来进行传统的 OLTP 业务。也有一些企业建立了数据仓库系统，但真正发挥效用的却不多见。和 TCP/IP, SMTP, Java 等相比，尚不存在可靠的、完善的、被广泛接受的数据仓库标准，影响了数据仓库项目的实施。

针对数据库的发展现状，国内对数据库产业的重视，这就在需要对数据库管理系统进行相应的性能测试，以次来为我们发展数据库、选择数据库和采购数据库提供参考。

1.2 系统性能测试

1.2.1 性能测试的概念

由于在发展数据库产业的过程中，以及之后对其进行选择的时候，我们需要衡量其性能，为此进行软件系统的性能测试。

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

性能测试在软件的质量保证中起着重要的作用，它包括的测试内容丰富多样。中国软件评测中心将性能测试概括为三个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试：

1. 应用在客户端性能测试的目的是考察客户端应用的性能，测试的入口是客户端。它主要包括并发性能测试、疲劳强度测试、大数据量测试和速度测试等。其中并发性能测试是重点，过程是一个负载测试和压力测试的过程，即逐渐增加负载，直到系统的瓶颈或者不能接收的性能点，通过综合分析交易执行指标和资源监控指标来确定系统并发性能的过程。主要目的是评价系统的当前性能、预测系统的未来性能和确认性能瓶颈并优化和调整应用。

2. 应用在网络上性能的测试重点是利用成熟先进的自动化技术进行网络应用性能监控、网络应用性能分析和网络预测。网络应用性能分析的目的是准确展示网络带宽、延迟、负载和 TCP 端口的变化是如何影响用户的响应时间的。网络应用性能监控的目的是了解系统在网络上的运行情况、访问量大小和系统性能等。网络预测则是考虑到系统未来发展的扩展性，预测网络流量的变化、网络结构的变化对用户系统的影响。

3. 应用在服务器上性能的测试目的则是实现服务器设备、服务器操作系统、数据库系统、应用在服务器上性能的全面监控。

而进行性能测试的目的是验证软件系统是否能够达到用户提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件，最后起到优化系统的目的。其目的包括：评估系统能力、识别体系中的弱点、系统调优以及验证稳定性和可靠性。对于性能测试的方法，取决于想要达到的结果。一般包括：基准测试、性能规划测试、渗入测试和峰谷测试等。在这里面基准测试较为常见，其最好的方法是：每次测试改变一个且只改变一个参数，然后通过测试在一个

相对短时间内获得一致的、可再现的结果。可再现的结果有两个好处：减少重新运行测试的次数；对测试的产品和产生的数字更为确信。这样在分析测试结果时我们就有迹可循，同时也降低成本，提高了可靠度。

1.2.2 数据库系统测试体系

由于现在数据库管理系统的广泛应用和高速发展，市场出现了很多数据库产品，并且随着这些产品的不断成熟，不仅是厂商在开发升级数据库管理系统十，还是我们在选择适用的数据库管理系统时，都需要系统性能测试的结果作为依据。

从 2002 年开始，中国软件评测中心与清华大学软件学院共同承担了国家 863 计划“数据库管理系统测试及其工具研发”课题，深入地研究了国内外软件测试技术的最新前沿，针对我国数据库管理系统的发展现状进行了深入的调查研究，恰当地选择相关的国际标准，通过详细地各项研究，形成了一套相对完整的数据库管理系统评测体系。该体系参照了数据库相关国际标准和有关软件系统测试规范，配合国产数据库管理系统的开发进程，将数据库管理系统测试分成四个方面：产品确认测试、标准符合性测试、基准性能测试和应用综合测试。具体包括：

- 产品确认测试。
- 标准符合性测试。三个标准：SQL 标准符合性测试、ODBC 标准符合性测试和 JDBC 标准符合性测试。
- 基准测试。2 个标准：TPC-C 测试和 TPC-W 测试。

1.3 论文的组织结构

在国家软件评测中心提出的数据库管理系统测试的四个方面中，选择了 TPC-C 基准作为国家对 DBMS 测试的重要基准之一。而根据我国数据库系统主要是用来进行传统的 OLTP 业务的状况，选择专门为 OLTP 应用环境提出的 TPC-C 基准进行分析，并设计实现 TPC-C 测试系统，这是十分必要的。对以后了解数据库的发展趋势以及国内数据库的发展都有重大意义。

本文主要内容是对设计和实现一个 TPC-C 基准测试系统 TpccTesting。为此，我们将完成以下工作：

第 2 章对 TPC-C 基准进行阐述和分析。

第 3 章基于 TPC-C 基准的分析，对 TpccTesting 进行设计。

第 4 章则是介绍在实现过程中的细节和 3 个解决 TPC-C 模型的关键：数据库连接池管理，多用户并发访问的模拟，OLTP 应用环境的模拟。

第二章　TPC-C 基准测试规范

2.1　TPC 及其制定的性能测试规范

就数据库测试而言，国际上 TPC-组织提出的性能测试标准和规范是大家最为熟悉的数据库测试规范。TPC 是事务处理委员会(Transaction Processing Council)的缩写，该组织最早成立于 1988 年，是由一些在计算机领域提供软硬件系统或者相关解决方案的厂家组成，总部设在美国。该组织对全世界开放，但迄今为止，决大多数会员都是美、日、西欧的大公司，比如：IBM、NCR、HP、Oracle、Microsoft 等。它的职责是制定商务应用基准程序(Benchmark)的标准规范、性能和价格度量，并依据这些基准测试项目发布客观性能数据。

TPC 是一家非盈利公司，主要致力于定义事务处理和数据库基准测试，并为行业提供客观、可验证的 TPC 性能数据。“事务处理”一词经常用于描述很多业务和计算机功能[11]。从计算机功能的角度看，事务处理指的是一系列操作或运行，包括磁盘读/写、操作系统调用或某种形式的从一个子系统到另一个子系统的数据传输。其目的则主要是为了针对特定的领域，如联机交易处理系统(On Line Transaction Processing, OLTP)、数据仓库或决策支持系统(Decision Support System, DSS)、电子商务解决方案等，制定相应的性能测试规范，从而为用户在选择相应解决方案的平台时提供参考标准。

TPC 系列基准测试可用于评估多种应用的性能，例如借记/贷记交易、部件批发供应商应用、销售趋势和金融分析（特殊业务分析）。[6]TPC 按照商业世界的公共理解将事务处理定义为商品、服务或资金的商业交换。根据 TPC 的定义，一个典型的事务处理将包括对数据库进行更新，以支持存货控制（商品）、航班预订（服务）或银行（资金）这样的功能。

在这些环境中，客户或服务代表通过连接到一个数据库的终端或桌面计算机输入和管理自己的事务处理。在典型情况下，TPC 基准测试的内容包括对事务处理（TP）和数据库性能进行评估，计算在每一时间单位中某一给定系统或数据库能够完成多少次事务处理（例如每秒钟完成的事务处理数量或每分钟完成的事务处理数量）。

TPC 组织制定的数据库测试规范包括 TPC-A、TPC-B、TPC-C、TPC-D、TPC-E、TPC-H/TPC-R 和 TPC-W 等，其中主要应用的是 TPC-C、TPC-H/TPC-R 和 TPC-W，而 TPC-W 已停止使用,而 TPC-C 也被 TPC-E 所代替，相关比较如表 2-1：

表 2-1　TPC-C、TPC-H 和 TPC-W 比较

	TPC-C	TPC-H	TPC-W
类型	联机事务处理 OLTP	联机分析处理 OLAP	电子商务应用
模拟环境	批发商销售货物	零售商市场分析	网上订购书籍
事务类型	三个更新事务：订购、付款、发货。两个查询事务：订单查询，库存查询。	二十二个复杂查询事务和两个更新事务。	十三种示同类型的网络交互事务，包括订单输入、查询等。
测试方法	五种事务并发执行的情况下，测试出所产生的新订单的数量。	查询与更新事务并发时处理的复杂查询数量。	多种事务并发时测试 web 交互次数。
量度单位	TpmC：每分钟发生的订购事务。 $/tpmC：每个订购的代价。	QphH/R@Size：当数据库大小为 Size 时每小时处理的查询数。 $/QphH/R@Size：每个查询的代价。	WIPS：每秒钟产生的 web 交互次数。 $/WIPs：每个 Web 交互的代价。

TPC 并不给出基准程序的代码，而只给出基准程序的标准规范,测试者需要严格根据标准其规范的内容来完成测试工具代码的编写。

2.2　TPC-C 性能测试基准

本论文的重点是研究 TPC-C 性能测试基准的测试模型，分析测试规范，然后完成对数据库的测试系统的设计与实现。本节将阐述 TPC-C 基准的主要内容。

2.2.1　规范概要

TPC-C 测试规范是在 1992 年 7 月发布的，它是专门针对联机交易处理系统(OLTP)工作量的一个衡量标准，OLTP 是一种读操作和更新事务操作剧烈交互执行的处理，也被称为业务处理系统。TPC-C 模拟了被复杂的联机事务处理应用环境创建的活动，它是通过把很多的系统组成部件和特定环境相关联来实现的，这种特定环境的表现是：

- 多种事务处理并发执行，充分体现了事务处理的复杂性；
- 在线与离线的事务执行模式；
- 多个在线会话终端；
- 始终的系统运行时间和应用程序运行时间；
- 大量的磁盘 I/O 数据流；
- 强调事务的完整性要求(即 ACID 特性)；
- 对于非一致的数据分布，使用主键和从键进行访问；
- 数据库有许多大小不一、属性多样，而又相关联的数据表组成；
- 存放在脚多数据访问和更新之件的资源争夺。

为此，TPC-C 测试规范中模拟了一个比较复杂，并且具有代表意义的 OLTP 应用环境，来对数据库管理系统的联机事务处理性能进行测试。

2.2.2 测试模型

TPC-C 测试规范中模拟了一个比较复杂并具有代表意义的 OLTP 应用环境：假设有一个大型商品批发商，它拥有若干个分布在不同区域的商品库；每个仓库负责为 10 个销售点供货；每个销售点为 3000 个客户提供服务；平均每个客户的一个订单有 10 项产品；所有订单中约 1%的产品在其直接所属的仓库中没有存货，需要由其他区域的仓库来供货。

TPC-C 的事务处理在一个以 9 张表为基础的数据库上实现处理过程，执行的事务包括：更新、插入、删除、终止，以及对主键和外键的访问。对于前四类交易事务，要求 90%的事务执行的响应时间应在 5 秒以内；对于库存水平查询交易，则要求响应时间在 20 秒以内。同时，测试过程中还要求被测试系统保证数据库事务的 ACID 特性。

数据库逻辑结构以及表数据量关系如图 2-1 和表 2-2：

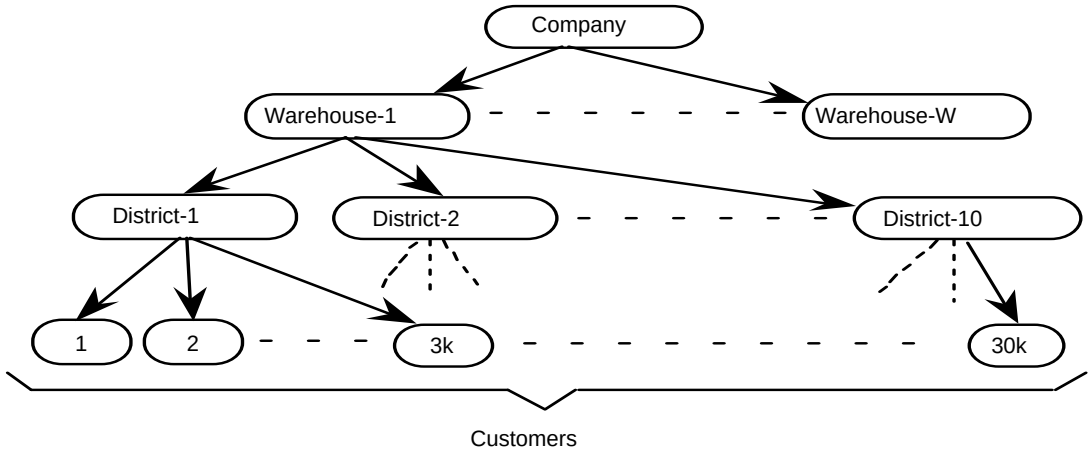

图 2-1　　数据库逻辑结构图

表 2-2　　9 张表的数据量及对应关系

Table Name	Cardinality (in rows)	Typical Row Length(in bytes)	Typical Table Size(in 1,000bytes)
WAREHOUSE	1	89	0.089
DISTRICT	10	95	0.950
CUSTOMER	30k	655	19,650
HISTORY	30k	46	1,380
ORDER	30k	24	720
NEW-ORDER	9k	8	72
ORDER-LINE	300k	54	16,200
STOCK	100k	306	30,600
ITEM	100k	82	8,200

2.2.3　五种事务说明

TPC-C 测试包括 5 个典型的 OLTP 事务，它们是

- New-Order(新订单事务)：客户输入一笔新的订货交易；
- Payment(支付事务)：更新客户账户余额以反映其支付状况；
- Delivery(发货事务)：发货(模拟批处理交易)；
- Order-Status(订单状态查询事务)：查询客户最近交易的状态
- Stock-Level(库存水平查询事务)：查询仓库库存状况，及时补货。

有关事务的具体描述如下：

- 新订单：其主要事务内容为对与任意一个客户端，从固定的仓库中随即选取 5-15 件商品，创建新订单。其中 1%的订单，要由于假想的用户操作失败而回滚。该事务的主要特点为读写、频繁、要求响应快，是系统中最典型的操作，也是系统处理中的主要工作量，最终也是以数据库系统每分钟能够处理的新订单数来对数据库系统的性能进行评价。
- 支付操作：其主要事务内容是对于任意一个客户端，从固定的仓库中随即选取一个辖区及其内的用户，采用随机的金额支付一笔订单，并且同时将该订单记录为相应历史订单。该事务的主要特点为 10 个批量、读写、较少、较宽松的响应时间。
- 订单状态查询：其主要事务内容为对于任意一个客户端，从固定的仓库中随机选取辖区及其内的用户，读取该用户的最后一条订单，显示

订单内每件商品的状态。该事务的主要特点为只读、较少、要求响应快。

- 发货：其主要事务内容是对于任意一个客户端，随机选取一个发货包，更新被处理订单的用户账户余额，并把修改后的订单从新订单中删除。该事务的主要特点为读写、频繁、响应快。
- 库存状态查询：其主要事务内容是对仪任意一个客户端，从固定的仓库和辖区选取最后的 20 条订单，检查订单中所有货物的库存。计算并显示所有库存低与随机生成的商品数量。该事务的主要特点为只读、较少、较为宽松的响应时间。

对于以上这 5 种类型的事务交易，前 4 种类型的交易要求响应时间在 5 秒以内；对于库存状态查询交易，要求响应时间在 20 秒以内。这 5 种交易最终的状态查询的比例分别不得少于 4%，5 种事务所要满足的时间、比例以及隔离级别如表 2-3：

表 2-3　事务比例

事务类型	事务最小百分比	最小键盘输入时间（秒）	90%事务响应时间要求（秒）	最小平均思考时间分布（秒）
新订单	n/a	18	5	12
支付	43.0	3	5	12
订单状态查询	4.0	2	5	10
发货	4.0	2	5	5
库存状态查询	4.0	2	20	5

2.2.4　测试指标

TPC-C 测试的结果主要有两个指标。即流量指标(Throughput,简称 tpmC)和性价比(Price/Performance，简称 Price/tpmC)：

- 流量指标(Throughput)tpmC.按照 TPC 组织的定义，流量指标描述了系统在执行支付操作、订单状态更新、发货和库存状态查询这 4 种交易的同时，每分钟可以处理多少个新订单交易。所有交易的响应时间必须满足 TPC-C 测试规范的要求，并且各种交易数量所占的比例也应该满足 TPC-C 测试规范的要求。在这种情况下，流量指标值越大说明系统的联机处理能力越高。目前 TPC 组织发布的最高 tpmC 值可以达到 1,000,000 以上、(通常由 IBM、HP 等主要硬件厂商发布，大都采用昂贵的集群服务器、高速的磁盘阵列设备，并配合快速的事务/消息中间

件系统)，即每分钟处理超过一百万个 NewOrder 事务。

- 性价比(Price/Performance，简称 Price/tpmC)：即测试系统的价格与流量指标的比值。价格指的是系统的总价格，单位是美元，而价格性能比为总价格/性能，单位是＄/tpmC。显然性价比最小越说明该测试系统的市场竞争力越强。目前 TPC 组发布的最高 tpmC 的系统，其性价比一般为 5-6US$/tpmC，而最好的性价比为 1.5-2.5US$/tpmC,其 tpmC 一般为 100,000-200,000tpmC。由此可见，这个指标的大小主要考察的是应用系统的商业价值，将更多的注意力集中于被测试系统的软硬件综合价格的合理配置。

2.2.5 测试工具、方法以及结果

按照 TPC-C 测试规范要求，测试工具和模型可以由厂商自行实现。在本文中，我们按照 TPC-C 标准规范自行开发了测试工具 TpccTesting。

常规情况下，一个实现了基本 TPC-C 测试模型的测试程序应该具有如下体系结构：即客户端模拟大量并发用户，由一个任务分发进程对用户的任务进行管理，按照某种方法，发送到 DBMS 执行，同时统计测试过程中各个事务的响应时间并计算 TPC-C 的吞吐量指标——tpmC。

而 TPC-C 的测试结果按照 TPC 组织的规定，有两种形式发布：测试结果概要(Executive Summary)和详细测试报告(Full Disclosure Report)[6]。测试结果概要中描述了主要的测试指标、测试环境意图以及完整的系统配置与报价，而详细测试报告中除了包含上述内容外，还详细说明了整个测试环境的设置与测试过程。

第三章　TpccTesting 的设计

3.1　数据库设计

从 TPC-C 基准规范的测试模型的描述中，可以 TPC-C 事务处理所需要的 9 张表分别为：Warehouse, District, Customer, History, Order, New-Order, Order-Line, Stock, Item。根据图 2-1 以及图 2-2 所示的数据库逻辑关系以及 9 张表数据量的对应关系，我们可以得到这 9 张表的实体联系图，如图 3-3 所示。

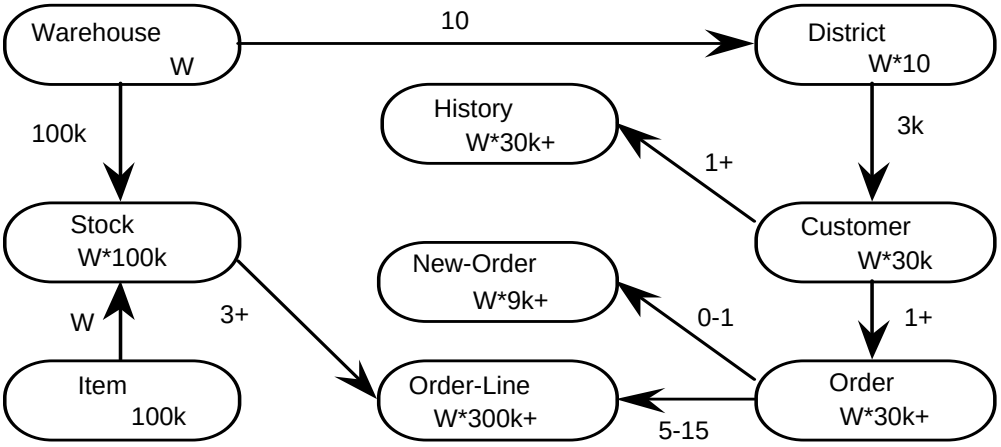

图 3-3　数据库表的 E-R 图

下面给出 TPC-C 基准测试程序所涉及的表的详细设计，如表 3-1 到表 3-9 所示，表中深色填充的为主键，浅色为外键：

1．商品仓库表

表 3-1　WAREHOUSE

属性名	属性定义	描述
W_ID	2*W 不同的 ID	建立 W 个仓库
W_NAME	variable text, size 10	
W_STREET_1	variable text, size 20	
W_STREET_2	variable text, size 20	
W_CITY	variable text, size 20	
W_STATE	fixed text, size 2	
W_ZIP	fixed text, size 9	
W_TAX	signed numeric(4,4)	销售税
W_YTD	signed numeric(12,2)	进行数据平衡的年份

主键：W_ID

2．销售地区表

表 3-2　　DISTRICT

属性名	属性定义	描述
D_ID	20 不同的 ID	每个仓库对应 10 个地区
D_W_ID	2*W 不同的 ID	
D_NAME	variable text, size 10	
D_STREET_1	variable text, size 20	
D_STREET_2	variable text, size 20	
D_CITY	variable text, size 20	
D_STATE	fixed text, size 2	
D_ZIP	fixed text, size 9	
D_TAX	signed numeric(4,4)	销售税
D_YTD	signed numeric(12,2)	进行数据平衡的年份
D_NEXT_O_ID	10,000,000 unique IDs	下一次订购数量
主键: (D_W_ID, D_ID)		
外键：D_W_ID 对应 W_ID		

3．消费者表

表 3-3　　CUSTOMER

属性名	属性定义	描述
C_ID	96,000 不同的 ID	每个地区供应 3000 个消费者
C_D_ID	20 不同的 ID	
C_W_ID	2*W 不同的 ID	
C_FIRST	variable text, size 16	
C_MIDDLE	fixed text, size 2	
C_LAST	variable text, size 16	
C_STREET_1	variable text, size 20	
C_STREET_2	variable text, size 20	
C_CITY	variable text, size 20	
C_STATE	fixed text, size 2	
C_ZIP	fixed text, size 9	
C_PHONE	fixed text, size 16	
C_SINCE	date and time	
C_CREDIT	fixed text, size 2	"GC"=good, "BC"=bad

C_CREDIT_LIM	signed numeric(12, 2)	
C_DISCOUNT	signed numeric(4, 4)	
C_BALANCE	signed numeric(12, 2)	
C_YTD_PAYMENT	signed numeric(12, 2)	
C_PAYMENT_CNT	numeric(4)	
C_DELIVERY_CNT	numeric(4)	
C_DATA	variable text, size 500	杂项信息
	主键: (C_W_ID, C_D_ID, C_ID)	
	外键：C_W_ID 对应 D_W_ID	
	C_D_ID 对应 D_ID	

4．历史记录表

表 3-4　HISTORY

属性名	属性定义	描述
H_C_ID	96,000 不同的 ID	
H_C_D_ID	20 不同的 ID	
H_C_W_ID	2*W 不同的 ID	
H_D_ID	20 不同的 ID	
H_W_ID	2*W 不同的 ID	
H_DATE	date and time	
H_AMOUNT	signed numeric(6, 2)	
H_DATA	variable text, size 24	Miscellaneous information
	主键: 无	
	外键：H_C_W_ID 对应 C_W_ID	
	H_C_D_ID 对应 C_D_ID	
	H_C_ID 对应 C_ID	
	H_W_ID 对应 D_W_ID	
	H_D_ID 对应 D_ID	

5．新定单表

表 3-5　NEW-ORDER

属性名	属性定义	描述
NO_O_ID	10,000,000 不同的 ID	
NO_D_ID	20 不同的 ID	

NO_W_ID	2*W 不同的 ID	
主键: (NO_W_ID, NO_D_ID, NO_O_ID)		
外键：NO_W_ID 对应 O_W_ID		
NO_D_ID 对应 O_D_ID		
NO_O_ID 对应 O_ID		

6．订单表

表 3-6　　ORDER

属性名	属性定义	描述
O_ID	10,000,000 不同的 ID	
O_D_ID	20 不同的 ID	
O_W_ID	2*W 不同的 ID	
O_C_ID	96,000 不同的 ID	
O_ENTRY_D	date and time	
O_CARRIER_ID	10 unique IDs, or null	
O_OL_CNT	numeric(2)	Count of Order-Lines
O_ALL_LOCAL	numeric(1)	
Primary Key: (O_W_ID, O_D_ID, O_ID)		
(O_W_ID, O_D_ID, O_C_ID) Foreign Key, references (C_W_ID, C_D_ID, C_ID)		

7．订购商品表

表 3-7　　ORDER-LINE

属性名	属性定义	描述
OL_O_ID	10,000,000 不同的 ID	
OL_D_ID	20 不同的 ID	
OL_W_ID	2*W 不同的 ID	
OL_NUMBER	15 不同的 ID	
OL_I_ID	200,000 不同的 D	
OL_SUPPLY_W_ID	2*W 不同的 ID	
OL_DELIVERY_D	date and time, or null	
OL_QUANTITY	numeric(2)	
OL_AMOUNT	signed numeric(6, 2)	
OL_DIST_INFO	fixed text, size 24	
主键: (OL_W_ID, OL_D_ID, OL_O_ID, OL_NUMBER)		

外键：OL_W_ID 对应 O_W_ID OL_D_ID 对应 O_D_ID OL_O_ID 对应 O_ID OL_SUPPLY_W_ID 对应 S_W_ID OL_I_ID 对应 S_I_ID		

8．商品信息表

表 3-8　　ITEM

属性名	属性定义	描述
I_ID	200,000 不同的 ID	100,000 items are populated
I_IM_ID	200,000 不同的 ID	Image ID associated to Item
I_NAME	variable text, size 24	
I_PRICE	numeric(5, 2)	
I_DATA	variable text, size 50	Brand information
主键: I_ID		

9．库存表

表 3-9　　STOCK

属性名	属性定义	描述
S_I_ID	200,000 不同的 ID	一个仓库存有 100,000 件商品
S_W_ID	2*W 不同的 ID	
S_QUANTITY	signed numeric(4)	
S_DIST_01	fixed text, size 24	
S_DIST_02	fixed text, size 24	
S_DIST_03	fixed text, size 24	
S_DIST_04	fixed text, size 24	
S_DIST_05	fixed text, size 24	
S_DIST_06	fixed text, size 24	
S_DIST_07	fixed text, size 24	
S_DIST_08	fixed text, size 24	
S_DIST_09	fixed text, size 24	
S_DIST_10	fixed text, size 24	
S_YTD	numeric(8)	
S_ORDER_CNT	numeric(4)	

S_REMOTE_CNT	numeric(4)	
S_DATA	variable text, size 50	制造信息
	主键: (S_W_ID, S_I_ID)	
	外键：S_W_ID 对应 W_ID	
	S_I_ID 对应 I_ID	

3.2 体系结构设计

一般程序设计结构分 B/S 结构和 C/S 结构，针对 TPC-C 测试程序而言，由于其规范要求的内容以及 TPC-C 模型所模拟的处理环境，有不同的设计方法。

在 B/S 结构下，大都采用 3 层软件体系结构，即是：安装数据库和数据库管理系统，实现数据层；服务器端使用 Web 服务器和相关中间件，实现业务逻辑层；最后使用远程终端模拟器模拟大量访问仓库的用户，实现表示层。测试系统结构如图 3-1 所示。

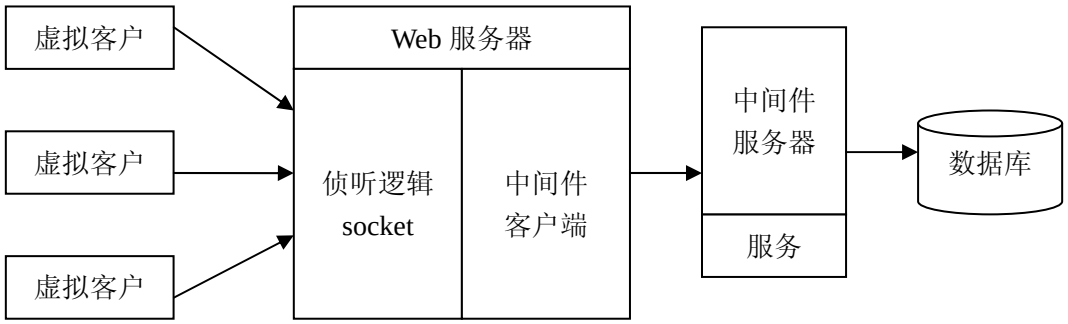

图 3-1　　三层软件体系结构

在 C/S 结构下，不需要使用 Web 服务器以及相关中间件，只需要采用网络编程技术创建在本机创建服务器端和客户端进行事务处理的模拟。这时客户端模拟大量并发用户，由一个任务分发进程对用户 任务进行管理，按照某种方法发送到 DBMS 执行，同时统计测试过程中各个事务的响应时间并计算 TPC-C 的吞吐量即可。由于 C/S 的配置环境和要求较少，并且易于安装和更新测试程序，所以在本文中将采用了 C/S 结构实现 TpccTesting。其体系结构如图 3-2。

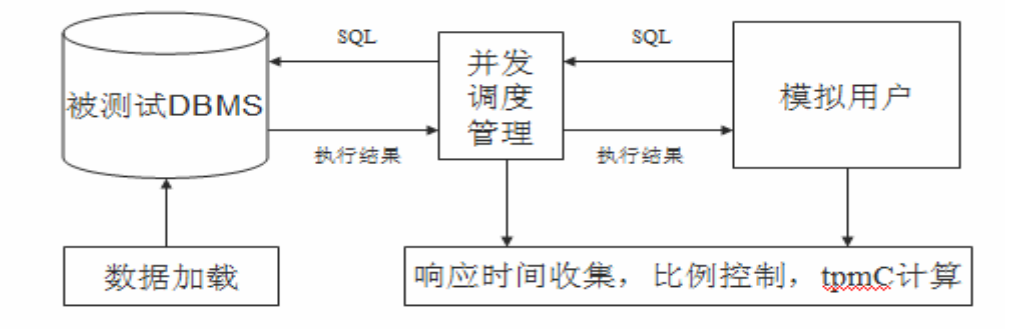

图 3-2　　TpccTesting 的体系结构

3.3　模块设计

3.3.1　模块划分

除了 TPC-C 的基准测试之外，许多其它的基准测试规范所建立的模型与 TPC-C 模型的调用结构都相似之处，都需要在其相应模型下所有模拟的连接池、用户事务管理、以及并发调度等模块。这样，我们充分使用面向对象设计相关技术以及软件工程中的相关开发方法，对 TpccTesting 进行模块化的设计。这样也就为其他需要未来需要测试的基准测试标准建立了一个基本的体系结构，其他的基准测试可以重用现有的许多模块和源码，然后改写自身标准中所说明的事务/任务的语句实现和其他功能需求，这样就可以在我们所设计的模块基础上构建其他测试工具的模型，帮助测试人员迅速构建所需要的测试程序和平台

基于以上的目的，论文采用了基于 C#语言的面向对象设计方法和软件工程中的开发方法和技术，将各个模块和对象封装在各个类中。对于测试程序的更新修改，只需要添加新的子模块，或者扩充所要更改的类的方法即可。这也是 C/S 结构下模块化设计的一大优点。

主要设计模块包括：

- 客户端服务器模块：用于创建客户端和服务器端，进行相关的通信交互模拟，并对客户提交的任务进行相关操作。
- 数据库模块：建立数据库，建立数据库表，并按输入向表内添加数据；进行数据库连接操作，并且创建数据库连接池，对连接池进行管理。
- 事务处理模块：主要是封装 TPC-C 规范中要求测试的 5 大事务。
- 信息收集统计模块：对测试过程进行监控，收集并统计事务的执行状态，根据 TPC-C 规范要求的事务分配比例进行控制。

根据系统的模块化设计以及 C/S 结构下测试程序的大致体系结构，从总体框架上将测试程序分为以下五个部分：被测试系统(DBMS)、TPC-C 测试客户端、

主从机端、并发调度/管理进行、系统监控与统计信息收集。它们之间的信息传递关系如图 3-3 所示：

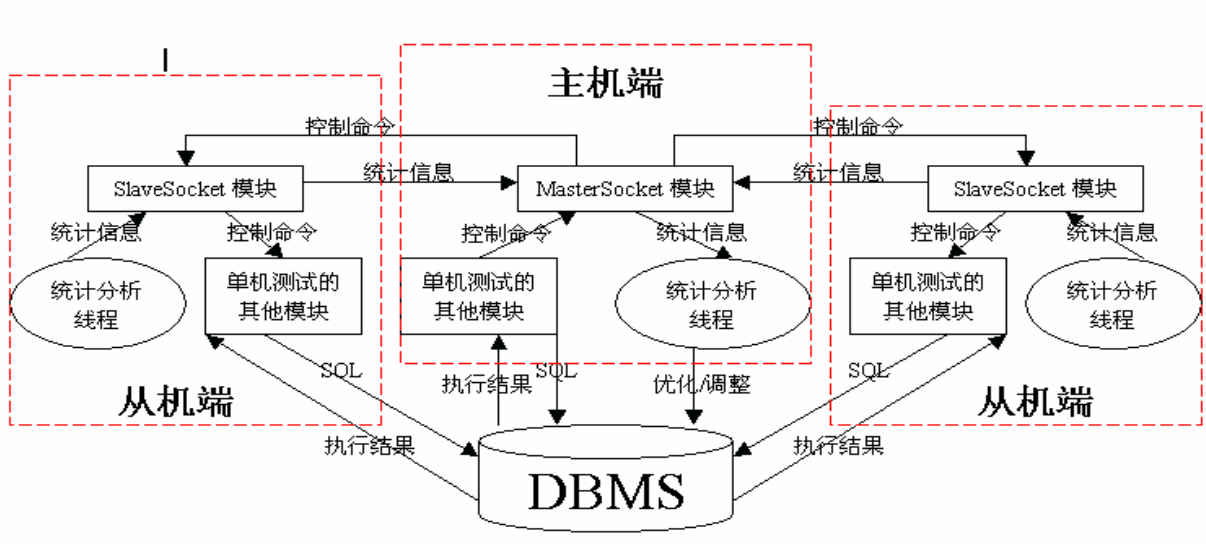

图 3-3　　结构框架信息传递示意图

测试程序的各大部分分别对应 C#类中的若干对象类，通过它们提供的方法借口，实现各个对象之间的信息传递与通讯。

3.3.2　数据库连接池管理

TPC-C 模型所描述的并发模型是大量用户对于数据库服务器的高负载并发访问。而数据库服务器所能够维护的连接数目是有限的，如果为每个并发用户都维护一个专有的连接，很明显地，在用户并发数量达到数百个之后，服务器就会因为需要维护数量巨大的用户连接而将系统资源耗尽。此时，对于连接的维护和管理就会成为基准测试的瓶颈，相对于真正的 OLTP 处理，显然这样测试所得到的结果不能真实反映数据库服务器的 OLTP 处理能力。因此，在这里我们需要建立数据库连接池来对数据库连接进行管理，让测试程序维护全部与数据库服务器建立的连接，将连接的管理与维护工作转移到测试程序一方处理，解决了服务器的连接维护瓶颈[6]。

3.3.3　用户事务管理

对 TPC-C 规范所要求测试的五种事务处理，由 TPC-C 模型的高并发负载的特点可以了解，在用户增加到比较庞大数量时，如上千个用户并发访问，那么在某一时刻就会有数百个甚至更多的事务执行请求，这时如何配合连接池的管理，来处理大量的用户任务，这就需要一个有效的调度方法。因为，TPC-C 在最终的结果输出时需要统计每个用户请求的响应时间与事务的执行数量、比

例、成功与否等情况。这时如果由每个用户单独从连接池中等待可用的连接，用户往往会无法精确统计一个用户请求从发出到接收执行的响应时间，而且可能发生一个用户长期占用一个连接，而其用户却一直处于等待状态的情况。

根据以上特点，在程序中，我们通过多线程技术来实现多用户的并发访问，每一个线程代表一个用户。同时，如果对线程（用户）进行直接调度，无论采用哪种调度方法都有可能出现某一个用户请求等待过长的时间才能获取可用的数据库连接，即等待时间超过了 TPC-C 规范所要求的 KeyTime 和 ThinkTime 之和。在这里我们采用将用户与用户行为分离的方法来解决这个问题，即每个用户与 TPC-C 说明的每个用户的行为是分别实现的。用户发出请求，我们把用户任务添加到任务队列中去，实现对任务执行的调度而并非直接对用户进行调度，这样用户发出任务后就可以开始响应时间的统计，而无需等待连接池的反馈[11]。这样可以保证用户响应时间统计的准确性，也可以保证用户任务执行的公平性。不会出现某个用户迟迟等不到其任务被执行的情况，也避免了大量用户并发访问连接池可能带来的访问冲突。

3.3.4 测试过程的监控和统计

基于本程序的设计目的和功能需求，我们还应该在程序中加入系统运行状态监控、测试过程中的信息统计功能。系统中应该存在一个监控进程，对系统运行中的统计信息进行收集和计算。对于操作系统级上的统计信息，操作系统一般都提供了相应的统计工具和获取这些统计信息的编程接口。对数据库级别上的统计信息，DBMS 都提供了相应的系统视图。通过对这些统计信息的收集，我们就可以充分了解基准测试的执行状况，以便于优化测试方式。

第四章　TpccTesting 的实现

4.1　测试系统的实现

测试系统 TpccTesting 编程语言使用 C# .NET，编译环境为 Visual Studio 2005，数据选用 SQL Server 2000，操作系统为 Windows XP。本节主要阐述该测试系统的执行流程，以及实现上的一些细节。

4.1.1　测试系统的执行流程

从 TPC-C 测试规范可以看出，在一次测试开始时，测试系统开始测试时需要一些准备操作，即调用测试程序的数据加载模块，创建数据库并加载一定数量的数据，然后将这些数据备份起来以备测试失败或数据错误时能够再次利用此次数据加载的基础进行 DBMS 参数调整。数据备份后对测试系统中的各种参数进行配置(包括：DBMS、操作系统等配置)，然后调用 TPC-C 测试模块进行测试。测试后测试程序输出测试过程中收集到的 TPC-C 事务响应时间的统计信息与系统吞吐量指标 tmpC，测试人员需要根据响应时间判断本次测试是否符合 TPC-C 规定的响应时间标准。如果事务响应时间符合 TPC-C 标准，则测试人员需要决定新的测试中需要加载的数据量，然后循环执行前面的步骤。如果事务响应时间不符合标准，则说明当前系统的性能不能满足 TPC-C 标准的要求，需要对当前系统进行调整。此时恢复备份的数据，并根据前一次测试的情况判断需要调整的各项参数，并在参数调整完后重新进行测试。在多次参数调整都失败后，则认为系统不能支持当前的数据量，已达到系统的最大负载，则取上一次通过的测试结果为最终的测试结果，并将其响应时间与 tpmC 值写入 TPC-C 测试报告。

测试流程如图 4-1：

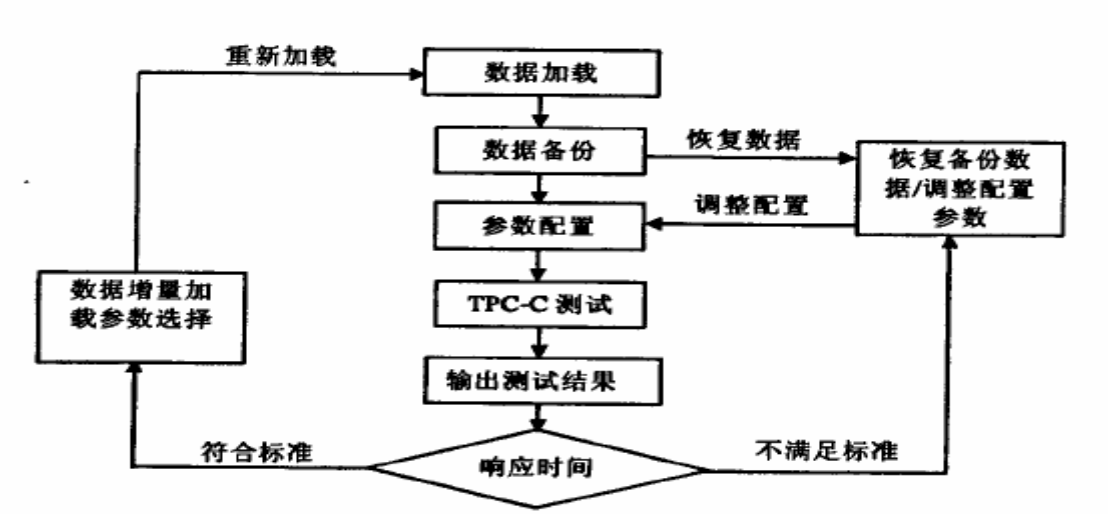

图 4-1　　测试流程图

4.1.2　测试系统的用户界面

明确了测试程序的执行流程后，就需要对其做可视化界面的设计，而最基本的工作就是创建窗体。窗体本身主要于应用程序执行时，提供与用户沟通的窗口界面，创建一个窗口程序的窗体画面，通常需利用类 Form 来完成，类 Form 位于命名空间 System.Windows.Forms，其继承了位于相同命名空间的类 ContainerControl，这个类被设计用以当作其他控件的容器，并且提供管理控件的功能，Form 类拥有不同的方法以及属性成员，提供修改窗体外观以及相关性质的操作。

Visual Studio .NET 提供了使用鼠标拖拽可视化组件的设计环境，这样比使用 No tepad 来实现可视化界面要简单很多[7]。这让我们可以快速的完成应用程序中设计用户界面相关程序代码的创建，而将大部分的时间集中于编写真正使用于解决的逻辑程序代码。Form 类允许在窗体上配置数量不等的各类控件，以及 Control 类提供的相关事件，如：键盘事件(KeyDown、KeyPress 和 KeyUp)、鼠标事件(MouseDown、MouseUp 和 MouseEnter 等)、Paint 事件和 Windows 控件等。

在这里需要注意的一个问题就是，本程序的实现需要多个窗体：数据管理、测试记录和曲线分析，那么如何让两个窗体进行沟通，并且于其中相互传递信息数据。一般有 2 个方法：其一是，由当前窗体本身创建另一个新的窗体对象，并且将信息传递给这个新的窗体对象，很明显这个方法较为单纯，而且不适用于本程序；我们采用第 2 类方法实现数据传递，即利用事件委派的机制来完成两个单独的窗体，如图 4-2 所示：

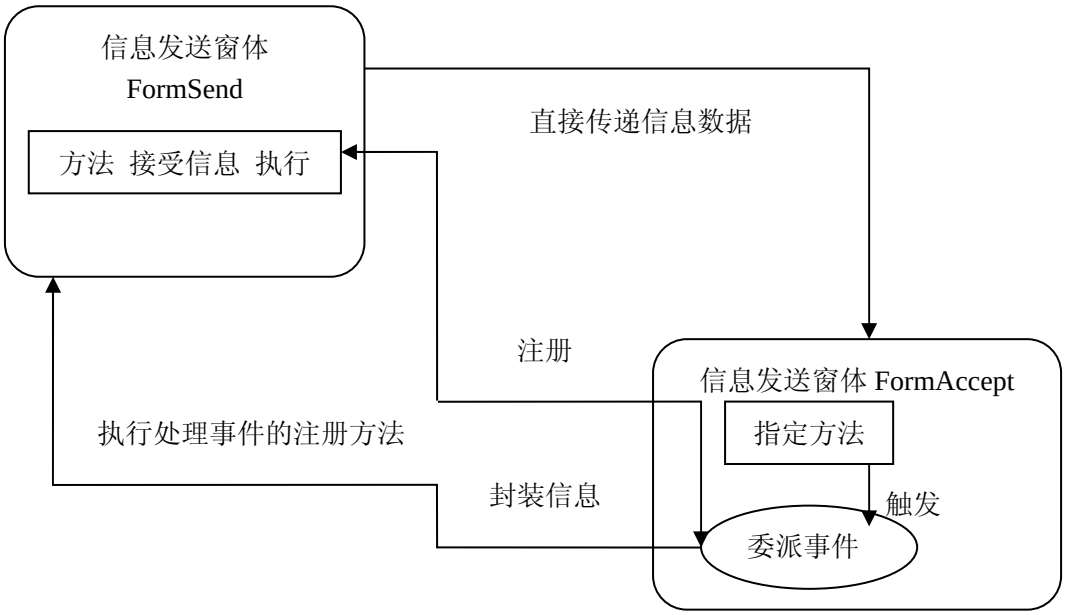

图 4-2　　使用委派事件进行沟通的过程

窗体对象 FormAccept 首先必须声明创建所需要的委派事件，并且是适当的方法触发事件，本程序是在窗体相互切换时将触发事件；而 FormSend 窗体对象除了创建 FormAccept 对象实例，同时必须注册 FormAccept 对象其相应事件被出发时所要执行的方法，并且经过所触发的事件，取得 FormAccept 窗体对象所传递的信息。

系统界面如图 4-3。

图 4-3　　TpccTesting 主界面

4.1.3 测试数据的加载

系统在开始测试前需要创建数据库，并且向数据库内加载信息，由第二章对 TPC-C 模型介绍中的数据库逻辑图(图 2-1)和数据量关系图(图 2-2)可以看出，各个表的数据量之间存在着特定的比例关系。这一点在第三章的数据库设计中就可以更明显的看出来，由数据库 E-R 图(图 3-3)可知，TPC-C 的数据量可以用 Warehouse 表中的数据量(行数)为基数进行估算。也就是说，数据库中有多少个 Warehouse 就可以确定其他 8 个表的数据量大小，即是数据库中数据总量的大小。所以在 TPC-C 测试报告中说明的性能指标可以达到支持多少个 Warehouse 数目，也就说明了其可以支持的数量大小[10]。

因此，在开始测试之前，由测试人员只需要输出建立 Warehouse 的个数即可，其他表的数据量则根据各表间的相应关系由系统自动确定，然后配置数据库网络等参数。点击“开始测试”按扭后，系统首先建立与 SQL Server 的连接，然后按照 3.2.1 节中数据库 9 张的属性结构建立数据库，然后开始进行数据加载。加载时，为了方便快捷所有表的所有属性都插入相同的数字，从 1 开始，一直到测试人员所要求的最大数据量的数字。最后断开与数据库的连接，进行对数据库事务处理的测试阶段。

这里数据库连接以及其他相关操作介绍详见本章 4.2.1 节。

4.1.4 事务处理的细节

在第 2.2.3 节中，已经对事务的处理有了大概的介绍，本节首先给出非均匀的随机函数 NURand(A,x,y) [10]的定义，然后详细说明五种事务处理上的一些细节。

非一致的随机 NURand(A,x,y) ：

NURand(A,x,y) = (((random(0,A) | random(x,y)) + C)%(y-x+1))+x

其中，“|”表示二进制逻辑或运算，“%”表示余运算；random(x,y)表示取 x 到 y 之间的随机数；A 是在 x 到 y 之间的一个常量(对于 C_LAST 属性，x=0，y=999，A=255；对于 C_ID 属性，x=1，y=300，A=1023；对于 OL_I_ID 属性，x=1，y=100000，A=8191)；C 是一个在没有改变性能时，可以被改变的在[0,A]上的随机恒量，相同的 C 值，在每个域(C_LAST，C_ID 和 OL_I_ID)上必须被所有的模拟终端使用。

新订单事务：客户选择时 C_ID 是 NURand(1023,1,3000)的结果，商品数目则是在[5,15]上的随机数(平均为 10)。对于事务说明中的 1%假想的用户输入失败，其选择方法为：先在[1,100]上随机选择一个 rbk 的值，然后使用

NURand(8191,1,100000)计算 OL_I_ID 的值，得到的商品为最后一个订购的商品且 rbk=1，则为无效的输入。而 1%的供应在供应仓库无货的处理方法为：取值 x 为在[1,100]上的随机数，若 x=1 则从偏远仓库供货，随机从其他 9 个仓库中选择一个。某一商品数量为在[1,10]上的随机数。

支付事务：主要为更新 CUSTOMER 表的数据。在选择支付的用户时，有 60%的根据末姓(C_W_ID，C_D_ID，C_LAST)随机选择，40%由客户编号(C_W_ID，C_D_ID，C_ID)选择。而供应地仓库 85%是本地仓库，15%为偏远仓库。实现时取 2 个在[1,100]上的随机数 x 和 y，x<=85 时客户选择时 C_W_ID 为当前选择的 W_ID，x>85 时 C_W_ID 随机另外 9 个仓库中的一个，y<=60 时 C_LAST 由 NURand(255,0,999)产生，y>60 时 C_ID 由 NURand(1023,1,3000)产生。[10]

订单查询事务：在选择查询的用户时，有 60%的根据末姓(C_W_ID，C_D_ID，C_LAST)随机选择，40%由客户编号(C_W_ID，C_D_ID，C_ID)选择。处理与支付事务中用户选择的处理一样。

发货事务：O_CARRER_ID 为[1,10]之间的随机数。OL_DELIVERY_D 为当地时间。

库存状态事务：选择最后 20 个订单。

五种事务在分配时，取数 k 的值为在域[1,100]上的随机数，如果 k<=38 则为新订单事务，39<k<=82 为支付事务，83<k<=88 为订单状态查询事务，89<k<=94，95<k<=100 为库存查询事务。

4.2 实现中的关键技术

本节将说明实现本测试系统所用的关键技术：对数据库连接的管理，模拟 OLTP 的应用环境以及多用户并发访问数据库管理系统的模拟。

4.2.1 数据库连接管理

在.NET Framework 问世之前，开发人员一直使用 ODBC、OLE DB 和 ADO 等数据访问技术。随着.NET 的引入，Microsoft 创建了一种处理数据的新方法，将其命名为 ADO.NET。ADO.NET 本身是.NET Framework 提供的一组专门用以存取数据的类，配合 SQL 语法对于数据库进行数据存取操作[7]。除此之外，它还扮演应用程序连接数据库的桥梁。其对象被设计使用于开发能够操作各种数据源的数据库应用程序，你能够利用 ADO.NET 所提供的各种类创建一套完整的数据库系统。

ADO.NET 有两个核心组件：数据提供程序和数据集。

数据提供程序(data provider)连接数据源，支持数据访问和处理。其中包含了几个主要的对象，如 Connection(创建数据源的连接，打开数据库)、Command(用以存取以及变动数据源的数据内容，执行存储过程)以及 DataAdapter(提供数据源与断线对象 DataSet 之间的桥梁)等。

数据集(data set)支持数据以关联的方式，在断开连接的情况下独立地缓存数据，根据需要更新数据源。

1．使用 ADO.NET 进行数据库连接的操作。

基本上程序的代码会进行以下几个相关的操作：

- 创建连接对象(Connection Object)。
- 使用连接对象连接并且打开指定的数据库。
- 使用连接对象创建 Command 对象搜索数据。
- 创建 DataAdapter 对象返回数据。
- 运用 DataAdapter 对象获取各种 Command 对象进行数据库的变动、返回记录集。
- 通过 DataAdapter 对象返回包含选取数据的高速缓存(cache)的 Dataset 对象，并且存放于客户端的机器上。

2．SQL Server 数据提供程序。

ADO.NET 的命名空间中包含许多数据库的数据提供程序，本测试系统使用的数据库为 SQL Server 2000，所以这里主要列出 SqlClient 命名空间中的一些重要的类。

表 4-1　　常用的 SqlClient 类

类	内容说明
SqlCommand	执行 SQL 查询、语句或者存储过程
SqlConnection	代表与 SQL Server 数据库的连接
SqlDataAdapter	代表数据集和数据源之间的桥梁
SqlDataReader	为结果提供只向前的只读数据流
SqlError	保存 SQL Server 错误和警告的信息
SqlParameter	代表命令的参数
SqlTransaction	代表 SQL Server 事务处理

另一个命名空间 System.Data.SqlTypes 把 SQL Server 数据类型映射为.NET 类型，提高了性能简化了编程。

3．使用 SqlConnection 连接和关闭数据库。

首先创建连接字符串。连接字符串由指定连接信息的参数组成，换言之就是用分号隔开的 key=value 对。

由于本系统安装环境不同，数据库连接的用户以及密码就会不同，所以这对参数 server = (local)\netsdk 指定了要连接的数据库为 SQL Server，而 integrated security = sspi 指定使用 Windows 身份验证来登陆 SQL Server，这样就不需要更改此段代码。

接着创建连接(SqlConnection 对象)，给它传递连接字符串。

然后在连接上调用 Open 方法，建立一个与数据库的会话。

一般情况下这句代码被包含在异常处理语句 try{}catch{}中，以便在数据库打开失败后做其他适当的处理。

使用完数据库后，调用 Close 终止会话。

代码如下：

```
String connString = ”server = (local)\netsdk; integrated security = sspi”;
SqlConnection conn = new SqlConnection(connString);
conn.Open();
//detailed operations
…….
conn.Close();
```

4．执行语句。

使用 Command 对象创建命令，为了对数据库执行命令，必须把每个命令与数据库的连接关联起来：

SqlCommand comm. = new SqlCommand();

comm.Connection = conn;

执行命令的方法有以下几个：ExecuteNonQuery，ExecuteScalar，ExecuteReader，ExecuteXmlReader。

执行语句使用 ExecuteNonQuery 方法，它不执行查询。我们用它来创建数据库和表。创建数据库只要执行 Transact-SQL 语句 CREATE DATABASE 即可：

comm.CommandText=”CREATE DATABASE merchant”;

comm.ExecuteNonQuery();

由于在创建数据库前我们无法连接该数据库，所以我们使用 ChangeDatabase 的方法来连接创建的刚数据库：

comm.ChangeDatabase(“merchant”);

之后使用 Transact-SQL 语句 CREATE TABLE 按第三章 3.2.1 节中所述的表的结构依次创建 9 张表。表创建完成后，按照测试人员输入的 Warehouse 的个数，开始向表中插入数据，使用 Transact-SQL 语句 INSERT INTO 插入，然后调用 ExecuteNonQuery 执行语句。

对于五种事务处理中所需要的查询语句，则是由 ExecuteScalar 来执行单个结果的命令，ExecuteReader 来执行具有多个结果的命令。在这里对 Transact-SQL 的写法不再进行描述。

5．事务处理

在 ADO.NET 中，事务是实现 System.Data.IDbTransaction 借口的类实例[7]。事务没有自己的构造函数，而是通过调用另一个对象的方法来创建，在这里调用的连接的 BeginTransaction 方法。命令与特定连接的特定事务相关联，这些命令提交的任何 SQL 都执行为同一事务的一部分。

首先打开连接，再为它创建一个事务：

```
conn.open();
SqlTransaction obj = conn.BeginTransaction();
```

之后创建命令，指定它执行 SQL 而不是存储过程。在所执行的命令中指定事务，以后与命令关联的 SQL 作为这个事务的一副本来执行：

```
SqlCommand cmd = conn.CreateCommand();
cmd.CommandType = CommandType.Text;
cmd.Transaction = obj;
```

执行完所有命令后，则调用事务的 Commit 方法提交数据：

```
obj.Commit();
```

这里一定要在执行完所有命令后调用 Commit，因为它会生成永久的变化并结束当前的事务。

事务回滚时需要调用 Rollback 方法：obj.Rollback();

6．数据连接池

在第三章所阐述的测试系统设计，第 3.2.3 节提到了为了解决了服务器的连接维护瓶颈，显著提高应用程序的性能和可缩放性，我们需要建立数据库连接池来维护接连。

连接池减少新连接需要打开的次数。池进程保持物理连接的所有权。通过为每个给定的连接配置保留一组活动连接来管理连接。只要用户在连接上调用 Open，池进程就会检查池中是否有可用的连接。如果某个池连接可用，会将该连接返回给调用者，而不是打开新连接。应用程序在该连接上调用 Close 时，池进程会将连接返回到活动连接池集中，而不是真正关闭连接。连接返回到池中之后，即可在下一个 Open 调用中重复使用。默认情况下，ADO.NET 中启用连接池。除非显式禁用，否则，连接在应用程序中打开和关闭时，池进程将对连接进行优化。表 4-2 描述了可用于调整连接池行为的连接字符串值。我们只需要在创建连接字符串时，加入这值即可，关于如何连接上面阐述过，这里不再说明。

表 4-2　　连接字符串值

类	默认值	内容说明
Connection Lifetime	0	连接返回到池中后，创建时间将与当前时间进行比较，如果时间跨度(秒)超过 Connection Lifetime 指定的值，该连接将被破坏。在聚集配置中可以使用它来强制在运行服务器和刚联机的服务器之间达到负载平衡。 如果值为零 (0)，则将使池连接具有最大的超时期限。
Enlist	‘true’	当为 true 时，如果存在事务上下文，池管理程序将自动在创建线程的当前事务上下文中登记连接。
Max Pool Size	100	池中允许的最大连接数
Min Pool Size	0	池中维护的最小连接数
Pooling	‘true’	当为 true 时，将从相应的池中取出连接，或者在必要时创建连接并将其添加到相应的池中。

- 添加连接：连接池是为每个唯一的连接字符串创建的。当创建一个池后，将创建多个连接对象并将其添加到该池中，以满足最小池大小的要求。连接根据需要添加到池中，但是不能超过指定的最大池大小（默认值为 100）。
- 移除连接：连接池进程定期扫描连接池，查找没有通过 Close 或 Dispose 关闭的未用连接，并重新建立找到的连接。如果应用程序没有显式关闭或断开其连接，连接池进程可能需要很长时间才能重新建立连接，所以，最好确保在连接中显式调用 Close 和 Dispose。
- 清除池：使用 ClearAllPoolsClearPool 和 ClearAllPools 。清除给定提供程序的连接池，ClearPool 清除与特定连接关联的连接池。如果在调用时连接正在使用，将进行相应的标记。
- 事务支持：连接是根据事务上下文来从池中取出并进行分配的。除非在连接字符串中指定了 Enlist=false，否则，连接池将确保连接在 Current 上下文中登记。

按照众多实现的经验来说，对连接池大小的设定为：在用户数超过 100 个的情况下，连接池大小 = 20 + 总并发用户数/50。

4.2.2　多用户并发访问模拟

如第 3 章所述，为模拟大量并发用户的访问，我们需要使用多线程来模拟这一过程。

使用 C#创建并使用线程，最简单的方式便是创建 Thread 类的实例对象，在此之前，必须引用命名空间 System.Threading，事实上，应用程序本身包含了一个主线程，可以创建一个新的 Thread 对象，将需要分开执行的程序代码，放入这个新的线程里来执行，以达到于应用程序支持多线程的目的。

Thread 类被设计用以创建新的线程，控制线程的行为，其提供的方法成员支持线程的管理操作，并且被声明为 sealed，无法被继承[7]，以下为类定义：

public sealed class Thread;

1．创建线程。

Thread 类提供如下构造函数来构建线程：

public Thread(ThreadStart statPoint);

其中 statPoint 参数是一个 ThreadStart 的委派，这个委派被定义用以封装线程对象所要执行的方法，也就是说对于 public delegate void ThreadStart()实例化对象，里面包含了线程所要执行的内容。创建完线程对象，并且定义完所要执行的方法之后，我们就要调用 public void start()来使线程开始执行方法。除此之外，我们在设计时提到了用户与用户任务的分离，在实现中表现为：当线程(用户)提交事务处理请求后，服务器接收并放入任务管理队列后给线程发送确认信息，然后线程挂起等待事务执行完毕，这里需要确定事务是由哪个线程(用户)发出，并准确发送确认信息给相应的线程(用户)。所以，我们还创建时还需要使用 public string Name {get; set;}来为给予线程一个唯一确定的“名字”，这里使用数字编号来区分。，综上，创建步骤为：

a) 编写线程(用户)的执行内容函数 CustomerOperation()，其中主要内容如图 4-1 所示。
b) ThreadStart customerTheadStart = new ThreadStart(CustomerOperation);
c) 创建线程 Thread Threading = new Thread(customerTheadStart);
d) 为线程命名：Threading.Name = “1”;
e) 启动线程：Threading.Start();

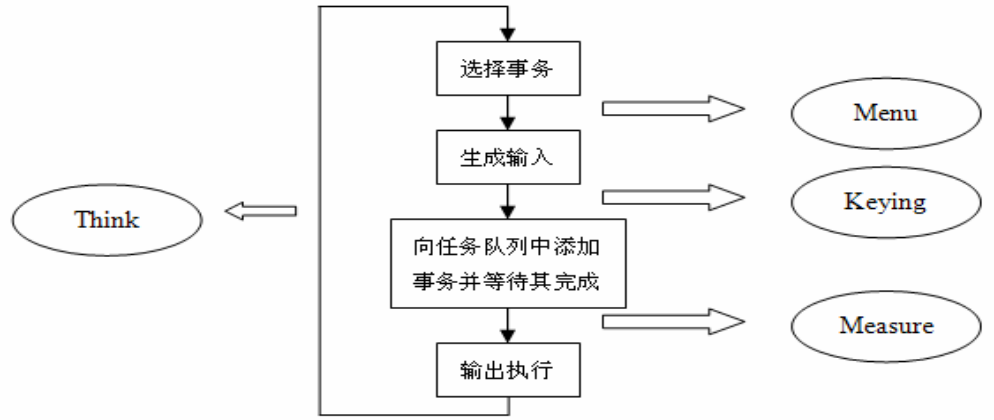

图 4-4　　一个虚拟并发用户的循环执行步骤

2．终止线程

一般对于一个线程，当其所需要执行的所有任务全部完成之后，该线程便会自动的终止。但有时我们需要强制地终止线程的执行，这就需要调用 Interrupt 方法。这个方法会阻断处于 WaitSleepJoin 线程状态下的线程，其定义非常简单：

public void Interrupt()

其为一个实例的方法，当线程个体使用此方法，指定的线程其执行的行为将会被阻断，当线程被迫终止时，会产生一个类型为 ThreadInterruptException 的线程终止异常，我们可以使用 try…catch 语句捕捉它。

3．同步线程

多线程的应用程序经常会遇到不同线程使用同一份资源的情形，在这种情况下，我们必须协调这些独立线程，这样的操作称为线程的同步化。

同步线程在资源一次只能由一个线程存取的时候特别重要，这可以避免同时存取一份资源所造成的冲突，C#利用 lock 语句来完成这一类的操作，用以控制某段程序代码对特定资源的存取，当这份资源由某个线程进行存取的时候，其他的线程将没有权限可以存取这份资源，直到其被释放。

lock 的语法如下

```
lock(objlock)
{
    //这里在本程序中主要是事务的选择，以及提交请求。
}
```

其中的 objlock 为所要锁定的对象，而其中则为所要封锁的执行代码，一旦于某个线程执行，objlock 对象会锁住直到其中的程序代码执行完毕，当这个锁定被释放后，其他线程才能够继续执行。

针对 TPC-C 规范的内容以及第三章的设计，明显地，我们对资源存取的行为较复杂，使用 lock 的控制不足，此时我们使用另外一个类 Monitor 来弥补不足。

4．Monitor 类

Monitor 类同样位于 System.Threading 命名空间，提供资源存取的控制机制，允许一个特定线程对资源作独占式的存取，与 lock 不同的是，它让你决定何时阻断或继续特定线程的执行操作。

Monitor 类提供一组静态方法，用以进行资源的控制，这里主要介绍与本系统相关的 Pulse 以及 Wait 这两个方法，定义如下：

public static bool Wait(object obj)

public static bool Pulse(object obj)

Wait 方法能用锁定目前的线程，并且将这个线程锁定的资源释放。当调用

Wait 暂时停止线程，Pulse 方法可以让你将暂停的线程唤醒，继续执行。这符合图 4-1 所示的并发用户循环执行的要求，其中 Thinking 这段时间，我们将其看作调用 Wait 直到被 Pulse 唤醒的过程。但麻烦的是，不论你调用 Wait 方法暂停几个线程，Pulse 方法总是会恢复第一个被暂停的线程，所以在程序需要对暂停线程进行一定的排队来解决这一问题。

4.2.3 OLTP 应用环境模拟

为了在 C/S 结构中模拟规范中所要求的 OLTP 环境，我们需要自己建立服务器和客户端，并且模拟用户在客户端向服务器发送请求的这一过程。为此，利用 C#的网络编程技术来实现这一要求。

网络编程的核心为网络协议(Internet Protocol，IP)[8]。无论哪种网络，我们都是通过 IP 协议来传输数据的，其提供了在网络设备之间传输数据的大部分功能，尤其是通过因特网连接的设备。此外还有最主要的两个利用 IP 的网络协议：传输控制协议(Transmission Control Protocol，TCP)和用户数据报协议(User Datagram Protocol，UDP)。[8]

对于网络编程技术的 URL 与 System.Uri 类，我们在这里不进行介绍，仅仅对本测试系统实现过程中所需要使用的 C#网络套接字编程进行说明。我们利用套接字(socket)来创建服务器和客户端，并使其进行相互之间的通信。为了确保模拟用户提交请求的可靠性和完整性，以及事务处理完后恢复用户运行的信号可以准确的收到，我们使用面向连接的套接字编程，即使用 TCP 协议的相关内容和方法进行数据传输。

1．Socket 类和 IP 相关

在 windows 环境下，针对 socket 编程，.NET 框架的 System.Net.Scockets 命名空间为需要严密控制网络访问的开发人员提供了 Winsockjie 接口的托管实现。[8]其中 Scoket 类是 Winsock32 API 提供的套接字服务的托管代码版本，为实现网络编程提供了大量的方法。

我们使用下面的构造函数来初始化 Socket 类的实例。

```
Public Scoket {
    AddressFamily addressFamily, //指定网络类型
    SocketType socketType,       //指定套接字类型(Dgram/Stream/Raw/Raw)
    ProtocolType protocolType    //指定网络协议(Udp/Tcp/Icmp/Raw)
};
```

一般来说，对于常规的 IP 通信网络，AddressFamily 只能使用 AddressFamily.InterNetwork。

在创建套接字(socket)之前，很明显的，我们需要对其 IP 地址进行处理。.NET 在命名空间 System.Net 中定义了 IPAddress 和 IPEndPoint 两个类来处理各种 IP 地址信息。对其内容我们不在这里详述，只介绍要用到的部分。首先使用 IPAddress 中的 Parse 方法将给定的字符串类型转换成 IPAddress 实例，以便在后面为 IP 地址赋值。在这里，由于本机建立服务器端和客户端的缘故，IP 地址使用本机地址“127.0.0.1”。

然后使用 IPEndPoint 类来描述一个主机的地址和端口信息，可以使用下面两个构造函数来创建 IPEndPoint 实例：

- IPEndPoint(long address, int port)
- IPEndPoint(IPAddress address, int port)

对主机端口，一般将其设置为 8080，利用上面转换的 IP 地址，完整的处理代码如下

```
IPAddress ipa = IPAddress.Parse(“127.0.0.1”);
IPEndPoint ie = new IPEndPoint(ipa, 8080);

Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
……
server.Bind(ie);
server.Listen(n);
server.Accept();
.....
```

对 IP 地址的处理完成后，开始进行套接字的创建。对面向连接的服务器端的套接字，在使用 Socket 类实例化对象后，我们需要将其与上面设定的 IP 地址绑定，方法为：Bind(EndPoint address //IPEndPoint 实例)。然后开始监听客户发送的连接请求，方法为：Listen(int con_num //这个参数表示了服务器可以接受的最大连接数目)。服务器进入监听状态后，如果有从客户端发来的连接请求，服务器使用 Accept()方法来接受连接请求。该方法会返回一个新的套接字，包含所建立的连接的信息并负责处理本连接的所有通信。

同服务器端一样，客户端的套接字在 Socket 类实例化后也必须与一个地址绑定。不同的是，这里需要使用 Connect(EndPoint remote_ep)方法，参数为代表想要连接的服务器的 IPEndPoint 实例。调用 Connect()方法后，它将一直阻塞到连接建立，如果连接不成功则返回一个异常。这与服务器端的监听对应，连接请求发出后，在服务器监听到有连接请求，然后调用 Accept()方法来接受连接请求，紧接着进行双方的通信。这里需要注意的是，服务器和客户端的建立顺

序以及连接请求的发送时间。如果建立顺序不对，或者在服务器为监听前发送连接请求，这样会产生错误甚至于死锁。

同过上述，在服务器和客户端的通信建立后，使用 Receive()和 Send()方法来传输数据。

数据传输完毕后，需要关闭套接字以释放其所占用的资源。这里调用 Shutdown(SocketShutdown how)和 Close()方法来完成这一工作。方法 Shutdown()用来禁用指定的套接字操作，其参数如表 4-3：

表 4-5　　SocketShutdown 枚举列表

成员	说明
SocketShutdown.Both	禁用发送和接收的套接字
SocketShutdown.Receive	禁用发送套接字
SocketShutdown.Send	禁用发送套接字

通过以上描述，本程序实现客户/服务器的时序如图 4-3 所示。其中，客户的实现是由多线程编程技术模拟大量并发用户来实现的，客户端以及服务器端的具体工作在前面的章节中已经介绍过，这里不再进行重复，仅给出客户(线程)/服务器的创建和结束过程。

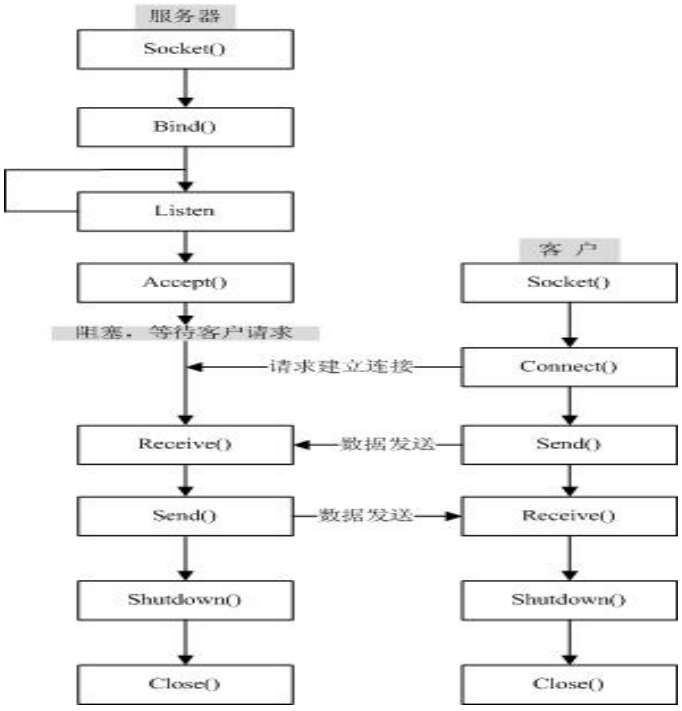

图 4-5　　面向连接的客户/服务器时序图

2．C#套接字助手类

.NET 框架为我们提供了标准的套接字接口来进行高级网络编程，但同时也为我们提供了套接字助手类来进行简单的套接字变成。这里主要介绍面向连接

的两个：TcpClient 和 TcpListener。它们的使用简化了套接字编程的步骤。

TcpClient 类提供了一些方法来在面向连接的通信中完成客户端的连接。其构造函数有：

- 默认的 TcpClient()：在本地任意一个可用端口创建套接字；
- TcpClient(IPEndPoint ep)：在创建时将其绑定到一个本地终结点实例；
- TcpClient(string name, int port)：最常用的，直接指定远程主机和其端口号从而建立连接，而不需要使用 Connect()方法。

TcpClient 的实例建立连接时也使用 Connect()方法，不过这个方法不是提供指定 IPEndPoint 的实例，而是指定主机名和其端口号。对于数据的传输和发送，首先要使用 GetStream()方法创建一个 NetworkStream 实例用以在套接字上交换数据，然后使用标准的 Read()和 Write()在套接口上读取和写入数据。在关闭 TCP 连接并释放 TcpClient 关联的所有资源时，使用 Close()方法。

TcpListener 类简化了 TCP 连接服务器端程序的开发。两者构造函数的格式相似：

- TcpListener(int port)：绑定到指定的本地端口号；
- TcpListener(IPEndPoint ep)：绑定到指定的本地终结点；
- TcpListener(IPAddress ipa, int port)：绑定到指定的本地 IPAddress 和端口号。

方法 Start()类似 Socket 类中的 Bind()和 Listen()方法的合并，它将套接字绑定到 TcpListener 构造函数中指定的终结点，并将 TCP 端口置于监听模式等待接收连接请求。方法 AcceptTcpClient()类似 Socket 类中的 Accept()方法，接受连接请求并将该连接指派给一个 TcpClient 对象，这个 TcpClient 对象负责处理这个连接的数据交互，而 TcpListener 对象则用来接受其他的连接请求，最后调用 Stop()方法来关闭 TcpListener 对象。

第五章　总结和展望

5.1　总结

在分析了数据库管理系统的发展现状后，对于其日趋成熟的技术和市场的大量需求，得出了对数据库管理系统进行性能测试的必要性。在进行系统性能测试和国家软件评测体系的介绍后，选择了评测体系中基准测试项目的最符合我们数据库发展现状的基准 TPC-C。

然后对 TPC-C 提供的测试模型进行了详细的分析，深入理解 TPC-C 测试规范的内容。在此基础上，对测试系统 TpccTesting 进行设计，更加完善了从分析中得出的 TPC-C 测试的基本方法和具体流程。对测试系统进行了模块化设计，抽象出模型所需要的所有表的结构属性。

在实现中，介绍了模块和界面的实现，重点介绍了 TPC-C 模型设计的三个关键点：OLTP 应用环境模拟的实现，多用户并发访问的实现以及对数据库连接资源的维护—数据库连接池的管理。

5.2　展望

测试系统 TpccTesting 仅仅实现了对 SQL Server 2000 的性能测试，仍然有很多主要的数据库管理系统的测试接口未实现。

系统中对事务混合的管理仍有欠缺，不能完全符合 TPC-C 基准的要求。

测试结果生成，未考虑保存之前的结果。

未能实现图表化显示测试结果，以便更直观的分析系统性能。

测试数据加载时，对数据处理不够完善合理。

参考文献

[1] (印)罗摩克里希纳 (瑞典)耶尔克．数据库管理系统原理与设计[M], 120-159．

[2] YU Ge WANG,Guo-ren WANG,Xin-hui,ZHENG Huai-yuan. Performance Analysis of an Object-oriented Database System with TPC-C Benchmark[J].

[3] 唐晓东． 基于数据仓库的数据挖掘技术[M], 1999，216-248．

[4] Schult, W.Polze, A.Hasso-Plattner-Inst., Univ.Potsdam, Aspect-oriented programming with C# and .NET[J], 2002.

[5] 沈明星．TPC-C 与数据库性能优化[D]，2007.

[6] 马跃．基于 TPC-C 标准的数据库基准性能测试工具的研究和实现[D]，2006.

[7] 张维华．支持 OQL 查询优化的索引技术及基于 TPC-C 的性能评价[D]，2001, 12-48.

[8] 吕文达．精通 C#程序设计[M]，清华大学出版社，2004．

[9] 殷泰晖，张强，杨豹．C#编程从基础到实践[M]，2007，电子工业出版社．

[10]Wenguang Wang, Richard B. Bunt. Refence Behaviour of the TPC-C Benckmark [J]，2000．

[11]Judith A. Piantedosi, Archana S. Sathaye, D. John Shakshober. Performance Measurement of TruCluster Systems under the TPC-C Benchmar, 2005.

[12]TPC Benchmark C. http://www.tpc.org//tpcc.

[13]James Huddleston 等著．C#数据库设计入门经典[M]，清华大学出版社，2006，48-119．

[14]黄嘉辉著．C# .NET 网络程序设计[M]，科学出版社，85-131．

[15]Patrid O’Neil 著．数据库原理、编程与性能[M]，机械工业出版社，2002．

[16]Paul Nielsen 著．Microsoft SQL Server 2000 宝典[M]，中国铁道出版社，2005．

[17]Paul Dickinson 著．ADO.NET 高级编程[M]，中国电力出版社，2003．

[18]张晓辉等编著．SQL Server 2000 管理及应用系统开发[M]，人民邮电出版社，2003．

[19]杨用．DM3 的 TPC-C 性能测试研究[D]，华中科技大学，2002．

致　　谢

四年一个轮回，四年前，我们随着世界杯的落幕而走进我们的大学，而四年后的如今要迎来世界杯的时刻我们又要离开，满是不舍与依恋却注定要走。

毕业论文正代表着大学的终结，完成它既有一种收获感，又有一种失落感，可无论如何它代表着我四年的努力，代表了我四年的历程。当它终于完工的时候，我不禁想起了很多人，很多事，尤其是辛勤培养我的老师们，谢谢你们！

感谢我的论文导师张坤龙老师，感谢张老师在我的论文完成过程中给与的指导与帮助，让我得以顺利地完成论文。

感谢梁洪峻老师，他既是良师，又是益友，他所教会我的不仅是书本上的知识，还有着做人的原则与风骨。谢谢。

感谢传授我知识的每一位老师，马上就要走出校门，走上工作岗位，我将带着你们所传授的技能去打拼，去奋斗，多谢你们。

感谢我的父母，没有你们，就没有我的今天，你们的支持与鼓励，永远是支撑我前进的最大动力。

感谢身边所有的朋友与同学，谢谢你们四年来的关照与宽容，与你们一起走过的缤纷时代，将会是我一生最珍贵的回忆。

谢谢！