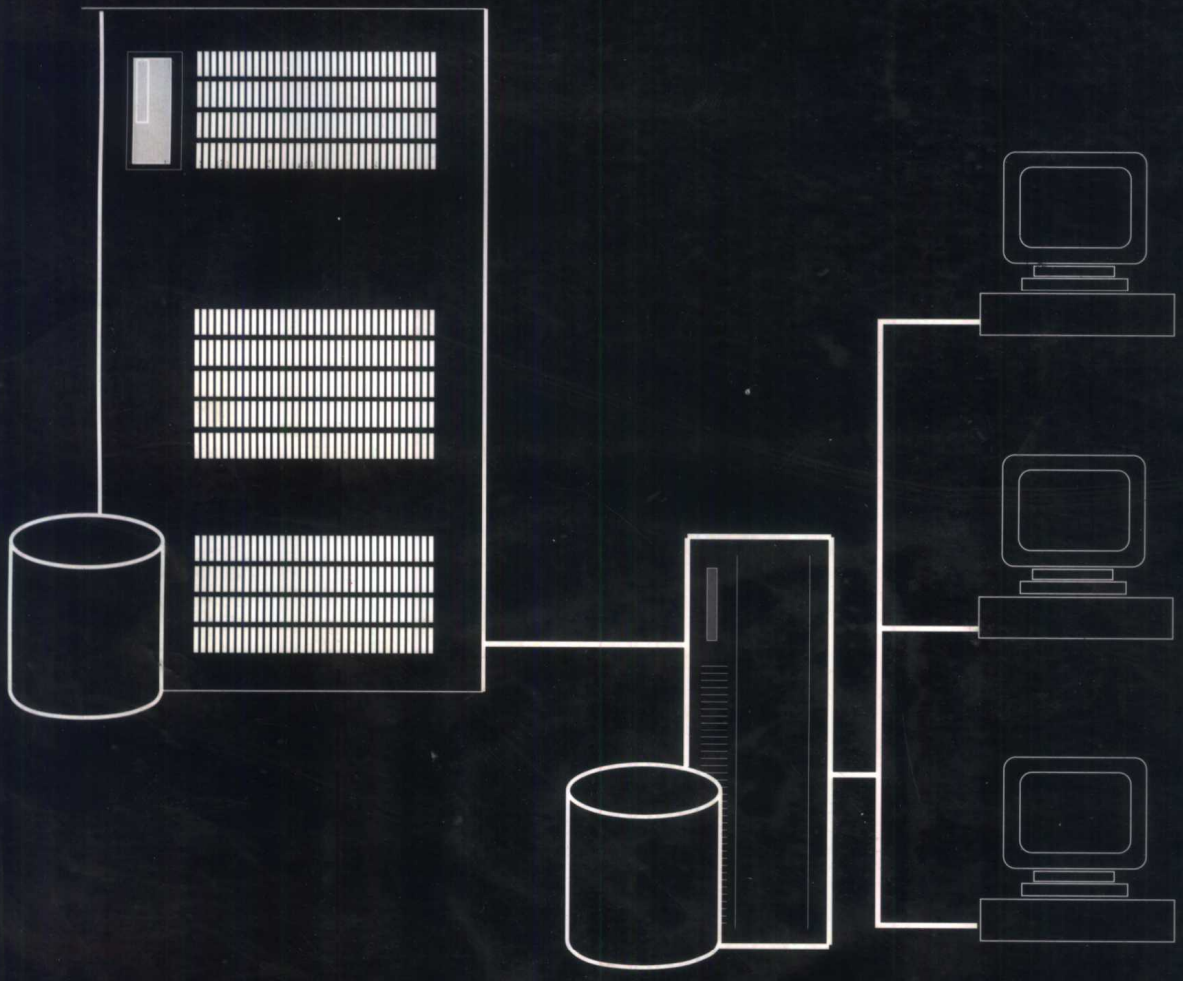
Revised Edition （修订版）

客户/服务器应用程序开发指南

Developing Client/Server Applications



WILEY

PHEI

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

W.H.Inmon 著

朱 莉 译

客户/服务器应用程序开发指南

Developing Client/Server Applications

(Revised Edition)

[美] W. H. Inmon 著

朱莉 译

电子工业出版社

内 容 提 要

客户/服务器环境是富有吸引力的。在客户/服务器环境下做开发，开发者可以彻底控制发生于客户/服务器系统中的开发与操作活动，并在硬软件和网络上花费很少的费用，但这决不意味着客户/服务器应用程序的开发可以草率行事。本书阐述客户/服务器环境的构造基础——它们是什么，它们如何实现，以及如果忽视了会发生什么后果。本书不是有关任何特殊的客户/服务器的技术，这里讨论的原理和最终结构适用于支持客户/服务器处理的所有技术。读完本书后，读者即可开发出结构完美、性能稳定的客户/服务器应用程序。

客户/服务器应用程序开发指南

[美] W. H. Inmon 著

朱莉 译

特约编辑：辛再甫

责任编辑：魏冬

*

电子工业出版社出版

北京市海淀区 173 信箱(100036)

电子工业出版社发行 各地新华书店经销

顺义县天竺颖华印刷厂印刷

*

开本：787×1092 毫米 1/16 印张：8.5 字数：198 千字

1995 年10月第 1 版 1995 年10月北京第 1 次印刷

印数：5000 册 定价：18.00 元

ISBN 7-5053-3257-0/TP・1208

0

前言

医生可以掩埋他的错误，但是建筑师只能建议他的客户栽培一些藤蔓来弥补他的过失。

——弗兰克·劳埃德·赖特

客户/服务器环境是富有吸引力的。与主计算机不同，在客户/服务器环境下开发者已经完全控制了发生于客户/服务器机器中的开发和操作活动。此外，由于花在硬件、软件和网络上的费用很少(相对于其他形式的处理)，客户/服务器处理中充满了漫不经心的态度。开发者因为完全控制了处理器以及开发与操作的费用，所以很容易在开发过程中草率行事。人们常常以一种懒散的、随便的方式为客户/服务器环境构造系统。但是如此的开发态度显然是危险的。任意的环境尺寸和规模以及对操作的各个方面的控制机会会导致混乱。简单地说，除非由结构原理作指导，否则在自治的客户/服务器环境中的设计者将制造混乱。环境越大，混乱就越厉害。就如同在主计算机环境中的开发要遵循一系列规则和原理一样，客户/服务器环境也应被控制。至少应受一些基本的原理控制。

本书阐述客户/服务器环境的构造基础——它们是什么，它们如何实现以及如果它们被忽略了会发生什么。这些原理组合在一起就形成了在所有客户/服务器环境中普遍应用的结构。本书不是有关任何特殊的客户/服务器的技术；在这里讨论的原理和最终结构适用于支持客户/服务器处理的所有技术。

本书的意图是给读者一些实际的解决方法。在这里很少提到理论问题。根据这本书，读者可准备构造非常好的、非常稳定的客户/服务器应用程序。下面是本书要讨论的一些内容：

- 客户/服务器环境中的性能
- 控制客户/服务器数据的主从关系的更新
- 在客户/服务器环境中操作处理和 DSS 处理的差别
- 应用程序的逐个开发与集中开发
- 客户/服务器处理和数据仓库
- 客户/服务器环境中的元数据

- 在客户/服务器环境中“需求驱动的开发”与“数据驱动的开发”
- 在客户/服务器环境中的 DSS 处理，以及其它

本书适用于开发人员、程序员、数据库设计者、管理者、数据库管理员、数据管理员和设计者。计算机的初学者也会发现本书大有裨益。

作者在此要感谢以下人士自始至终对本书的支持：

Platinum Technology 公司的 Sheryl Larsen

Knowledgeware 公司的 Mark Gordon

Chevron 公司的 Cheryl Estep

EPNG 公司的 Gary Noble

AGS Consulting 公司的 Bill Pomeroy

AMS 公司的 Patti Mann

Connect 公司的 Claudia Imhoff

独立顾问人 John Zachman

Storagetek 公司的 Sue Osterfelt

Prism Solutions 公司的 Ed Young

律师兼系统分析员 Ed Berkowitz

另外，感谢 Melba Novak 在办公事务上的支持。

目录

1

客户/服务器环境中的结构

设计一个事物常常是把它放在与它相关的更大的环境中考虑——一把椅子放在一间房子里，一间房子放在一幢屋子中，一幢屋子放在一个小区里，一个小区放在一个城市规划中。

——伊埃罗·萨里南

工人对建筑结构的定义是当铺设电源线时不必挖开马路。正确的建设结构是首先铺设电源线，然后再建马路。对结构最好的定义是准备好所有主要的需求，然后按一定顺序建立结构，使各部分以无破坏的方式配合在一起。

举一个反面的例子，就能很好地理解客户/服务器环境下的结构，此时事物并非以一种无破坏的方式构造。考虑以下事件的顺序，它们在客户/服务器环境下是完全可能的：

1. 在客户/服务器环境中的一个结点上建立一个小程序用于为一个称为“on-board”的汽车零部件数据库服务。
2. 重新设计此数据库使它包括国外的汽车零部件和国内的汽车零部件。
3. 重新构造代码使程序允许修改数据库，以及装载/访问数据库。
4. 加入数据来显示零部件流通——交易——以及在途的零部件。
5. 再次改变代码和修改数据从而允许从客户发出的请求通过服务器。
6. 为了能决定谁有改变数据的权力，加入一些代码并改变数据来实现“更新控制”的特性，等等。

在反复修改代码和数据之后，系统变得一片混乱。不参与需求分析——认为系统很简单因而不知不觉大量增加了复杂性——开发者就陷入了维护和操作的泥潭。仅因为客户/服务器系统比它们的主计算机兄弟们小，并不能意味着它们更容易建立和维护。从某种角度看，客户/服务器系统比它们的主计算机兄弟们更复杂。在主计算机环境中，对于它的所有费用和复杂性，有一些基本的服务自动地执行，而在客户/服务器环境中，这些不得不分别地做。由于这些原因，对客户/服务器的处理和开发做结构上的研究是非常重要的。什么是客户/服务器结构的关键因素？

1. 技术方面

- 网络中的处理器
- 网络中的数据
- 网络自身
- 程序——系统的和应用的

2. 费用
3. 使用方法
   - DSS 处理和协同处理
4. 自治与集中
5. 组织上的动态性
6. 数据结构

客户/服务器处理——基础

在它的最简单形式中，客户/服务器结构可被描绘成如图 1.1 所示。

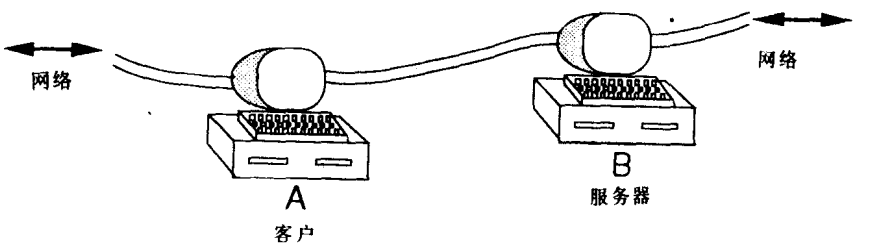


图 1.1　客户/服务器环境的简单视图

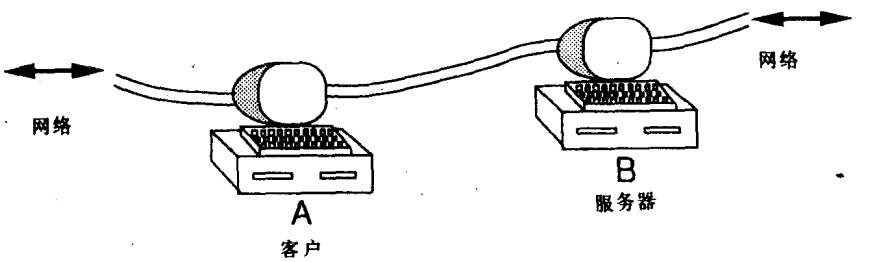


图 1.2　网络上的不同结点可由不同的处理器组成，例如，PC 和工作站

图 1.1 表示一个网络连接了两个 PC 或工作站(在网络中称为“结点”)。虽然其它的 PC 或工作站(即结点)没有画在图 1.1 中，它们显而易见也是通过网络的方法连接起来的。在图中的两个 PC 有一种特殊的关系。在 PC-B 上保持和管理的数据可通过 PC-B 的控制来由 PC-A 访问。PC-A 称作“客户”，PC-B 称为“服务器”。换句话说，PC-B 为 PC-A 的需求服务。这种简单的构造使在客户/服务器结构上建立的应用程序的复杂性降低很多。仅从技术观点看，图 1.1 所示的简单结构有许多变异。例如，图 1.2 表示了一个 PC 和一个大型工作

站连接在一起。

但是各结点之间在处理规模上的差异不是唯一的变化；另一个重要的变化是网络本身。图 1.3 表示了在网络中两个结点之间的间歇性连接。

一个网络同样也能与另一个网络连接，如图 1.4 所示。

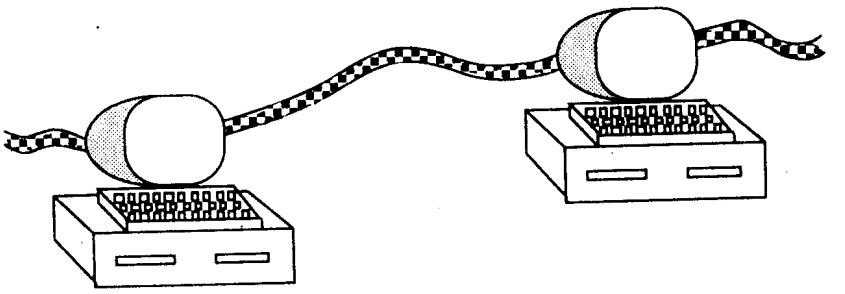


图 1.3　不仅各结点上的技术可以变化，而且技术自身也是基于多种多样的技术

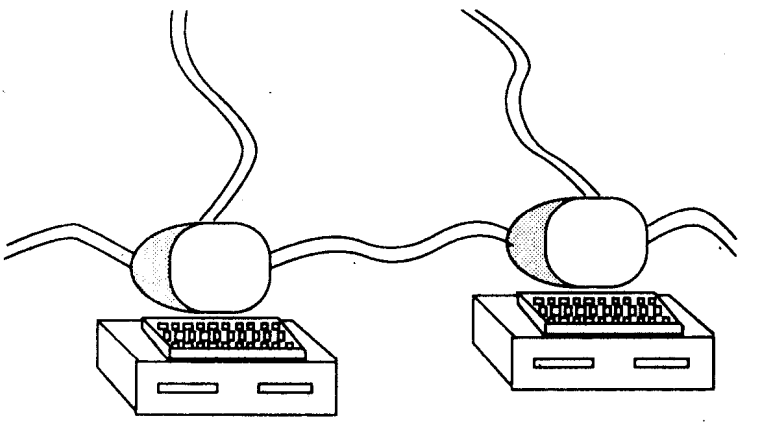


图 1.4　到不同网络的多重连接结点

网络中的各结点组织在一起形成“环”。环所包含的各结点共享同一个网络。图 1.5 表示了一个环。

在一个环内移动数据的速度(在正常环境下)是用毫秒来衡量的。

环与其它环之间的连接是通过桥。桥实际上就是一个设备，它控制一个环到另一个环之间的传输。图 1.6 所示为一个桥。

在客户/服务器环境中，结点和网络有很多种可行的排列。但是客户/服务器结构的主要问题并不是围绕着结点和网络的物理构造，而是围绕着发生于结构内的处理以及数据在网络中的存储和流动。将注意力集中在构成客户/服务器环境的技术上是件有意思的事情。当然开发者/设计者需要通晓客户/服务器环境涉及的技术，但是开发者应该注意的不是在技术上，而是在所要建造的应用程序上，以及更大的处理结构和数据结构上。

网络中每个结点内重要的软件部件的一个简单布局如图 1.7 所示。图 1.7 表示出每个

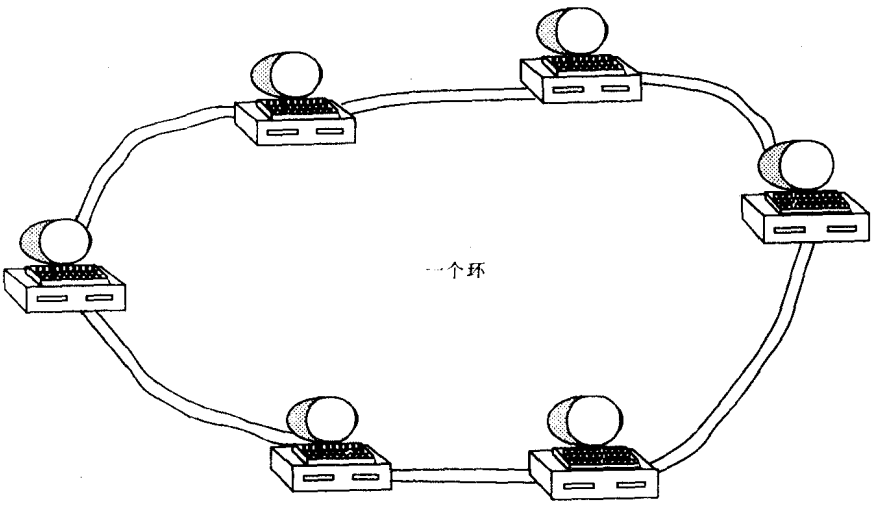


图 1.5 环

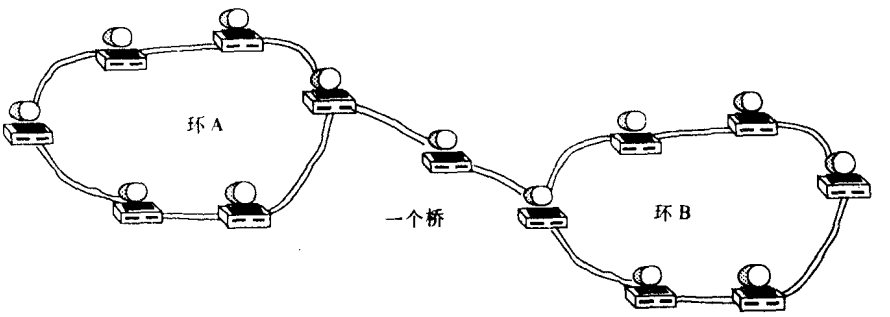


图 1.6 环 A 和 B 之间的桥

结点都有一个操作系统、应用程序、内部数据和外部数据。外部数据可存储在多种类型的媒介，例如磁带、软盘等上。(本书以下各图中的软盘符号将用于表示所有用于数据存储的外部媒介。)每个结点通过网络与其它的结点连接。结点内的通信由数据包的层间协议来实现。对比其它的结点，某些结点在某个或某些方面具有更强大的软件部件。但是所有结点共享通用特性。

一个有趣的问题是，客户/服务器结构与传统的集中式主计算机结构有何不同？集中式主计算机结构毕竟比客户/服务器处理存在的时间长。图 1.8 表示了从接收终端的角度来看传统的集中式主计算机的结构。在终端上做的工作很少。在集中式主计算机结构中的终端能做的所有事情就是显示预先制作的数据和预先格式化的数据及数据集。

对照图 1.8，图 1.9 给出了客户/服务器结构。

在图 1.9 中，原始数据被传递到一个结点。在到达此结点时，原始数据在显示之前通过了一系列的变化。在此结点，原始数据可被分析，或与其它数据结合在一起。在显示之前，原始数据可以多种方法格式化。但是在单纯的终端上接收数据并不仅是客户结点对数据所

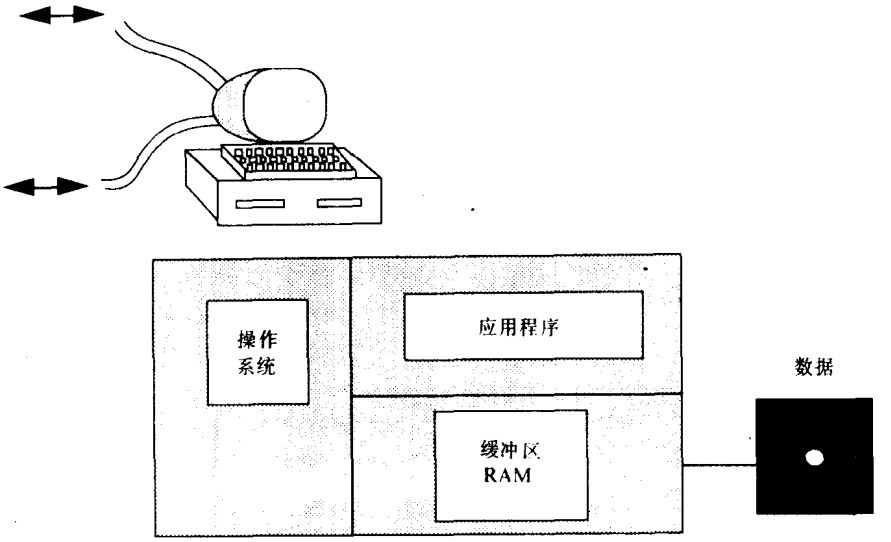


图 1.7　网络中每个结点的软件部件

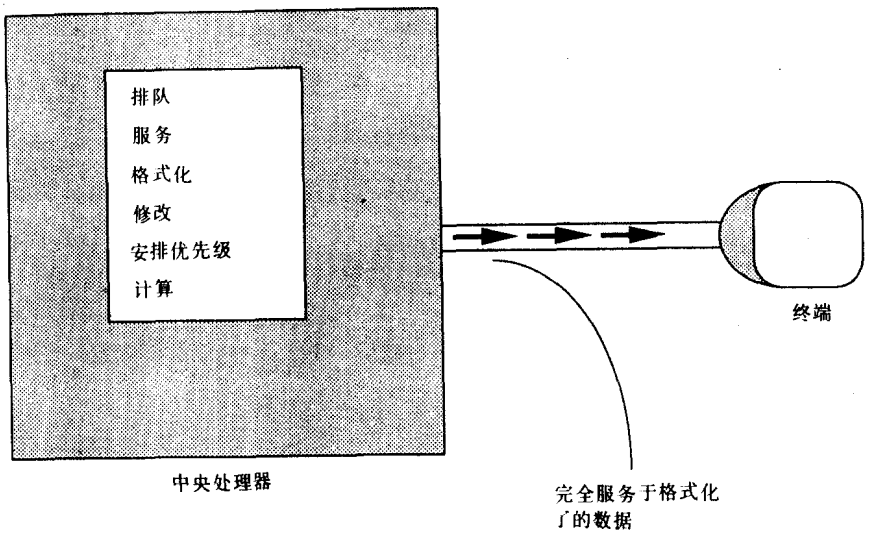


图 1.8　传统的集中式结构

能做的唯一事情。图 1.10 表示出与主计算机环境中的数据不同，数据一旦到达客户/服务器结构的结点，它可被以多种方法重新计算、重新格式化、重新分析，并且可与其它数据结合在一起而无需返回到服务器。

有时客户/服务器环境被称为是一个“聪明的”环境，因为在数据传递到客户结点之后，对数据会进行很多处理。在主计算机环境中，一旦数据到达终端，仅会对数据做很少的处理，如果有处理的话。由于这一原因，主计算机环境被称为是一个“愚蠢”的环境。

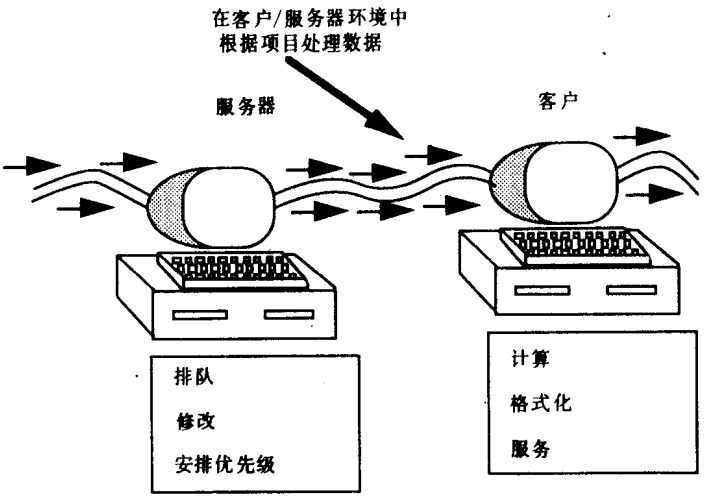


图 1.9　原始数据从服务器传递到客户。多数处理可在客户机处理器上运行。

费用

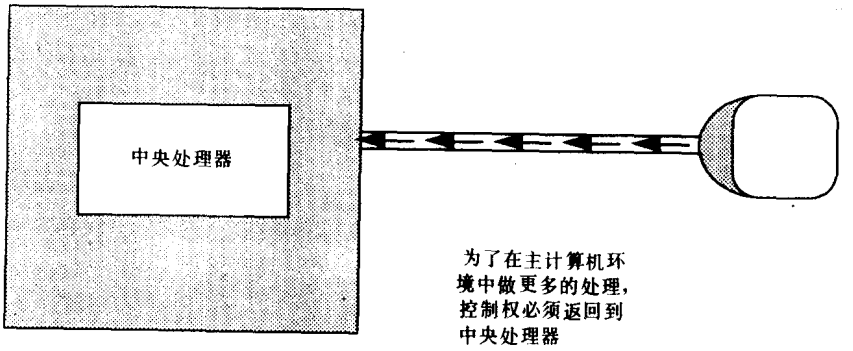
客户/服务器结构的第二个关键因素是费用。客户/服务器处理的全部花费要比大型主计算机处理器的相应费用要低。在费用上的差异如下：

- 处理器费用
- 网络费用
- 开发费用
- 操作费用
- 软件系统费用
- 维护费用

在这些费用中，仅最后一项在客户/服务器和其它环境中通常保持(或能够保持)为常量。客户/服务器在维护上的费用比它们的大型兄弟要少的原因是，客户/服务器的应用程序通常比较小和简单。但是，如果对客户/服务器的应用程序不做结构上的研究，则客户/服务器上的应用程序的维护，会变得与任何环境上的任何其它的应用程序的维护一样困难和昂贵。

应用程序的使用方法

客户/服务器结构的环境中第三个关键的因素是：客户/服务器的应用程序以哪种方法建立，一经建立，这些应用程序如何使用。客户/服务器处理的方法可分成协同的或 DSS 的。在方法上的差别也许不仅是在应用程序开发上意义最深远、最重要的因素，恐怕也是最易造



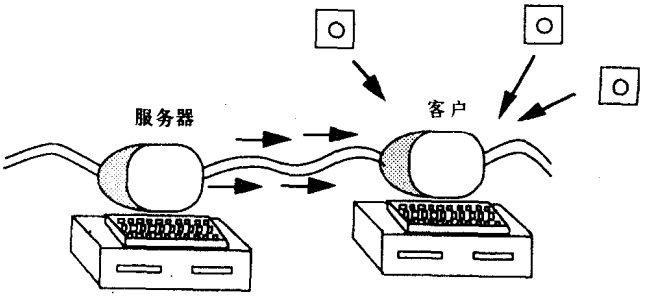


图 1.10　数据一旦到达客户方，它可被处理多次而不必将控制交回服务器

成误解和最易忽略的因素。

从什么是协同处理开始解释。协同处理是影响日常所做出的详细的事务决策的处理。这些事务如，保险公司有多少保险总额？在帐目中的最后一笔支出是哪个？借贷余额是多少？公司做出的对保险单的最后一次赔偿是哪个？等等。协同处理的一些显著特征是：

- 协同处理在当前值数据上操作——数据的准确度依赖于使用的当前时刻。
- 协同开发一般一次只做一个应用程序。
- 协同处理是需求驱动的。意思是首先明确一系列需求，然后产生所期望的应用程序。
- 协同处理以逐条记录的方式操纵——修改或访问——数据。
- 协同处理访问的归档数据是那些具有高访问可能性和相对低容量的数据。
- 数据所有权问题——谁可修改数据——是与操作环境密切相关的。
- 协同处理是为做迫在眉睫的日常决策的具体事务服务的。

DSS 处理(即，决策支持系统处理)与协同处理正好相反。DSS 处理用于支持管理工作，而不是具体事务。DSS 处理有下面这些特征：

- DSS 处理在归档数据上操作。归档数据一经正确登录就不能被修改。归档数据从根

本上与可修改的当前值数据不同。

- DSS 处理是在综合数据上操作，这与对单个应用程序的集中数据的操作相反。
- DSS 处理本质上是数据驱动的；协同处理是需求驱动的。
- DSS 处理以集合方式操作数据，而非以明细方式。
- DSS 处理可以访问那些不太具有访问可能性的归档数据。
- 谁可修改 DSS 数据不是问题，因为 DDS 数据一经写入就不能被修改。
- DSS 处理是为做长远决策的管理工作服务的。

以上特征证明了协同处理和 DSS 处理之间具有很大的差异。理解协同处理和 DSS 处理之间的差异是在客户/服务器环境中取得成功的基础。如果开发人员没有了解这些差异(或者更糟，开发人员根本没有意识到存在着问题)，结果会产生一个非常不和谐的客户/服务器环境。

在客户/服务器环境中协同处理和 DSS 处理之间最具深远意义，最明显的差异也许是在它们对硬件资源的使用上。协同处理和 DSS 处理展示了它们对硬件利用的非常不同的形式。图 1.11 给出了这些硬件利用的形式。

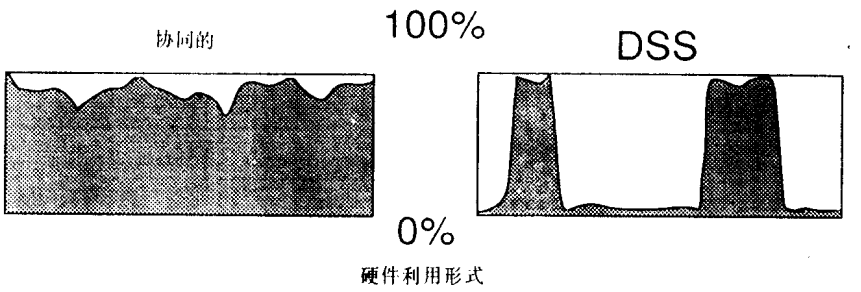


图 1.11　协同处理和 DSS 处理展示出极不同的硬件利用形式

图 1.11 表示出在协同环境中对客户/服务器硬件的使用是比较稳定的，只有微小的高峰和低谷。在每天工作结束时，使用协同数据，能够计算出对客户/服务器硬件的平均利用值，得到一个有意义的数值。

DSS 处理对硬件利用的形式有着根本的不同。DSS 处理的形式是一个二进制形式。在 DSS 客户/服务器处理过程中，或者硬件是被大量地使用，或者硬件根本不被使用。计算在 DSS 环境中客户/服务器硬件的利用会导致一个没有任何意义的数值。

如果与可利用的处理资源总数相比，在客户/服务器环境中所做的处理很少，使得协同环境和 DSS 环境之间对硬件利用的不同变得可以忽略，则二者的差异不会那么明显。确实，如果在客户/服务器网络上仅仅做少量的处理，并且只使用总硬件能力的百分之十或更少，则协同处理和 DDS 处理之间意义深远的差异就不能体现出来。

但是对于客户/服务器环境，如此小的硬件利用在经济上常常是不可行的。在客户/服务器环境中的重要的应用产品将使用足够的资源，这使客户/服务器环境的基础资源利用模型变得非常重要。当对硬件的利用增长时，协同处理和 DSS 处理在硬件利用之间的差异就变得重要而且显而易见。

DSS 与协同的差异和客户/服务器处理

在客户/服务器环境中，协同处理和 DSS 处理之间的差异对建立应用程序所应采取的方法具有深远的影响。协同处理和 DSS 处理应有逻辑的和物理的明确区分。混淆这两种类型的处理，无论是由于疏忽还是出于有意，将导致：

- 维护瓶颈
- 性能瓶颈
- 应用复杂性

在客户/服务器环境中区分协同处理和 DSS 处理的最好方法，是把协同处理孤立在一个环上，而把 DSS 处理孤立在另一个环上。

简言之，如此具有吸引力的客户/服务器环境如果在一开始被一个不知情的设计者否定，其主要的原因就是他随意地混淆了协同处理和 DSS 处理。

自治与集中

客户/服务器环境的另一个主要的结构因素是自治的处理与集中的、共同的处理。客户/服务器环境的主要需求之一是由每个独立的结点提供的自治处理，即独立的处理、独立的数据、独立的利用等等。数据一旦到达结点，用户即对此数据有了控制权，即使对预先打包的应用也一样。把预先打包的应用中的数据移动到结点上的数据私有文件中不会有什么危险。

在所有情况下，每个结点都需要某种程度的自由。但是，当数据或处理是“公用的”，它暂时存在于某结点，却需要在整个网络上传输时，对于这样的数据和处理有使之保持一致性的要求。例如，假设在网络上的一个结点是为汽车保险统计员使用的。保险统计员经常使用此结点做一些自己的计算。直到需要关于汽车数据的特殊分析时才会有人关心统计员在做什么。但是当统计员决定改变汽车保险的共同费用，包括承保类别、保险单的有效期、续保条项等等时，必须以“共同的”方式来改变费用以使在网络上所有受到影响的人都知道这一变化。在每个结点上都有私有的、自治的数据和公用的、共同的数据。对这两类数据不加区分和在结点内随意混用，将导致整个客户/服务器环境的混乱。

矩阵

建立一个矩阵来把不同类型的客户/服务器应用程序分类，如图 1.12 所示。

在客户/服务器环境中结点的自治应用程序是一个完全的自拥有程序。换句话说，结点的自治应用程序与网络上的其它结点之间没有数据存储或通信。