计算机网络基础与应用系列丛书

Testing Client/Server Systems

(美) Kelley C.Bourne 著
赵涛 张晓平 王虎东 等译

由经验丰富的专家撰写

客户机/服务器系统测试

适用于新一代客户机/服务器系统

机械工业出版社
CMP

计算机网络基础与应用系列丛书

客户机/服务器系统测试

(美) Kelley C.Bourne 著

赵 涛 张晓平 王虎东 等译

机 械 工 业 出 版 社

本书为适应新一代客户机/服务器系统的发展，详尽地阐述了对客户机/服务器系统测试的目的、内容、方法及注意事项。书中内容深入浅出，并注意理论与实例的结合。既是测试人员、开发人员和管理人员的必备宝典，又可供广大计算机爱好者参考，很有使用价值。

本书版权登记号：图字：01-98-0618

图书在版编目（CIP）数据

客户机/服务器系统测试/（美）伯恩（Bourne, K. C.）著；赵涛等译．－北京：机械工业出版社，1998

（计算机网络基础与应用系列丛书）

书名原文：Testing Client/Server Systems

ISBN 7-111-06284-1

Ⅰ.客… Ⅱ.①伯… ②赵… Ⅲ.计算机网络-网络服务器-测试-基本知识 Ⅳ.TP393

中国版本图书馆 CIP 数据核字（98）第 08180 号

出 版 人：马九荣（北京市百万庄大街 22 号 邮政编码 100037）

责任编辑：傅豫波 李 红

北京市南方印刷厂印刷·新华书店北京发行所发行

1998 年 5 月第 1 版第 1 次印刷

787mm×1092mm 1/16·21.25 印张

印数：0 001-5 000 册

定价：36.00 元

译 者 序

在当今的信息时代，计算机网络起着举足轻重的作用。而基于客户机/服务器模型的应用程序是网络应用的最基本和最广泛的形式。新一代的基于图形用户界面的客户机/服务器系统与传统系统有着诸多的差异，从而决定了我们对客户机/服务器系统的开发，包括系统的测试要有新的方法才能适应新的要求。只有在正确的方法指导下，一切开发活动才能有条不紊地高效地进行。作为商业化的系统，更是要高效、可靠，这更把系统的测试放在了关键的地位。

本书条理清楚地介绍了客户机/服务器系统与传统系统的不同，系统地阐述了整个测试过程中每种测试以及测试中每一步的目的、内容和方法。不仅给出了方法论，还会指导你一步步地完成整个测试实践。同时还分类介绍了处于市场领先地位的各方面的测试工具。最后本书还提醒你注意测试中常常会被忽略的地方。

总之，本书系统介绍了开发部门从了解客户机/服务器系统测试的特殊要求，选择合适的测试工具，进行测试实践，直到最终得到一个高效、可靠的系统的全过程。本书有很好的可读性，条理清晰，言简意赅，图文并茂，理论和实例相结合。它理应成为测试人员、开发人员和开发管理人员桌上的宝典。

在此，我们还要特别感谢参与了部分章节翻译的黄辉、梁岗、周晓勇。

因为急于把这本好书呈献给广大读者，加之能力有限，所以翻译中肯定会有不少错误，敬请读者指正。

译 者

1998 年 2 月于清华园

前　言

从大型机、普通文件、基于批处理的环境到客户机/服务器、图形用户界面（GUI）系统的变化，代表了软件开发领域的一个重大的转变。软件开发者和他们的管理者应该认识到伴随这个转变，测试领域也应该经历同样巨大的变化。用来测试基于过程和批处理的测试技术和工具，不能对客户机/服务器系统进行有效的测试。

为什么需要这本书：

这本书试图说明测试传统、基于过程的系统和客户机/服务器系统之间的差别。最显著的差别就是客户机/服务器系统几乎总同图形用户界面（GUI）相联系，而传统系统一般是基于字符或行的用户界面。这个已经很大的区别只是两种不同方法差别的冰山露出海面的一点点。其他同等重要的区别列在第一章中。

对于刚开始在客户机/服务器、GUI环境下进行软件开发的开发者，这本书可以作为他们关于测试这样一个有规划的、方法化的活动的首次介绍。对于其他人，它将主要是客户机/服务器形式的测试的一个介绍。无论是哪种情况，都希望读者能够更熟悉测试客户机/服务器系统的工具和技术。

一些专用软件工具被开发用来测试客户机/服务器系统。这本书中将介绍很多这种工具。我不准备专门介绍某种特定的工具和供应商。我将强调突出工具的主要种类并且给出每个类别的有代表性的例子。

这本书的读者：

以我之见，这本书的读者涉及到对开发客户机/服务器软件感兴趣的很多职业。开发者当然要熟悉这本书中的概念。

开发者的管理者对软件测试过程必须要有个很完全的把握。如果没有这样的把握，测试很可能就会不完全、不充分、低效率。

质量保证（QA）人员自然对完全并有效地测试客户机/服务器系统感兴趣。这本书中介绍的工具已成为（或很快会成为）测试领域的主流产品。

最后，但不是最次要的，用户团体必须以更高的层次参加到测试的工作中来。用户可能失去的最多。他们对系统不仅仅有金钱的付出。如果最终的系统效率很低且很难使用，真正着急的是用户自己。

这本书的内容：

第一部分给出了客户机/服务器系统测试的概述，尤其强调了与传统的基于过程的系统的差别以及为什么需要测试的新方法。这里简要介绍了测试的不同目标和自动测试工具的基础知识。

第二部分勾画了测试计划应何时和如何进行。这里列出并描述了不同种类的测试和在这方面能提供很大帮助的工具。这部分介绍了一旦发现和改正了错误后用什么样的办法和工具来处理它们以及怎样知道已经进行了足够的测试的讨论。

第三部分进入了测试过程的心脏。它描述了用于测试客户机/服务器系统的主要工具。

同时给出了着眼于领导市场潮流的工具的例子。

第四部分覆盖了客户机/服务器测试时所有经常被忽略的方面。在开发和实施系统的混乱中，很容易就忽略了这里列出的内容，这无疑给以后的错误敞开了大门。这部分涉及到的例子有测试 SQL 代码、安全性和系统性能。常常是整个系统已经开发和实施完毕时才有暇顾及这些问题。在项目的这个阶段再做改进是很昂贵、费时和狼狈的。

最后一章试图预测客户机/服务器系统和测试的未来。像在所有领域一样，预测客户机/服务器系统和测试也很冒险。其中某些趋势很明显。因特网和基于万维网的系统对于单位越来越必不可少。任何系统，包括客户机/服务器系统的更佳性能总是受人欢迎。将来的客户机/服务器系统会变得更复杂这个预言也没有问题。分布的和复制的数据库随着客户机/服务器系统对它们的访问变得常见。像一切预言一样，这章的做法也并不是板上钉钉，不过我很有信心，它们的绝大多数都会成真。

目　录

译者序
前言

第一部分　客户机/服务器测试概述

第 1 章　客户机/服务器系统与传统系统的差异 …… 1
1.1　图形用户界面 …… 1
1.2　事件驱动逻辑与过程式编程 …… 3
1.3　图形用户界面应用程序导航更加复杂 …… 4
1.4　继承 …… 4
1.5　普通文件与 RDBMS 的差别 …… 5
1.6　对象类库 …… 6
1.7　多文档界面（MDI）与单文档界面（SDI）应用程序 …… 7
1.8　分割处理 …… 8
1.9　快速应用程序开发（RAD） …… 9
1.10　不同的硬件 …… 9
1.11　多重软件供应商 …… 10
第 2 章　测试概念与目标 …… 12
2.1　测试概念 …… 12
2.2　测试目标 …… 28
2.3　达到测试目标的不同途径 …… 30
第 3 章　测试工具 …… 32
3.1　什么是测试工具 …… 32
3.2　客户机/服务器系统的测试工具用途概述 …… 32
3.3　使用测试工具的优点 …… 36
3.4　使用测试工具的缺点 …… 39
3.5　选择并获得测试工具 …… 41
3.6　测试工具矩阵 …… 45

第二部分　测试实践

第 4 章　测试计划 …… 47
4.1　测试阶段 …… 47
4.2　桌面检查、同级复查和代码检查 …… 60
4.3　测试人员 …… 64
4.4　测试度量 …… 67
4.5　图形用户界面的控件以及建议的测试 …… 68
4.6　快速应用开发如何影响测试 …… 71
4.7　开发一个测试计划 …… 73
4.8　通过平台测试 …… 74
4.9　强迫生成错误条件 …… 74
4.10　协定 …… 75
第 5 章　测试管理工具 …… 77
5.1　什么是测试管理 …… 77
5.2　测试管理工具提供的一般功能 …… 100
5.3　特殊测试管理工具提供的功能 …… 104
第 6 章　测试一个客户机/服务器系统的时限制 …… 106
6.1　测试工作应该什么时候开始 …… 106
6.2　测试资源 …… 109
6.3　测试活动有哪些，何时完成 …… 117
6.4　如何可以更精确地估计测试所需要的时间 …… 119
6.5　如果测试工作超期了该怎么办 …… 121
第 7 章　软件错误处理 …… 125
7.1　私有错误与公有错误 …… 125
7.2　错误跟踪 …… 126
7.3　通讯 …… 133
7.4　错误的纠正和分发 …… 136
7.5　错误的分析 …… 140
7.6　错误跟踪工具 …… 141
第 8 章　测试什么时候完成 …… 148
8.1　必须的测试有多少 …… 148
8.2　可靠的度量方法 …… 149
8.3　其他测试 …… 150
8.4　管理部门说测试已经完成了 …… 154

第三部分　测试工具

第 9 章　编辑器与调试器 …… 155

9.1 编辑器 …… 155
9.2 调试器 …… 164
9.3 调试时可用的其他工具 …… 178
第 10 章 捕获/回放测试工具 …… 181
10.1 捕获/回放的基础 …… 181
10.2 何时使用捕获/回放测试工具 …… 183
10.3 开发测试脚本的提示 …… 184
10.4 捕获/回放测试工具的缺陷 …… 186
10.5 一些特定的工具 …… 186
10.6 选择一个捕获/回放工具 …… 206
第 11 章 强度测试与载入测试工具 … 209
11.1 强度测试 …… 210
11.2 载入测试 …… 211
11.3 何时进行强度和载入测试 …… 212
11.4 调试由强度测试发现的错误 …… 215
11.5 硬件与软件工具 …… 215
11.6 选择自动测试工具的指导方针 …… 216
11.7 特定的工具 …… 219
第 12 章 向导 …… 231
12.1 什么是向导 …… 231
12.2 现实的局限性 …… 231
12.3 Mercury Interactive …… 231
12.4 Segue 的 GO! …… 241
12.5 在获得一个向导以前会问到的问题 …… 244
12.6 总结 …… 245

第四部分 客户机/服务器测试中被忽略的部分

第 13 章 SQL 测试与数据库 …… 247
13.1 测试 SQL …… 247
13.2 测试数据库 …… 266
13.3 数据字典 …… 272
第 14 章 安全系统 …… 274
14.1 对安全系统的需要 …… 274
14.2 客户机/服务器系统潜在的弱点 …… 278
14.3 其他破坏安全的方法 …… 291
14.4 其他安全测试 …… 294
14.5 用于加强或测试安全系统的第三方工具 …… 295
14.6 总结 …… 297
第 15 章 其他有关测试的问题 …… 298
15.1 批处理 …… 298
15.2 数据转换 …… 302
15.3 性能与优化 …… 307
15.4 代码管理与版本控制 …… 315
15.5 测试第三方的应用程序 …… 317
15.6 管理数据库 …… 318
15.7 构造测试数据库的工具 …… 320
15.8 日期与时间测试 …… 323
第 16 章 客户机/服务器系统及其测试的未来 …… 325
16.1 运行在 Internet 网或企业内部网上的基于万维网的应用程序 …… 325
16.2 性能 …… 326
16.3 单一的测试平台 …… 327
16.4 软件的完整生存周期 …… 327
16.5 质量保证小组的重要性 …… 327
16.6 多层的客户机/服务器系统 …… 328
16.7 日渐复杂的客户机/服务器系统 …… 328
16.8 对测试和错误的定义将会更广 …… 328
16.9 商品化的应用 …… 329
16.10 自动化的数据生成工具 …… 329
16.11 企业范围的错误及测试数据的资料档案库 …… 330
16.12 测试工具的改进 …… 330
16.13 更大的系统将要求更多的测试 …… 330
16.14 用户的经验将更成熟 …… 330
16.15 配置问题 …… 331
16.16 专业人员上哪去 …… 331

第一部分　客户机/服务器测试概述

第 1 章　客户机/服务器系统与传统系统的差异

一个“传统”的计算机系统由一台大型计算机和一定数量的“哑”终端组成。计算机系统所有的运算能力全在这一台计算机里。这部分硬件一般是一台主机。用户通过字符终端与计算机进行交互。通常，用户界面由输入的难懂的指令和接收的相对简单的屏幕信息组成。人们一般将在传统系统上运行的应用程序和像 CICS 和 COBOL 这样的语言联系起来。

一个客户机/服务器系统由一台或多台客户机通过网络连接到一台或多台服务器上组成。这个网络可以是一个局域网（LAN）或是一个广域网（WAN）。计算机的计算能力很大部分同时存在于系统的客户机和服务器层。用户通过图形用户界面（GUI）和系统进行交互。这个界面向用户提供了鼠标、键盘控制和对象，如下拉式列表框、按钮、单选按钮、菜单和图形。应用程序一般由如 PowerBuilder、Visual Basic、C++、和 Developer/2000 一类的工具编写。

1.1　图形用户界面

客户机/服务器系统和传统的、基于过程的系统最明显的不同是客户机/服务器系统几乎总是和图形用户界面相联系。传统系统有一个基于字符或行的用户界面。客户机/服务器体系结构在桌面层上提供了足够的运算能力来产生和处理一个图形界面。传统系统的终端通常被认为是“哑”的。他们没有独立的计算功能。没有这种能力，终端不可能依靠主机提供一个图形界面。

一个图形用户界面在屏幕上向用户描述了大量的对象（例如：控件）。菜单、命令按钮、图形按钮、下拉式列表框、工具条、网格、滚动条、图形、帮助屏幕和多文档界面表（MDI）只是用户可选的对象中的一部分。表 1-1 列出了流行的客户机/服务器开发工具中一般提供的对象。这种多样性与前一代只允许用户在提示符处输入数据或命令的系统形成了鲜明的对比。传统系统中的命令通常是缩写或难懂的指令。

表 1-1　一般对象类型

命令按钮
图形按钮
静态文本
数据窗口
网格
多行编辑区
图片
单行编辑区
编辑掩码

（续）

列表框
下拉式列表框
选中框
单选按钮
滚动条
菜单

用户非常欢迎这种图形对象与客户机/服务器的结合的存在和变化。图形对象的存在使应用程序使用起来更容易和直观。对新系统的培训花费和要求比传统系统明显低得多。不幸的是，受用户欢迎的部分给开发人员和测试人员增加了额外的测试负担。向用户发行产品前各个窗口中的每个对象都需要全面的测试。测试过程的复杂化是由于一个窗口中的每个对象可以（并通常会）影响或控制窗口中的其他对象。为测试一个窗口中对象所有的可能情况需付出的努力随对象数量呈几何级数增长。

图 1-1 和图 1-2 展示了一个窗口对象影响另一个窗口对象的小例子。单选按钮组可被用于决定向用户显示什么对象（标签、区域、按钮、菜单项）。选中一个按钮要求应用程序使一组标签、工具条按钮和菜单不可见的同时使另一组对象可见。图 1-1 展示了当第一个单选按钮被选中时用户的所见。图 1-2 是当第二个单选按钮被选中时屏幕的快照。上面的例子是相当简单的。关于不同对象相互影响更复杂的例子可能包括多窗口。比如说第一个窗口显示了一个数据窗口或有主记录的网格。双击一行可能会启动显示相关细节记录的第二个窗口。使这一切更复杂化，用户可以被允许修改或删除细节窗口中的内容，并且这些改变将会反回来影响主窗口。

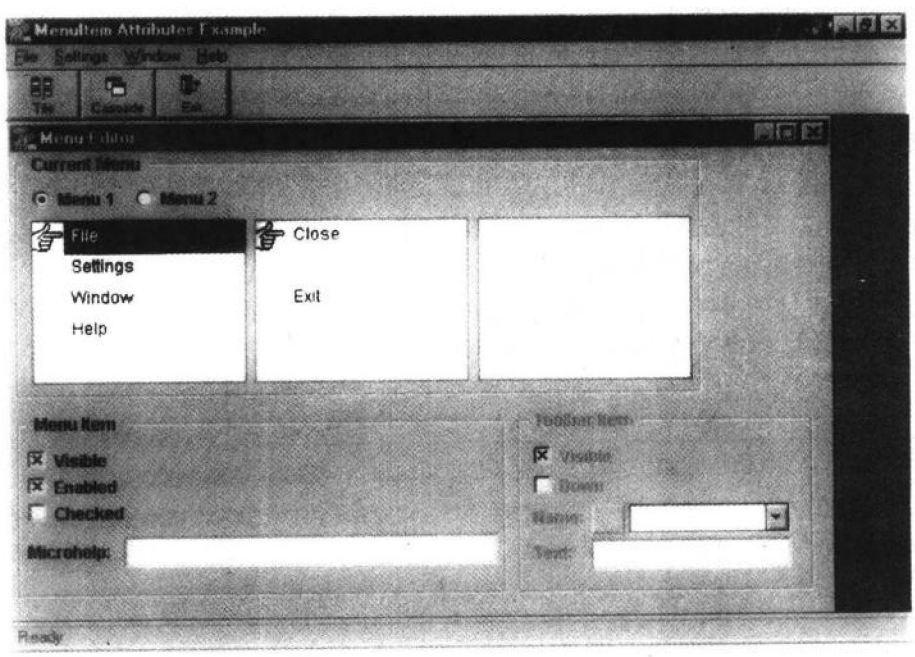

图 1-1　当第一个单选按钮被选中时的屏幕示例

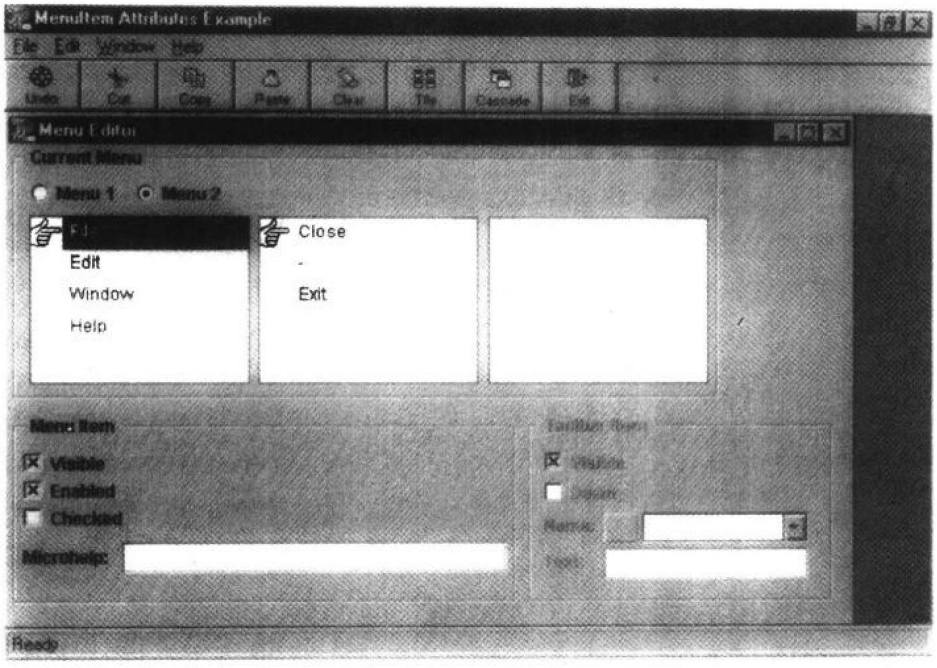

图 1-2　当第二个单选按钮被选中时的屏幕示例

就像这些简单例子所示，对象间的相互依赖关系可以迅速变得非常复杂。一个被设计和制造用来处理现实世界问题的客户机/服务器系统，将会包括上百、甚至上千这样的相互依赖关系。每一个这样的关系都必须被确认、说明和测试。不仅仅要做一次测试，而且每个对象要被重复测试以确定对象是否已经改变或是否要加入附加对象。

图形界面展示了客户机/服务器系统与传统系统的巨大差异，但只是两种不同系统结构实现对比的一小部分。以下将重点介绍其他显著的差别。

1.2　事件驱动逻辑与过程式编程

传统系统的编程语言和逻辑全是过程式的。一个过程式的语言是指计算机按程序源代码中指定的顺序执行语句。这种逻辑顺序只有当数据中的值引起不同的循环或控制顺序改变时才会发生变化。

客户机/服务器和图形用户界面系统不是过程式的，它们是事件驱动的。这意味着计算机针对发生的事件执行相应的程序。这里的事件是指用户采取的行动。像键盘活动、鼠标移动、鼠标击键动作和按键的动作，都是事件的例子。因为事件发生的顺序不可能预先知道，事件驱动系统相对来说更难测试。开发人员不可能知道用户下一次将要选中哪个按钮或菜单项。实际上，应用程序必须在任何时候对所有发生和可能发生的事件做好正确处理的准备。

显而易见，窗体中的每一个控件不一定会影响里面的其他控件。大部分标签或静态文本类型区对其他对象没有影响。数据条目区一般是被动的，但并不总是这样的。许多选中框只是用来表示一个值的开或关。

在潜在的许多被放在窗口中的对象和控件之外，还有大量的由自身所包含的控件所引起的事件存在。在 PowerBuilder 4.0 中，一个图形按钮共有 11 个标准事件，而一个窗体有 41

个标准事件。用户可自定义 PowerBuilder 关于每类对象的事件。Visual Baisc3.0 中的命令按钮有 12 个标准事件，而一个 Visual Baisc 窗体有 22 个标准事件。为有关每个窗体、对象、或控制的事件写的代码必须完全被测试。

对所有可能事件顺序组合的测试很快就变成了一个恶梦。简单的记录哪些顺序组合已经被测试而哪些没有被测试本身变成了极其繁琐的工作。对于有大量窗口和对象的系统，从统计上讲不可能测试全部组合。

使对于客户机/服务器系统的测试更为复杂化的是，一个简单的用户动作（比如：按下一个键）可以引发大量的事件。例如，一个 PowerBuilder 用户按下 Tab 键可以引起下列事件：

1）当前焦点对象的 Modified 事件。

2）当前焦点对象的 LoseFocus 事件。

3）得到焦点对象的 GetFocus 事件。

鼠标在一个区域的单击也可以产生同样的事件集。显而易见，每种不同的事件都需要被测试。

1.3　图形用户界面应用程序导航更加复杂

在客户机/服务器的图形用户界面应用程序中导航是极其繁杂的事情。这种繁杂并不是客户机/服务器或是图形用户界面规范造成的，而是给用户带来方便的新屏幕形式导致了这种情况。大多数图形用户界面开发包允许通过选择菜单项、单击命令按钮、双击表格的一行或单击工具条图标来引出一个屏幕。

导致繁杂的应用程序导航的原因经常是应用程序设计的不足。许多应用程序和它们的导航不是被特意设计，而只是自然成型。花在项目设计阶段的时间和精力的不足，不能归咎于客户机/服务器规范或是特定的图形用户界面工具。

应用程序导航的复杂性无疑给那些进行测试的勇敢的人们增加了额外的负担。应用程序中所有可能的导航路径都要被测试到。特别要注意测试屏幕间可能的依赖关系。测试还要深入进行以给应用程序以足够的强度。这样内存泄漏和资源限制才可能会被发现。

测试还要保证没有无意中引入了环路。有个环路的例子是：如果窗体“B”可以由窗体“A”来调用，窗体“C”可由“B”达到，如果窗体“C”允许用户调用窗体“A”，则一条环路被引入。在一个只有三个窗体的应用程序中，这个很容易被察觉。而在含有几十或几百个窗体的应用程序中，发现和改正它们非常困难。环路带来的问题是它耗费系统资源，而且会不可避免地引起一个全局保护模式错误（GPF）或者其他种类的崩溃。

1.4　继承

继承是面向对象程序设计（OOP）的一个主要原则。继承的定义是祖先的属性可以传给后代。后代中还可以定义新的属性。例如，祖先窗口包括“Add”、“Delete”、“Modify”按钮。从这个祖先继承功能的后代不用再为这些功能编写重复的代码。

继承的首要优点就是它带来的代码的高度可重用性，也就是软件开发中的“Holy Grail”。编写基本功能一次，然后继承这个功能，将大大减少要编写的代码量。除了只编码一次节约的时间和资源以外，有更少的代码要测试也是一个好处。更少的代码要测试意味着

投入测试过程的资源更少以及测试会更快完成。

继承的第二个好处是它有助于保持应用程序和应用程序套件中对象的外表和功能的一致性。这对于奋勇做战的开发者来说似乎不是一个很重要的好处，但这给用户培训带来了很大方便。

继承给测试客户机/服务器系统带来的影响是双重的。首先，被继承的对象被高度依赖，它们的可靠性和与要求的一致性必须做到毫无问题。这些对象要及早且彻底地测试。祖先对象在开发项目中很可能要尽早开发。在生成后代之前，祖先对象一定要花足够时间进行测试。

继承对测试的第二个影响是回归测试。回归测试是确保对已有代码的改变没有给原来正确的代码带来错误的测试。回归测试在一个使用继承的系统尤其关键。这是因为大量对象都依赖于少数的祖先。对任何祖先修改不正确将搞坏系统的一大部分。对祖先对象的任何修改或改进，不管它们看起来多小，一定要随之对所有后代进行严格的回归测试。

不是所有的图形用户界面开发环境都支持继承。就是支持继承，对它的实现程度也不相同。PowerBuilder 4 和 5 都支持开发者使用继承。可以使用继承的对象种类有窗口、菜单和用户对象。Delphi 中的可继承的对象类型和 PowerBuilder 的非常相似。Visual Basic 4.0 不支持开发者从一个祖先对象中继承功能。

1.5　普通文件与 RDBMS 的差别

在普通文件和关系型数据库系统间有几个重要的差别。大多数区别会影响到客户机/服务器测试如何进行。下面的部分中将讨论不同的数据存储技术间的一些差别。

1.5.1　如何访问它们

普通文件和关系型数据库管理系统（RDBMS）之间的一个显著的不同就是在普通文件系统中用户直接与数据交互，而在 RDBMS 中要通过一个数据库引擎。普通文件可以被打开、读、写和关闭。数据以一个个记录的形式从普通文件中读出。应用程序通过调用系统函数来实现这些。

想从关系型数据库中读数据的应用程序，首先要和数据库建立连接。访问关系型数据库最最常用的语言是结构化查询语言（SQL）。RDBMS 中的数据以集合形式被访问。

1.5.2　谁拥有它们

普通文件的所有者是这个磁盘文件的所有者的帐号。其他用户对这些文件可以有读和/或写的权限。其他的安全性粒度只能由应用程序自己实现。

一个帐号拥有关系型数据库。这个帐号经常是“dbo”或数据库所有者帐号。数据库中的多数表和其他对象也归 dbo 所有，但是用户个人也可以拥有表。每张表的所有者可以给其他用户这张表的访问权限。访问权限可以用读、插入、更新和删除权给出。一个客户机/服务器 RDBMS 的安全性可以完全脱离应用程序本身实现。

1.5.3　它们存储在哪里

普通文件存储在与大型机连着的磁盘驱动器上。应用程序一定要准确知道普通文件存储

的位置和它们的文件名。没有这些信息，程序就不可能访问这个普通文件。

关系型数据库也存储在磁盘驱动器上。通常这些驱动器与数据库服务器相连。关系型数据库可以被分别存在不同的服务器上。数据库的一半可以在 Omaha 城，而另一半可能在 Kansas 城。应用程序不需要知道数据库被分存在多个地方。

1.5.4　安全措施

传统系统的安全措施常常有由应用程序本身来实现。在客户机/服务器系统中，安全措施可以内置于数据库中，在应用层实现或通过两者的组合。通过把安全措施内置于数据库中更为有效和安全。这是客户机/服务器系统的标准。

1.5.5　数据依赖性与数据独立性

普通文件系统的一个应用程序需要知道很多有关数据是怎样在文件中存储的信息。它必须知道文件的名字和位置以及每一张表的记录布局。如果文件的名字或位置有变化，应用程序也要做相应的改变才能正确识别这些普通文件。如果普通文件中的记录布局有变，应用程序就需要好几种相应的变化。记录要读入的那部分空间的大小要重新设置以适合新的记录大小。为了与一个记录中的域的当前位置相对应，变量也要发生变化。使用普通文件的应用程序，高度依赖于普通文件的记录布局，这么说是非常准确的。

相反，通过关系型数据库来访问数据的应用程序有高度的数据独立性。它们使用的表可以从 Omaha 移动到 Kansas 城，而应用程序根本就不必知道。表内的列重新排列不会影响到应用程序。表中添加或删除列，应用程序也不用改变。甚至于列可以从一张表移动到另一张表里，可以通过使用视图根本不让用户知道。

1.6　对象类库

一个对象类库是一组经过测试的、可靠的对象。构成一个典型库的对象的例子有窗口、菜单和用户对象。这些对象能够通过继承，给应用程序提供快捷、一致、没有错误的成员。使用一大批可被继承的对象可以大大地加速一个客户机/服务器系统的开发过程。它使开发者更接近抓过来现有的软件部分就能组装成一个应用程序的梦想。

上面的话代表了理想的情况，面对大量开发项目的问题是如何达到这样的理想情况呢？获得和使用对象类库的两个选择是：1）自己开发；2）向软件供应商购买。通过很多软件开发的实践，两种选择都有它的长处和缺点。

购买对象类库的优点是它垂手可得、开发完全、测试充分、并有供应商的支持。上面每一点和自己开发相比都会有巨大的不同。购买对象类库意味着，当它买来装上后，你就可以立即学习并使用它了。你可以同自己开发一个实际的类库所用的时间比一比。第一步将会是确定这个系统中的通用需求成员。这些构成了类库中的初始成员。下一步将是实现、编码和彻底测试这些对象。建立之后，要给出所有对象的文档，以使得其他开发者知道它们的存在以及怎样使用它们。显然，这些步骤要比选择一个软件供应商，并向它订一个类库所花的时间长得多。

自己开发的类库的主要好处是它是符合定制的，而不是 off－the－rack 解决方案。它可以准确地专为你的需要进行开发，而那种“一种尺寸适合所有的人”的购买库，不会花时间

和精力开发你所不需要的功能。这里隐含的假设是开发者知道需要什么。这个听起来很平常，但是开发者在没有经历过几个客户机/服务器图形用户界面的开发项目之前，他们常常不能明确地知道应该做些什么。所以，在这个过程中肯定有很多修正要添加进去。

另外一个需要长期规划的领域是支持领域。自己开发类库的支持将会是个长期存在的任务。开发工具的新释放可能给对象库带来重大的改变。换到另外的操作系统（如从 Windows 3.11 到 Windows 95 或 Windows NT）的工作也不是自动进行的。第三方供应商将有希望用供应商提供的解决方案消除这些问题。

一个对象类库的花费也应该精打细算。类库所需的总花费要清楚地算出。应该计算类库的期望生命期的总花费，而不仅仅是获得它的费用。一个第三方产品的费用应包括初始注册费、每年的维护费和培训花费。培训花费要反映出上课的金钱和时间，还要包括开发者熟悉工具的时间。内部开发的花费应包括设计、建立、测试类库、建立文档和发行以及持续的支持所花费的资源。计算总费用的不精确很可能会导致不现实的期望和结果。

有关继承和祖先对象专题的一个题外话是：许多软件工作室选择购买由第三方开发商开发并销售的对象类库。尽管这样的类库中的对象已经经过了足够的测试，你还是应该在使用之前对它们进行测试。在第三方类库上花费大量的时间、培训和金钱后发现它没有被充分地测试而且含有很多错误的确会很烦人。这时你的选择应是希望那个供应商能改正他产品中的错误或重新换一个新类库。没有什么其他办法能帮助你赶上你的开发计划或者使你的管理人员和用户欢欣鼓舞。

1.7　多文档界面（MDI）与单文档界面（SDI）应用程序

在 Windows 3.11 环境下，允许开发者生成多文档界面（MDI）应用程序。本质上，这意味着一个应用程序中可以开多个窗口（叫做表单）。用户可以快速方便地切换于不同的表单间。表单之间也可以互相引用，允许一个表单中显示的数据影响到其他表单的数据。

单文档界面（SDI）应用程序中每一时刻只有一个窗口是活动的。用户不能同时处理多个文档。传统系统一般都是 SDI 的，而客户机/服务器图形用户界面应用程序更可能是 MDI。这主要是因为最近的开发工具（PowerBuilder、Visual Basic、Delphi 等等）大大简化了 MDI 应用程序的创建。另外一个促使 MDI 应用程序流行的原因是 Windows 的用户已经熟悉了 MDI 应用程序并且希望和需要这样的功能。

两种应用程序的例子都垂手可得。所有的主流字处理器程序（如 Microsoft Word、WordPerfect）都是 MDI 应用程序。绝大多数的电子表格（如 Lotus 1－2－3 和 Microsoft Excel）也是。用户可以在 Word 里打开多个文档并对它们独立操作。通过在“Windows”菜单下选择你需要的项就可以轻松地在它们之间切换。

Windows 环境下的 SDI 应用程序的例子有 Notepad 和 CardFile。Notepad 只允许你同一时刻打开一个文档。要打开另一个文档，就会把前一个关掉。CardFile 在只有一个数据文件可以被打开，这个方面和它类似，要打开第二个包含名字和地址的文件，第一个必须被关闭。

测试一个 MDI 应用程序比起测试 SDI 应用程序来要困难得多。原因是它允许用户打开多个文档，这给应用程序带来了很大压力。当多个表单活动时，资源的使用率大大增加。更高的资源使用率会引起一些在低资源使用率时不太可能被注意的问题。

另外，MDI 应用程序中的每一个表单本质上是一个不同的“线程”。众多线程的存在带来了应用程序把一个线程错当成另外一个线程的可能。这种可能在单线程的应用程序中根本不存在。强度测试必须进行以保证应用程序能够正确处理多于一个的线程。

1.8　分割处理

一个应用程序的“分割”是用来描述应用程序中的各个组成部分存在于哪里和在哪里执行。组成部分可能存在于客户端机器上或者一到多台服务器上。一共存在三种分割方法。图 1-3 以图形的方式表示了这三种不同的分割方法。每一种方法在下面论述。

在单层的方法中，用户界面（客户端）软件和数据库软件都存在于一台计算机上。这种方法并不代表真正的客户机/服务器体系结构。它用于单用户的单机应用程序。

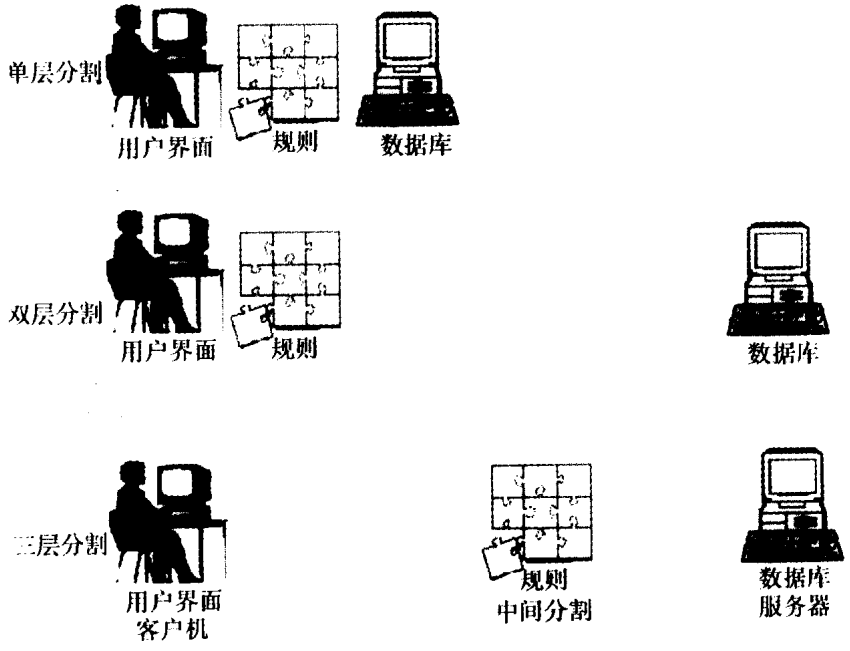

图 1-3　应用程序分割

双层的分割方法涉及到两台计算机：一台客户端机器和一台服务器。执行用户界面软件的客户端机器很可能是一台桌面 PC。客户端机器或许包括了应用程序的逻辑和业务规则。服务器储存数据并处理客户机的数据库请求。

三层的或 N 层的方法包括至少三台机器。客户端机器执行用户界面软件。象双层方法一样，数据库服务器存储数据并处理对它的数据库请求。第三台机器是为这个系统处理逻辑和业务规则的服务器。客户机和应用程序服务器都对数据库服务器进行数据请求。

分割应用程序的两个主要原因是 1）提高性能 2）增加灵活性。把系统的业务逻辑和表示逻辑分离兼有公布和清楚表述业务规则的效果。当规则被嵌入用户界面时就带来了忽略它们的趋势。把它们分离出来使设计者、用户和开发者都能清楚地表达业务逻辑。

开发多层结构的能力在客户机/服务器产品中变得越来越普遍。PowerBuilder 5.0 允许用户开发这样的应用程序。Oracle 的 Developer 2000 更是允许逻辑通过拖动方式从一层移动到另一层。Visual Basic 也提供掛钩来开发多层应用程序。在很短的时间之内分割应用程序的

能力将成为开发工具所期望具有的特征。

多层的结构能够大大改善客户机/服务器系统的性能。这个原因是系统中引入了额外的计算力量。更重要的是，把业务规则分离到它们自己的服务器上，使这些规则能更有效地实现。不同于把它们硬塞进表示层，它们的代码可以更有效地编写。

分割应用程序使得测试它更难。这是因为测试过程中包括了额外的机器层。不同层的开发工具和/或语言可能不同。这就要求开发者要学额外的语言技能。直到他们对这些语言很熟悉，他们的技能和准确性才有所保障。

现在，测试必须专门进行到它包含的每一层中的硬件和软件。很可能所选用的测试工具并不支持每种硬件平台和软件环境。这需要开发者和测试者掌握另外的测试产品。这也会降低劳动生产率和准确性。

分割系统的一个优点是如果决定了这样做会构成更好的实现，逻辑可以从一个部分移到另一个部分。不幸的是，移动逻辑意味着系统的大改动。当逻辑像这样移动时，要进行大量的回归测试。这个回归测试要覆盖重定位的逻辑和依赖于它们的所有函数。

1.9　快速应用程序开发（RAD）

快速应用程序开发（RAD）和协同应用程序开发（JAD）是由软件开发者和用户代表共同参加的开发的规范。RAD 的基本概念是开发者和用户共同设计系统中的屏幕。开发者迅速地把实现这些屏幕的最基本功能编好，然后把它们交给用户看。然后用户和开发者回顾这些屏幕以确认它们达到了用户的要求。这个周期一直继续到系统的基本部分定义完毕。一旦设计被用户接受，开发者将完成完全实现屏幕需要的代码。

RAD 方法有一些极其积极的方面。因为高效开发工具的使用，开发者能够非常迅速地设计出系统的基本屏幕。屏幕的“草图”在几分钟之内就能完成。系统的导航也能够很快地装配完毕。

RAD 允许用户在开发周期中很早就能见识到系统将来看起来怎么样。在传统开发项目中，长篇大论的说明写给用户看。不幸的是这些说明非常枯燥并且很难被最终的用户理解。经常是他们在不完全理解的情况下同意那些说明。

尽管 RAD 开发有很多优点，它也有严重的不足。一个主要的缺点是项目规划经常漏掉重要的测试阶段。测试，像在传统开发项目中一样，常常被忽视，并且给予很不现实的少量时间和资源。

RAD 对最少量测试的概念的另一个贡献是用户看到了应用程序的原型，但他们不能完全意识到原型并不是完整的系统。用户经常告诉开发者他们可以接受这样的原型。这句话有让开发者想赶快把东西交到用户手里之嫌，结果导致了最少量（或没有）的测试。

另一个 RAD 和传统软件开发项目之间的基本区别是：应用程序 RAD 系统是按阶段发布（relecse）的。传统项目一般一次发布，也叫“big bang”。分阶段释放的项目产生了大量的内部的发布或构件（build）。每一个发布必须在交给用户之前被构造和测试。

1.10　不同的硬件

传统系统由一致的硬件组成。它有可能由一台大型机和大量相同种类的哑终端构成。相反地，客户机/服务器系统由种类众多的不同硬件构成。数据库服务器可能是一台运行某个