Java Testing and Design

From Unit Testing to Automated Web Tests

Java 测试与设计

——从单元测试到自动 Web 测试

(美) Frank Cohen 著
周 磊 译

PEARSON Education
清华大学出版社

序　言

对每种软件的最终测试是为了让用户通过使用该产品能够实现某些实际目标。不论有多少新技术融入到软件设计当中，其成功的标准只能通过使用该软件的用户群体的规模大小进行评判。

Frank Cohen 熟知这一标准，他开创并且倡导了一个名为 TestMaker 的开源项目。该标准改变了成千上万名软件程序员、质量认证技术人员和信息技术管理员的日常生活。

自 1990 年以来，我的公司——Cooper 软件公司——已经成为关于交互设计的目标引导(Goal-Directed)方法的倡导者。这个强大且有效的方法在交互设计领域已经广为人们所接受。但当我看到 Frank 如何将这些方法应用到软件测试自动化和可扩展系统设计的时候，感到非常振奋。Frank 对这些方法的应用对于目前的开发环境，如 J2EE、.NET 以及用于构建信息系统的最新集成技术来说，是特别重要的。

本书介绍的软件测试新方法将向您说明如何创建以及应用高效、在线的目标引导设计和测试技术。根据用户的需求，并遵循 Frank 对可扩展系统设计的建议，您最终会得到用户能够满意的有效测试结果。

做为一名程序员、质量认证技术员或者 IT 管理员，您对系统设计、自动测试工具和相关技术可能会有多种选择。本书不仅列出了相关的方法学，而且还为您提供了一些有效的工具、技术和代码，它们能够自动测试 Web 使能(Web-enabled)应用程序，特别是 Web 服务应用程序。

Frank Cohen 在企业级软件设计领域有着 25 年以上的丰富经验。通过阅读本书，您将获取他在帮助用户实现其目标时所提供的丰富经验。

——Alan Cooper[1]
Cooper Software 公司主席

1　Alan 是 *About Face 2.0: The Essentials of Interaction Design* 和 *The Inmates Are Running the Asylum* 的作者、也被誉为 VB 之父。

前　言

现在您已经学习了程序设计、测试和应用软件开发的基本课程，并准备进行更深入的学习。虽然 Web 基础架构(在低廉设备组成的网格上使用开放协议的路由网络)无处不在，但在本书之前却没有任何参考资料可以用来指导您如何设计、编码并测试 Web 使能应用程序的可扩展性、性能和功能。本书将介绍一种快速有效的方法，该方法是从基本的 Java 知识通向构建具有产品价值(production-worthy)、可扩展性和高性能的 Web 使能应用程序的有效途径。

本书适合于为大型合作 IT 组、软件开发公司和服务提供商工作的软件开发人员、QA 技术员和 IT 管理员。通过从对体系结构的讨论到给出可以在用户环境下应用的源代码，本书对其他软件开发书籍进行了扩充。书中的案例分析演示了真实的应用技术，这些技术使得软件项目更具可靠性、可扩展性和安全性。

本书为软件开发人员提供了下一代 J2EE、.NET 和开源系统中所具有的新 API、协议和工具的详细清单。这些新软件库、工具和技术对于我们来说是一个重大的推动，它们将促使软件开发人员、QA 技术员和 IT 管理员学习更多的技术，从而使他们能够设计出复杂、多功能、可互操作的软件应用程序。

另外，本书介绍了一些下一代的网络协议，包括 Web 服务互操作性(Web Service Interoperability，WS-I)、安全声明标记语言(Security Assertion Markup Language，SAML)、使用可扩展标记语言(eXtensible Markup Language，ebXML)的电子商务，还有自由联盟(Liberty Alliance)协议。技术供应商提供的工具(例如，BEA WebLogic 服务器、IBM 的 WebSphere、Sun 原有的 Java 系统 Sun One、还有 Microsoft 的.NET)正在不断变化。所有这些革新都会促使软件开发人员、QA 技术员和 IT 管理员踌躇不前。在这些环境下，工具、协议、平台和技术所做出的每一种选择都会影响系统的可扩展性和可靠性。

本书是根据我的工作经验而编写出来的，我所工作的各个公司都曾测试并解决过一些与复杂的互操作信息系统相关的问题，尤其是 Web 服务这样的互操作信息系统。本书涵盖了一些宝贵的知识、提示信息和技术，并应用了 5 万多名从事 TestMarker 开源项目的软件开发人员、QA 技术员和 IT 管理员的可靠方法学。更多的细节可以通过站点 http://www.PushToTest.com 获取，有关软件设计、测试和自动化方面的新思想，每天都会在基于 Web(Web-based)的社区会议上进行交流。

本书描述了对建立关于 Java 的 Web 使能应用程序的体系结构选择，并且说明了每种选择对可扩展性和可靠性的影响。还说明了如何在具体情况下测试和优化这些系统。本书还描述了 Web 使能环境中智能测试代理的需求，并描述了一个测试代理框架，该框架具有一个关于最新的基于 Web 的测试技术的指南，本书还提供了 TestMaker 软件，该软件是我的免费开源框架，用于构建智能测试代理，以验证基于 Java 的 Web 软件的性能、可扩展性和可靠性。本书还给出了一些案例分析和简捷有效的代码，这些内容主要集中于 Elsevier Science、2Wire、Sun Microsystems 和 BEA 如何成功地使用智能测试代理技术在 Web 使能的 Java 项目中建立具有可扩展性的 Java 应用程序。

在本书中，您将看到对功能强大的 Web 使能 Java 体系结构、一些构造技术、简捷有效的

代码和一些智能测试代理的深层次讨论，这些技术可以验证 Web 使能应用程序的可扩展性、功能和可靠性。主要包括以下一些内容：

- Java 和基于 J2EE 的动态数据库驱动的 Web 使能体系结构
- 通过 SOAP、XML-RPC、.NET、HTTP、HTTPS 协议集成的应用程序
- J2EE 和.NET 的互操作性问题
- 为 BEA WebLogic、IBM WebSphere、SunONE 的开发者提供的性能工具包
- 电子商务(eCommerce)网站的体系结构和优化
- 使用公钥基础结构(PKI)、HTTPS、SSL 的安全 Internet 服务
- 下一代安全技术的使用：WS-I、SAML、Liberty
- 避免和解决并发问题
- 与 J2EE、Java Web Servcies、P2P、.NET 相关的体系结构、代码和测试代理
- 使用 Python/Jython 和 Java 的、易理解的测试脚本
- 包括电子邮件协议(IMAP、SMTP、POP3)应用的可扩展体系结构

PushToTest 网站为本书提供了一个活跃的用户社区。该网站包括了本书所有的源代码和应用程序，可以根据具体的需求扩展它们，以创建使用 Java 语言编写的、可靠的、可扩展的 Web 使能应用程序。

我衷心希望本书对您有所帮助。

——Frank Cohen

目　录

第 1 章　影响 Web 使能软件的因素 …… 1
1.1　软件开发的三个基本原理 …… 1
1.2　新领域——Web 使能应用程序 …… 2
1.3　目前编写高质量的软件为什么非常困难 …… 6
1.3.1　2.0 版本可靠性的神话 …… 6
1.3.2　对目标的管理请求 …… 7
1.3.3　在异构规则的情况下使用同构规则 …… 9
1.3.4　带有 bug 的语言 …… 10
1.3.5　邪恶孪生副总统问题 …… 10
1.4　软件开发简史 …… 11
1.4.1　Web 使能应用程序 …… 12
1.4.2　测试图和生命周期过程 …… 13
1.5　测试方法 …… 15
1.5.1　点击流测试 …… 15
1.5.2　单元测试 …… 16
1.5.3　功能系统测试 …… 17
1.5.4　可扩展性和性能测试 …… 18
1.5.5　服务质量测试 …… 18
1.6　定义测试代理 …… 18
1.7　为单个用户进行的测试 …… 21
1.8　创建智能测试代理 …… 23
1.9　本章小结 …… 24

第 2 章　应用程序的性能何时会带来问题 …… 25
2.1　什么是标准 …… 25
2.2　Web 性能的标准 …… 26
2.2.1　功能有效吗 …… 27
2.2.2　性能可以接受吗 …… 28
2.2.3　失效的频率是多少 …… 28
2.3　Web 使能应用程序的度量工具 …… 28
2.4　优秀性能的四种测试 …… 29
2.5　优秀测试代理的组成部分 …… 30
2.6　Web 使能应用程序的类型 …… 31
2.7　Web 使能应用程序点系统 …… 32
2.8　Web 使能应用程序的框架 …… 35

2.8.1 饼体系结构 35
2.8.2 采用饼体系结构和智能测试代理 36
2.9 在饼环境下建立智能测试代理 39
2.9.1 脚本语言和测试代理 40
2.9.2 产生有意义的数据 47
2.10 本章小结 47

第 3 章 模拟测试 48
3.1 为有意义的测试结果模拟用户行为 48
3.2 生命周期、项目和人的本性 49
3.3 Mircomax 生命周期 51
3.3.1 问题分类 51
3.3.2 问题排序 52
3.3.3 报告问题 53
3.3.4 评价问题的准则 53
3.4 Web 使能应用程序的测试 54
3.4.1 功能和可扩展性测试 54
3.4.2 功能性测试 55
3.4.3 可扩展性测试 56
3.4.4 测试功能和可扩展性的模块 57
3.4.5 管理风格 61
3.5 服务级别约定 62
3.6 网格计算和智能测试代理 64
3.6.1 简单安装、使用和维护的方法 64
3.6.2 自维护系统 65
3.7 理解性能和可扩展性准则 67
3.7.1 定义 SPC 67
3.7.2 运转中的 SPC 68
3.8 模拟用户目标 71
3.8.1 测试状态 72
3.8.2 使用 UML 和代码注释对测试进行建模 72
3.8.3 整体考虑测试 73
3.9 本章小结 74

第 4 章 Java 开发和测试自动化工具 75
4.1 3 次潮流 75
4.1.1 桌面应用程序的开发和测试自动化 75
4.1.2 客户/服务器应用程序的开发和测试自动化 77
4.1.3 Web 使能应用程序的开发和测试自动化 78
4.2 troika——第 4 次潮流 80
4.3 测试自动化生命周期 82

4.4 本章小结 …… 84

第 5 章 从方法学到设计的途径 …… 85
5.1 实现 troika 的工具 …… 85
5.2 怎样获得 TestMaker …… 88
5.3 五分钟学会 TestMaker …… 89
5.4 运行 TestMaker …… 90
5.4.1 熟悉 TestMaker 图形环境 …… 90
5.4.2 打开并运行测试代理 …… 91
5.4.3 通过新代理向导建立代理 …… 94
5.5 喜欢 Jython 的原因 …… 100
5.5.1 Jython 十分快速 …… 101
5.5.2 可以免费获得数据结构和列表迭代 …… 101
5.5.3 自动选择类型的动态变量 …… 101
5.5.4 函数是一流的 …… 102
5.5.5 Java 集成 …… 102
5.5.6 Bean 属性内省 …… 103
5.5.7 Sun 公司正在采用 Java 中的脚本 …… 103
5.6 使用 Jython 并入 Junit …… 103
5.6.1 JUnit 用于可重复测试 …… 103
5.6.2 一个 JUnit 示例 …… 104
5.6.3 JUnit 和 TestMaker …… 105
5.7 本章小结 …… 107

第 6 章 HTTP/HTML 环境中的设计与测试 …… 108
6.1 HTTP/HTML 环境 …… 108
6.2 HTTP/HTML 环境中的常见错误 …… 110
6.2.1 兼容性和浏览器缓存 …… 110
6.2.2 无效数据 …… 111
6.2.3 会话问题 …… 113
6.2.4 创建 HTTP 测试代理 …… 113
6.3 现有的 HTTP 通信 …… 115
6.3.1 理解 cookie、会话和重定向 …… 122
6.3.2 验证响应数据 …… 128
6.4 本章小结 …… 137

第 7 章 协调 SOAP 和 XML Web 服务 …… 138
7.1 Web 服务简介 …… 138
7.2 用于 Web 互操作的 XML-RPC …… 140
7.3 XML-RPC 的缺点 …… 144
7.4 SOAP 的全局互操作性 …… 144

7.5 Web 服务可扩展性技术 …… 145
7.6 Web 服务互操作性问题 …… 147
7.6.1 发现 …… 147
7.6.2 定义 …… 147
7.6.3 请求和响应 …… 148
7.6.4 展望 …… 150
7.7 利用 TestMaker 来理解 WSDL …… 150
7.7.1 构造 SOAP 调用 …… 154
7.7.2 不同类型的 SOAP 调用 …… 156
7.8 验证响应数据 …… 163
7.9 使编写代理更容易 …… 164
7.10 资源 …… 164
7.11 本章小结 …… 164

第 8 章 状态、设计和测试 …… 165
8.1 状态问题 …… 165
8.2 测试有状态系统的生命周期 …… 167
8.3 建立状态的技术 …… 168
8.4 准备和安装代理 …… 170
8.5 使用数据库配置测试 …… 180
8.6 使用 Lingo 使测试内容更接近真实 …… 184
8.7 本章小结 …… 188

第 9 章 集成.NET Web 服务 …… 189
9.1 互操作性和集成 …… 190
9.2 .NET 的不同之处 …… 192
9.2.1 文档样式的可扩展性 …… 194
9.2.2 .NET 中的 SOAP 头 …… 195
9.2.3 WSDL .NET 样式 …… 196
9.3 .NET 环境的测试代理 …… 198
9.4 近期需要考虑的事项 …… 204
9.5 本章小结 …… 205

第 10 章 创建与测试 Intranet 和安全环境 …… 206
10.1 基础知识 …… 206
10.1.1 路由安全 …… 206
10.1.2 虚拟专用网 …… 207
10.1.3 网络段和子网 …… 208
10.1.4 传输安全 …… 209
10.1.5 SSL 上的 SOAP …… 213
10.1.6 .NET 护照授权 …… 214

10.1.7 HTTP 基本授权 …… 214
10.2 SOAP 和安全 …… 215
10.3 产生认证和密钥库 …… 216
10.4 本章小结 …… 219

第 11 章 从构造到测试 Web 应用程序框架 …… 220
11.1 交易办公桌和智能测试代理 …… 220
11.2 可扩展的测试目标 …… 220
11.3 系统基础架构 …… 221
11.4 用户原型 …… 223
11.5 理解测试需求 …… 224
11.6 构造测试 …… 225
11.7 将用户原型实现为代码模块 …… 228
11.8 实现主组件 …… 235
11.8.1 安装 …… 235
11.8.2 运行 …… 237
11.8.3 清除 …… 238
11.8.4 用于测试配置的属性文件 …… 239
11.9 实现日志组件 …… 239
11.10 避免测试可扩展性问题 …… 240
11.11 本章小结 …… 243

第 12 章 将测试代理结果转换为行为知识 …… 244
12.1 分析测试结果得到的信息 …… 244
12.1.1 目标：我们的新 Web 站点需要处理 50 个并发用户的峰值负载 …… 245
12.1.2 目标：Web 站点注册页需要万无一失 …… 245
12.1.3 目标：顾客的月末报告请求不能减慢订购服务 …… 246
12.1.4 目标总结 …… 246
12.2 五大问题模式 …… 246
12.2.1 资源问题 …… 246
12.2.2 并发问题 …… 247
12.2.3 组件问题 …… 248
12.2.4 竞争问题 …… 248
12.2.5 崩溃的恢复问题 …… 249
12.3 结果分析中的主要因素 …… 249
12.4 结果数据产生误导的几种情形 …… 251
12.4.1 节点问题 …… 251
12.4.2 隐藏的错误 …… 251
12.4.3 Dileep 的困惑 …… 252
12.4.4 消除反复 …… 252

12.5 再论股票交易示例 …… 253
12.6 本章小结 …… 257

第 13 章 高容量数据中心的并发性和可扩展性 …… 258
13.1 简介 …… 258
13.2 2Wire 组件管理系统 …… 259
13.2.1 了解在网络上发生的事情 …… 260
13.2.2 复杂环境中的测试 …… 262
13.3 测试方法 …… 264
13.4 测试环境需要考虑的事项 …… 266
13.5 测试环境的约束 …… 267
13.6 本章小结 …… 268

第 14 章 为 SOAP 的可扩展性进行正确选择 …… 269
14.1 SOAP 流行的原因 …… 269
14.2 SOAP 编码类型 …… 270
14.3 简单对象访问需要简单的测试 …… 272
14.4 是否让开发工具来决定编码类型 …… 274
14.5 性能工具包 …… 275
14.5.1 如何获得性能工具包 …… 275
14.5.2 安装性能工具包 …… 276
14.5.3 开始安装 …… 276
14.6 其他事项 …… 285
14.7 本章小结 …… 286

第 15 章 电子邮件环境中的多协议测试 …… 287
15.1 需要和益处 …… 287
15.2 限定项目的范围 …… 287
15.3 测试设计和方法学 …… 289
15.4 安装和配置 …… 291
15.4.1 从原型到测试代理脚本 …… 292
15.4.2 创建 Muriel …… 292
15.4.3 创建 Marilyn …… 296
15.5 结果分析 …… 306
15.6 本章小结 …… 307

第 1 章　影响 Web 使能软件的因素

凭我的个人经验，除了编写代码的方式之外，还有其他因素也会影响 Web 使能应用程序的功能、可扩展性和性能。下面列出了在软件开发阶段中起到重要作用的 3 个因素：

- 在一个公司中，软件开发、质量认证和信息技术范围内的工作会影响软件的功能、可扩展性和性能。每一个独立的工作小组能够开发出高质量的、可互操作的 Java 代码。本书的第一部分将讨论对功能、可扩展性和性能有影响的组织与社会问题。然后您将了解到一些用于度量和提高可扩展性与性能的新过程和新技术。
- 所做的设计和体系结构的选择(服务器软件、协议、模式、平台、硬件、路由)对功能、可扩展性和性能也有影响。本书的第二部分将从 Java 开发人员的角度出发，描述隐藏在具有互操作性和可扩展性的 Web 使能应用程序之后的技术。
- 测试与监测系统、软件和组件的方式也影响着功能、可扩展性和性能。本书的第三部分将列出具体的案例分析和简捷有效的智能测试代理的代码，通过这些代码来检测 Web 使能应用程序的功能、可扩展性、性能和可靠性。

本章首先给出 Web 使能应用程序和 Web 基础架构(Web 使能应用程序在其上运行)的定义。然后，求解在许多公司中都会出现的棘手问题，这些问题使得人们难以通过 Java 提供互操作、高质量的 Web 使能应用程序。接着，本章简短介绍了近年来软件开发实践的发展史，从而说明了软件开发人员、QA 和 IT 管理员人员的重要性。之后，本章将说明在 Java 软件中影响功能、可扩展性、性能和可靠性的体系结构与软件开发的因素。最后还通过使用智能测试代理介绍了测试应用程序的新方法，这个新概念将贯穿于全书始末。

1.1　软件开发的三个基本原理

我还记得当我坐在我的第一台计算机前的情景。那是 1977 年，我还在读中学。我在键盘上输入了以下代码：

```
print "I hope Maggie Movius likes me."
```

BASIC 语言解释器执行上述代码并且在终端显示：

```
>I hope Maggie Movius likes me.
```

这简直就是个奇迹！它是如此简单、如此容易理解！但对没有使用过计算机的人很难作出什么解释。我那时就想象使用微机我能够实现什么呢？

时光流逝，我参与了组建软件开发社区的活动。该社区重点强调软件开发实践。软件由在计算机中可选的部分发展到成为美国经济的必要组成部分。

软件开发社区快速共享了关于如何建立软件的优秀思想，以及如何淘汰不良观点。例如，线程和流＝优良、专用通信协议＝劣质程序。

不久之后，我注意到更复杂的软件，例如 Web 服务器软件包，这是许多软件开发人员工作成果的共同结晶。在商业软件领域，软件开发小组意味着要支付薪水。通过薪水，才带来了软件产品生命周期、质量保障、版本控制、产品发布调度和代码评价(code reviews)。

我在一个商业软件公司的一个开发组工作，而现在要尽量给出一个简单的开发示例，我写出了下面的简单示例程序：

```
printf(" I hope Maggie Movius likes me.");
```

我编译上述代码并且在源代码控制系统中对其做了检测。大约一个小时以后，一名 QA 技术人员给我发来了一封电子邮件：

Frank：您好，很高兴与您相识。我看到了您的第一个评测。您应该知道您的代码缺少了本公司的版权号，并且还不符合本公司的编码标准，因为您用了 printf 而不是 printing，printing 才是本公司在标准库中的打印函数。

——Norman

这段经历教会了我软件开发的 3 个原理：

- 即使一个程序仅仅只有一行代码，也需要维护它。
- 软件的每个部分都应与其他软件交互。
- 提交(deliver)高质量的软件不仅仅是编写代码。

回到现在，在为 TestMaker 这个流行的免费开源测试自动工具设计其主要体系结构时，我发现这 3 个原理能够帮助我快速提交具有可互操作性、Web 使能的 Java 软件，并且还节省费用。在本书中，还将给出更多的设计原理以及目前用于创建、编码和测试高质量软件应用程序的实用技术，并根据具体环境提供测试代码。

1.2 新领域——Web 使能应用程序

软件开发过程需要经历开发和领悟阶段。在开发阶段，要考虑到每个因素。软件开发人员要考虑软件体系结构，尤其要考虑应用程序的业务逻辑和表示代码的位置。表示代码用于处理窗口、鼠标、键盘和其他用户交互指令。业务逻辑是定义应用程序的行为和操作的指令。在领悟阶段，软件开发人员要尽量广泛接受当前最优秀的实践经验。

图 1-1 解释了软件开发和领悟阶段的 3 代转变。第一代的软件体系结构在单一系统上建立了表示和业务逻辑。在第二代中，客户/服务器体系结构突破了由 60 年代大型机采用的占主流地位的庞大且集中的数据中心体系结构。在客户/服务器体系结构中，桌面系统只是将服务器提供的数据显示出来的“哑”终端。早期的 Web 都仿效了客户/服务器体系结构。浏览器将请求发送给服务器并且将接收到的 HTML 代码显示在屏幕上。

随着浏览器功能的改进——applets、JavaScript、ActiveX 组件和 DHTML 相应地出现——包括业务逻辑的一些系统都建立在桌面端。然而，主要的业务功能还是位于服务器端。

图 1-2 说明了我所期望的发展情况。“网格”使应用程序将业务逻辑模块部署到桌面端或者服务器端。通过使用 Web 服务和端到端(P2P)技术，各个模块可以相互发现。在这代体系结构中，业务逻辑模块的多个备份在数据中心的网格上运行，这样就能够实现失效恢复、动态路由和功能说明。这些运行在 Web 基础结构上的体系结构已经被很多公司采纳。在通向“网格”世界的过程中，Web 基础结构将会发展为对 Web 使能应用程序的支持。

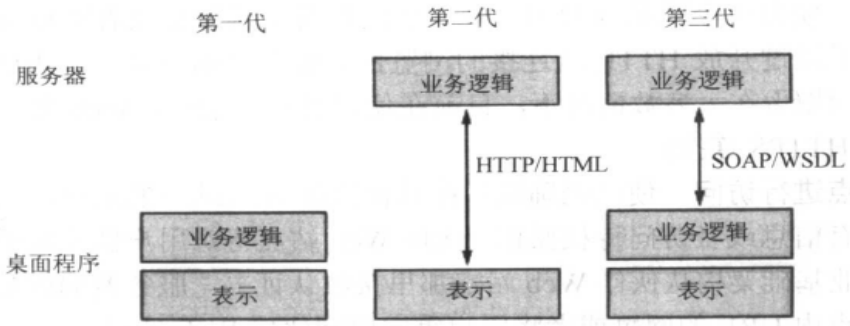

图 1-1　软件开发的前 3 代转变，从桌面端应用程序到客户/服务器，再到早期的 Web

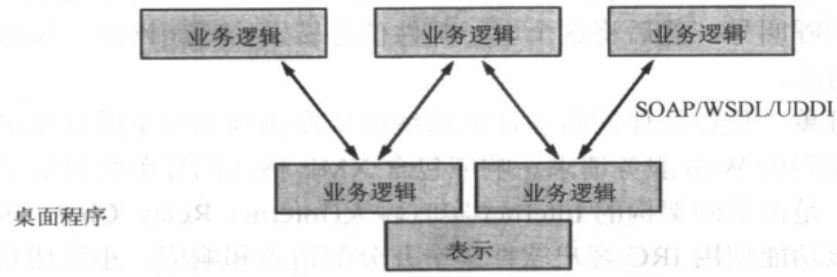

图 1-2　在现有的 Web 基础架构下使用 Web 服务和端到端(P2P)技术的下一代体系结构

Web 使能应用程序是由软件实现的互操作系统，它采用开放标准协议在 Web 基础架构之间进行通信。这些应用程序是从一些非常简单的术语发展起来的。浏览器向服务器发送需要获得文档的请求。每个人都很快意识到在网络和 Web 服务器上进行操作是多么的廉价。于是，当前每个人几乎都在使用 Web。到 2003 年，NetCraft 的 Web 服务器收到了 40 936 076 个站点的响应——这个数字仅仅是 Web 站点的数字。企业软件开发人员可以使用相同的 Web 基础架构开发数百个其他类型的应用程序。

请考虑目前典型的企业基础架构。公司所支持的许多应用程序在图 1-3 中作了说明。

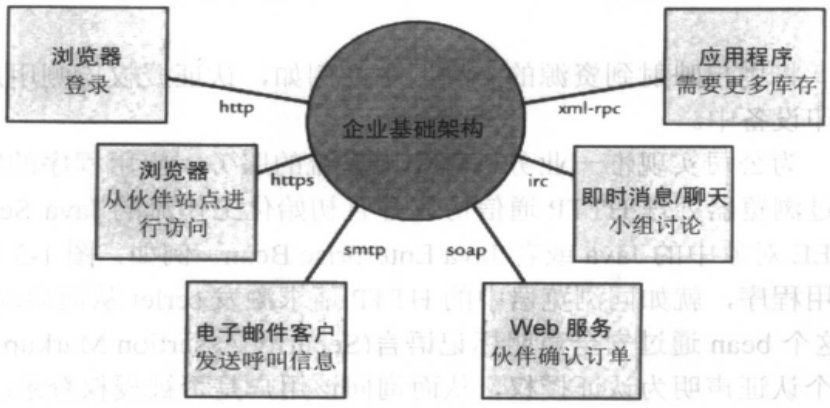

图 1-3　现代 Web 基础架构支持更广泛的 Web 使能应用程序和功能

图 1-3 中说明的功能是在当前典型公司中使用 Web 基础架构所提供的服务。下面详细地表述了这些服务：

- **登录功能** 使得单个职员或者用户确认他们获得私有信息或者特权操作的权限。这个功能使用了通过开放 HTTP 所连接的浏览器。服务器响应包含表明成功或者失败的简单 HTML 代码(在大多数情况下，目前在使用登录功能时，Web 使能应用程序采用的是安全的 HTTPS 连接)。
- **从伙伴站点进行访问** 使已经确认却在其他伙伴 Web 站点的用户不需要再次登录就可以查看私有信息或者访问特权操作。伙伴 Web 站点将对用户做出接受或者否认的认证服务。企业基础架构从伙伴 Web 站点那里接收认证书。服务器响应是 HTTP 网页重定向到该站点内 URL 的网页或者将用户重定向回伙伴 Web 站点。
- **发送呼叫信息** 使得电子邮件直接转发到职员的呼叫机。通过这个功能，企业基础架构接收到标准的 Internet 电子邮件服务。当它收到一个电子邮件信息时，将从目录中查询该职员的呼叫号，然后将这个电子邮件信息转发给呼叫网络。该服务将发送确认的电子邮件信息。
- **伙伴确认订单** 使得伙伴的自动订单系统确认在执行系统中的订单的细节。其请求来自 SOAP 编码的 Web 服务请求。响应包含 XML 格式的订单交付信息。
- **小组讨论** 是由基础架构的 Internet 实时聊天(Internet Relay Chat，IRC)服务推动发展起来的。该功能使用 IRC 客户软件集中并分配请求和响应。小组成员之间的通信包括有价值的知识，因此需要记录 IRC 信息流，为搜索引擎服务建立索引，并且要通过 Web 浏览器接口显示出来。
- **需要更多的库存** 使得基于非浏览器的应用程序在已脱销的情况下与订单服务进行通信。设想一个像沃尔玛(Wal-Mart)的超市，它在货价上内置了很多传感器，因而能够发现库存不足的情况。这个功能采用 XML-RPC 协议请求更多的库存，以便给订单和库存服务发送一个 XML 编码的基于 HTTP 协议之上的请求。响应中包含一个确认号。

本书将说明如何建立并测试提供这些服务的系统。另外，本书还提供在相同 Web 基础架构上运行的下一代服务和协议。在具体讲解之前，先来看一下图 1-4 中所示的处理这些功能的后端系统到底发生了什么情况。

在图 1-3 中，使得企业基础架构能够提供上述所有服务的后端系统，也使用了标准的 Web 技术，其中包括：

- **认证授权** 将用户映射到资源的一种服务。例如，认证授权检测用户是否可以将文档写入到打印设备中。
- **应用程序** 为公司实现惟一业务进程和工作流的服务。应用程序的实现通常包括两个部分：通过浏览器处理 HTTP 通信协议并且初始化工作流的 Java Servlet，还有实现工作流的 J2EE 对象中的 Java 或者 Java Enterprise Bean。例如，图 1-3 中的登录请求发送给一个应用程序，就如同浏览器中的 HTTP 请求激发 serlet 从而启动 Java Bean 中的登录处理。这个 bean 通过安全声明标记语言(Security Assertion Markup Language，SAML)协议将一个认证声明为认证授权，从而询问该用户是否被授权登录(当然，应用程序服务并不局限于 Java 语言。在实现应用程序的业务逻辑时，开发人员对操作系统和平台有很多选择，包括 Java、.NET、C 和 C++)。
- **网络操作系统(Network Operating System，NOS)** 控制网络层对设备和服务的访问。NOS 通常是 Windows、Unix、Linux、Macintosh 或者其他操作系统，它为网络资源、硬件资源和简单目录服务提供软件接口。

- **即时消息服务器(IM)**　为 Internet 实时聊天(IRC)服务提供协议一级的支持。从图 1-3 的小组讨论示例中，可以看到开发人员小组使用 IRC 进行通信并且协同开发一个新的软件模块。对话中包含了公司的内部信息。加入 IRC 聊天室的请求将初始化一个工作流，该工作流使用轻量级目录访问协议(Lightweight Directory Access Protocol，LDAP)确保用户在网络操作系统目录之中。

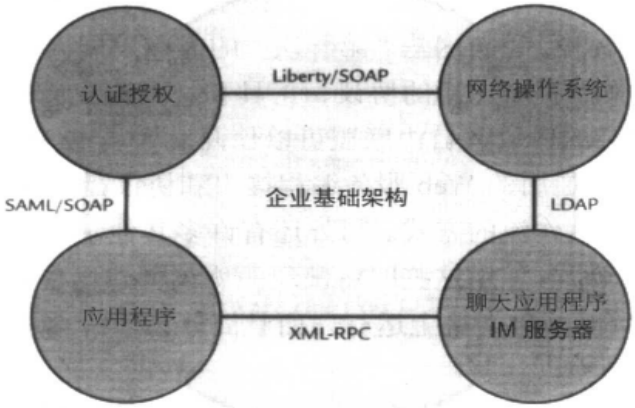

图 1-4　后端系统使用相同的 Web 基础架构连接多个不同系统发出不同请求的应用程序

图 1-4 所描述的系统和协议并不完全。实际上，一般企业中的后端系统都会有其他许多的系统和协议。将这些系统绑定并且将最广泛的用户粘连在一起就构成了一个 Web 基础架构。

是什么因素驱动我们在 Web 基础架构上进行这些开发的呢？以我个人的观点，我认为我们生活在“第一次(firsts)”的时代。考虑下列因素：

- 在信息技术(Information Technology，IT)产业中，软件开发人员、QA 技术人员和 IT 管理员第一次开始认同可互操作的体系结构。我们可以假定每个公司都已经安装了 Web 基础架构。
- 软件开发人员、QA 技术人员和 IT 管理员第一次开始认同需要联合制定安全标准。我们为之工作的公司或者我们所加入的小组目前都与其他公司或者小组建立了合作关系。相应地，我们的系统也需要共享认证、身份管理和目录服务。我们需要提供 Web 站点的联邦，当用户登录到联邦中的一个站点时，则它同时也登录了联邦中所有其他站点。
- 软件开发人员、QA 技术人员和 IT 管理员第一次开始认同在每个应用程序中可扩展性是实现目标的关键所在。从一个公司中的单一职能到整个世界的经济运转，一切都要依靠 Web 使能应用程序。当这些应用程序无法运转、或者运转得很慢时，一切都将停滞不前。
- 政府规章第一次成为软件应用需求的一部分。例如，美国国会在 1996 年通过了健康保险携带和说明法令(Health Insurance Portability and Accountability Act，HIPAA)。HIPAA 给美国的每个健康组织带来了重大的影响。HIPAA 对病人数据的访问有着严格的控制，并且要求这些数据能根据病人在不同健康供应者之间的所在情况而作出相应的改动。HIPAA 是在未来政府规章中所添加的信息系统最佳实践方案的先驱。
- 互操作工具和技术第一次与集成开发环境(Integrated Development Environment，IDE)工具相结合。过去，工程师如果想要使用 CORBA，他将不得不使用外部的 CORBA 工

具集定义接口。而现在，大多数流行的 IDE 环境中都包含 CORBA、Web 服务、RMI、COM+、消息队列和数据库/数据访问工具。

- 最后，Java、其他互操作开发环境和开源项目第一次将对操作系统和平台的依赖因素降低到几乎不必考虑的地步。在一个 Web 使能应用程序的环境下，所有的操作系统和平台都几乎具有相同的特征。软件开发人员建议操作系统和平台供应商提供相应的产品、工具和技术，而这些早先只是提供给系统管理员和 IT 管理员的。

我们感觉到计算机软件业已经向这些“第一次”的方向发展。不必再为软件开发人员在分布式、互操作的系统下工作而担心。新的协议、工具和技术给我们注入了新鲜的血液。例如，SAML 定义了简化单一登录的协议，自由联盟可以使开发人员在 Web 使能应用程序中建立联合安全模型，WS-I 标准化了通用的 Web 服务编程接口和协议，BPEL4WS 在 Web 服务应用程序中定义了表示和实现业务工作流的方式。另外还有许多其他示例。本书的第二部分列出了许多这样的新协议并且为您演示了使用这些即将出现的协议创建优良设计和测试策略的方法。所有这些新协议都可以在 Web 基础架构上运行，而且简单易学。本书中所涉及的内容将给整个世界带来更高的生产力和效率。

1.3 节将讨论在许多公司中出现的棘手的行为问题，这些问题使得在 Java 中传递互操作、高质量的 Web 使能应用程序变得非常困难。

1.3 目前编写高质量的软件为什么非常困难

计算机世界为软件开发工具、技术、硬件和知识提供了最广泛的选择，从而可以建立具有互操作性、高质量的 Java 软件应用程序。然而，用户难以通过现有的应用程序顺利完成任务，导致一种失败感。但是，作为 Java 软件开发人员，我们可以改进我们所编写的软件代码，加入更大商业组织对用户实现他们的目标，避免他们的挫折感也有等效的影响。我敢肯定您一定看过或者经历过下面 5 个影响软件开发因素中的一个或者几个：

- 2.0 版本可靠性的神话
- 对目标的管理请求
- 在异构规则的情况下使用同构规则
- 带有 bug 的语言
- 邪恶孪生副总统问题(the evil twin vice presidents problem)

上述每个问题对软件开发工作都有一定的影响，但它与设计或者编写代码却是分开的。下面逐个分析每个问题。

1.3.1 2.0 版本可靠性的神话

在您赶制一个庞大的新软件版本时，您遇到过这个问题吗？您寄希望下一个主要版本将会很稳定、运行可靠、修改了所有存在的问题并且为用户提供成功的产品。另外，您预计新软件将证明符合您的管理要求，这会给您带来升职并且会得到一个假期。但是，请回到现实！

编写代码相当困难。在现代 Java 开发工作中，您的代码模块构成了一个更大的系统，而且其他模块随时都可能发生变动。您的代码处在一切都是变动目标的环境之中。这使得您需要

学习新的知识，而且您可能不喜欢这种情况：其他开发人员就坐在您的眼皮底下，而您的模块却要依赖他们的模块！

在软件开发中，一直存在着这样的问题，不论作出许多细小的改动还是进行几处大的改动。灵活的编程技术通过一次重大的检查就可以提供许多小的实现。相信 2.0 版本可靠性的管理会选择这次重大的检查。以我的经验，如果没有考虑周全，选择这次重大检查将会是失败的。不够优秀的开发小组经常迫于完成“重大版本”的压力，但这并不值得。完成的软件一定存在需要立刻解决的错误和问题，但是软件开发人员却需要几周时间重新集结共同处理其维护的版本。如果他们没有得到“放松时间”而擅自离岗或者做些不该做的事情，就应该解雇他们。

最佳的软件开发采用“波浪式”开发。当软件开发人员获得灵感的时候，他快速、兴奋地编码直到灵感的版本出现在代码块中。如果开发小组其他人员同意这个设计，该开发人员就将新代码集成到现有的代码库中，直到用户对新的或者重新返工的部分感到满意为止。不断重复，其他开发人员灵感地重新设计从而会对代码进行优化。

敏捷编程技术包括“波浪式”灵感的软件开发思想。您需要确定敏捷编程是否适用于您。我已经对软件开发人员讲过，“灵感波浪式”技术并不是敏捷编程技术的全部。我建议您采用最适合您的方法。

在软件开发持续的过程中——从作出细小的改动到维护现有代码，再到开发重大新版本从而替代现有代码库——“灵感波浪式”技术对中间功效的开发要比不必完全重新编码的维护更加有效。

1.3.2 对目标的管理请求

在软件开发的经济系统中，工程管理通过引导资源和测量进度来确定检测点。许多管理者认为他们的管理是正确的，其中包括选择软件如何开发。管理人员可以查看新的开发技术并且坚持它的适用性。这引来了一个问题，软件开发人员的问题求解风格和开发力度要适用于管理人员所制定的开发技术。

以我个人经验，敏捷编程(也称为极限编程或 XP 编程，它的支持者(例如 SCRUM)详见 http://www.controlchaos.com/)对管理来说通常是“银弹”。管理的思想是：交付 2.0 版本项目是相当烦乱的事情；我们应该使用相应灵活的编程技术去发行一系列小的版本。事实上，尽管敏捷编程技术看起来像是管理的求解方法，但是在敏捷编程和“2.0 版本可靠性”神话之间并没有一定的联系。

敏捷编程中有几点对软件开发组织有很大的益处。例如，敏捷程序员认为应该采用许多小的改动，然后再测试为这些小改动所编写的新代码。当处理“2.0 版本可靠性”问题的时候，这种方法可能非常方便。所带来的问题是，管理可能对所有开发人员要求使用敏捷编程技术。

软件开发人员都有不同的问题求解的风格，而一些风格与敏捷编程技术之间相抵触。理解自身的开发力度和问题求解的风格对于发布成功的软件是很重要的。以我的经验，可以通过问题求解的风格和软件开发的力度对程序员进行分类。图 1-5 说明了程序员的类型和他们对应的人数。

起初，您可能怀疑为什么图 1-5 中包括用户和超级用户。以我个人的观点，每个计算机用户都是程序员。您见过 PostIt 记录的次数多吗？这些记录中带有用户喜欢的并且是用在连接监视器应用程序上的提示或者技术。PostIt 记录就是在编程。

图 1-5 表明了每种程序员类型都有不同的问题求解风格。用户是最大的使用群体，他们使用过程性的编程风格去求解问题。他们在应用程序的现有结构上完成任务。超级用户也是过程性问题的求解者，但是他们一般使用多个应用程序，因此如果问题在一个应用程序内无法求解，他们会在另一个应用程序中寻找求解方法。管理员也同样是过程问题求解者，但是他们比用户和超级用户可以访问更多的功能，甚至可以与代码编写者进行联系。这种访问权限给了他们更多问题求解的能力。

图 1-5 该图表明了在程序员中不同类型问题求解风格的相对人数

过程程序员和脚本编写者是企业问题求解者。一个脚本编写者很快有多个求解问题的方案是件好事。他们求解越多的问题就会越高兴。代码重用、效率和维护是下一步马上需要考虑的问题。脚本编写者要在整个软件体系结构都通过软件开发工具或者所使用的框架实现的情况下进行工作。例如，VB 用户可视化地布局应用程序，开发环境会询问该开发人员是否为每个可视化按钮填充可执行代码。开发环境编写其他的用于初始化、运行和关闭应用程序的代码。过程程序员也在其他人员开发出大部分体系结构的情况下进行工作，他们目前只是寻找一个功能的简单代码。

对象程序员要有重用的观点：如果软件代码在整个系统中完善了一个缺少的功能，它就应该重新编写。而且，在软件设计过程中所选取的代码必须对使用该软件功能的每个小组成员都简单易懂。对象编程是软件开发的高级部分。每个对象设计选择都会对系统产生重大的影响，因为即使最小的对象也会对整个系统的功能和性能产生影响。

互操作程序员能够为企业带来活力。他们在一定范围内审视所有的系统，设计应用程序软件，这些软件有时候会使用 Web 使能应用程序所提供的应用程序编程接口(Application Programming Interface，API)完成功能调用，尤其是 XML Web 服务。然而，他们的工作要比对象程序员更加困难，因为 API 通常提供单调的、非面向对象的方法调用。

最后，调配(orchestration)程序员通过设计他们所需要的功能去求解编程问题，然后定义业务工作流，以实现目标。调配程序员所编写的代码没有图形用户接口，所有编写的应用程序决策都对应着指导应用程序如何执行工作流从而达到满意结果的一系列优先选择。例如，银行中的贷款应用程序过程是一个工作流，调配程序员通过许多可用的步骤去实现处理贷款的软件。

试想，当管理在过程性程序员当中实现了面向对象设计过程的时，会发生什么情况呢？设想当管理人员决定为他们脚本程序的开发小组使用一个业务调配系统的时候，会发生什么情况呢？理解软件开发人员的问题求解风格对于发布成功的软件应用程序是至关重要的。

在图 1-6 中说明了查看软件程序员问题求解风格的另外一种方法。

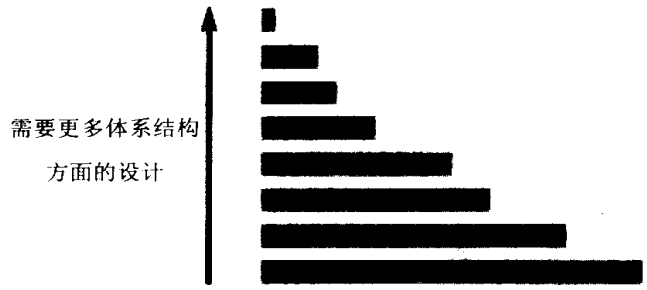

图 1-6　对于不同类型问题求解技术的计算机程序员，从图 1-6 中可以看出，在开发过程中的人数越少，就需要越多的软件体系结构方面的设计

图 1-6 说明了通过问题求解类型进行分类，软件程序员各个分组的相对人数。例如，过程程序员通常要比面向对象程序员多一些。这对我们设计成功的软件开发工具、平台和操作系统有着很大的影响。例如，如果我们的目标是吸引开发或者维护应用程序软件的系统管理员，那么我们的设计必须和他们的问题求解风格相一致，如在软件应用程序中，主要模块、模块之间的通信、外部访问协议、对象和变量的命名方法、资源的使用等要易于理解。另一方面，如果我们的目标是吸引更多的调配程序员，那么我们的设计重点必须放在如何连接有用的 Web 服务上面。

理解自身的开发力度和问题求解风格对于了解管理决策是如何影响软件开发工作的，有着至关重要的作用。

1.3.3　在异构规则的情况下使用同构规则

我中学的化学老师喜欢用自相矛盾的话题激发学生的好奇心。例如，他会问，“如果有一种能溶解一切物质的溶液，那您会用什么容器来盛放它？”。当遇到软件开发人员、QA 技术人员、IT 管理员告诉我他们的公司在单一平台上尝试着标准化基础架构的时候，我就想起这位老师。对，也就是同构的需求。这很显然是最自相矛盾的话。

这个矛盾就是将同构的基础架构等效成为去除任何异构。这是不可能的事情。任何事物都要在先于它之前的事物的基础上建立，难道会替代异构规则吗？这会带来杂乱，而且难以维护并且还不稳定——任何一处失效都会造成其他地方的多处失效。最终，没人知道如何去维护最早的系统，它将会变得陈旧不堪。

但是并不是说对同构的工作就停止不前了。这几乎是不可能的事情。取而代之的是我们要经常求解异构的问题。因为——作为本书的读者——我们是 Java 开发人员、QA 技术人员、IT 管理员，我们必须在异构系统中求解功能、性能和可扩展性的问题。因此，在需要将异构基础架构变换成为同构架构的时候，我列出了值得注意的如下几点：

- 为系统、系统的部件以及很多年没人管理过的陈旧系统作些补丁。在异构的环境下，如果公司不打算替代系统，那么打补丁才是最有可能的。
- 为脆弱性的系统提供对应用程序的服务支持的设计。当触及到、处理或者管理资源的时候，要保证代码可以处理资源不足的情况。
- 认可数据中心目标可变的性质。您的应用程序需要和其他应用程序有效地合作。当新的应用程序出现在数据中心而且在公司的小组之间没有很好的协调时，您会更好地体会到这一点。

- 为更复杂的应用程序提供客户端系统级测试。除了建立一个单元测试之外，还需要为可以驱动整体的求解方案而不仅仅是您所提交的部分功能提供客户端测试。当功能测试失效的时候，系统级测试将说明下一步的工程，而且 QA 技术人员和 IT 管理员在您的模块或者应用程序失效时，能够有一个诊断工具。在本章的稍后部分中，将介绍建立客户端系统级测试的一种新技术。

现在，您可能理解了这些问题是如何联系到一起的。作个总结，这 5 个问题共同使软件开发过程变得混乱、更具挑战并且不可预计。下面将介绍的另外两个问题会影响成功的软件开发。

1.3.4 带有 bug 的语言

以我个人的经验，大多数公司通常情况下会选择带有 bug 的列表作为软件开发人员、QA 技术人员和 IT 管理员之间交流的公共语言。而不是询问使用库存贸易应用程序的财经顾问是否能实现他们的目标，问题的提出就像这样：38283 错误是否解决了？

当编写或者维护一个软件应用程序的时候，您所选择的语言将会影响交付成功软件的能力。这一内容将在本章的稍后部分讨论用户原型和智能测试代理时进行介绍。但是现在，先来考虑这样一个示例：为一个软件应用程序编写一个新模块，您将选择下列哪种测试策略？

1. 通过发送一个简单的样本值测试代码。如果通过测试，那么您可以登记这些代码并且将它交给 QA 或者 IT 小组的成员。
2. 通过最广泛可能的操作，并模拟所有不同用户使用该模块的可能情况对代码进行测试。
3. 通过发送无效数据测试模块。如果失败，则准备交给 QA。

您注意到遗漏了什么吗？对于单用户来说会发生什么？您知道使用这个应用程序的单用户是为了实现他们的目标。他们在应用程序中操作，成功地使用了许多功能，每个功能用到了一组模块，直到他们用到了这些功能的正确组合，然后……，应用程序运行失败了。

我所遇到的多数开发人员认为上述 3 种选择是仅有的选择。他们认为必须进行无效数据的边缘测试，或者进行所有可能的广泛测试。我曾经见过开发人员通过对中间(middle)测试将这种差异分开。

对中间进行测试需要作出大量的假设，其中包括用户聚合小组将如何使用 Web 使能应用程序，以及作为一个小组来实现他们共同目标所采用的步骤。事实上，现代软件应用程序的形式和功能为用户选择他们在应用程序中的功能步骤提供了可能性。每个用户有其自身使用应用程序的目标和方法。但是在您的小组中需要为每个用户测试其使用的成功性，您如何描述这样一个用户，还有如何描述与其他成员之间的通信呢？

我推荐应该通过先建立仿造单用户的测试代理进行软件开发。仅选择一个用户，了解他的目标，弄清楚他期望使用的步骤，然后根据这个单用户建模测试。对单个用户需求的理解越深入，您的测试代理就越有价值。

一些开发人员已经记住了原型用户方法。他们为这些原型命名，描述其背景和特性。他们深入说明了原型，因此开发小组的其他成员可以更好地了解测试代理。目标驱动用户原型测试为开发人员、QA 和 IT 管理员之间的通信提供了非常有效的方法。

接下来，我们将讨论上述影响交付成功软件应用程序的 5 个问题中的最后一个问题。

1.3.5 邪恶孪生副总统问题

对于影响交付成功软件应用程序能力的 5 个问题中的最后一个问题，公司中对软件开发和

质量保证负有责任的部门认为它们是一个组，而对运行数据中心负有责任的部门认为它们却是另一个组。这种情况看上起随时都在发生，但是，这种想法断开了软件开发人员、QA 技术人员和 IT 管理员之间的联系。

如果长期持续这种情况，就会引出邪恶孪生副总统问题(the evil twin vice presidents problem)。在我们的时代，如果这个问题存在，就会使所有事情都慢下来，调度会变得不能满足人们的需求，并且人们的感情也会受到伤害从而离开公司。

我提出的对这种情况的解决方案为软件开发人员提供了引导单元测试，这些单元测试以后会被 QA 技术人员用于可扩展性和性能的测试，以证明服务质量的 IT 管理员可以使用这个相同的测试(您将会发现我在本书中反复多次回到这个话题。您将在第 2 章看到，我称这种求解方法为 Troika)。

总之，本节列出了影响交付成功软件应用程序能力的最重要的 5 个因素。这些问题大多是行为方面的而且与应用程序的编码没有必然的联系。然而通过本书，这些问题可能并不会影响您，本书用到了新的测试方法学、新的测试工具和架构集合、软件开发丰富的背景知识，即使您的公司遇到任何或者全部这 5 个问题时，本书也会帮助您正面地解决上述问题。

1.4 软件开发简史

现代 Web 使能应用程序的开发与过去的软件开发有所不同。正如本章前面所提到的，即便单行程序也与您没遇到过的人员所开发的代码库有互操作的关系。如果您在 20 世纪 70 年代访问国际电话电信公司(International Telephone and Telegraph，ITT)，您就会发现 ITT 工程师需要为公司的大型机数据中心编写软件的每个部分。回到现在，您会发现应用程序是由库组装在一起的，这些库是分散的而且通常是不相关的代码。那么我们是如何发展到现在这种情况的呢？

起初，因为计算机软件的功能很少，所以开发很容易。而且也很容易测试，因为公司拥有基础架构的全部。将计算机软件设计成为具有可扩展性还是很容易的，因为终端用户在公司雇员中是有限的。计算机软件的开发可能很容易，但是却很难使用。一些项目虽然提高了业务的效率，但是它们却很难使用，而且难以进行测试。

当用户、开发人员、测试人员和管理员适应了 Web 环境时，就随之诞生了现代软件开发的时代。在 Web 环境下，应用程序是为用户提供互操作性和连接性的计算机软件。即使是像个人数字助理(Personal Digital Assistant，PDA)那样的袖珍计算器，也希望能和 Internet 相连接。本书参考了这些具有互操作性和连接性的 Web 使能应用程序软件。所使用的设备可能是寻呼机、移动电话、掌上电脑、游戏面板、专用设备、台式机和服务器。如果软件内用到了超文本传输协议(Hypertext Transfer Protocol，HTTP)、简单对象访问协议(Simple Object Access Protocol，SOAP)、简单邮件传输协议(Simple Mail Transfer Protocol，SMTP)或者其他的 Internet 协议，那么该软件被认为是 Web 使能应用程序。

通过建立 Web 使能应用程序，开发人员又进入了一个崭新的生产力开发阶段。业务管理员将新的开发软件倡导视为提高生产力最快捷、最有效的方式，而用户喜欢在第一时间提供给他们新的访问和功能。通过选择恰当的软件测试工具、技术和方法，Web 使能应用程序的测试人员和系统管理员可以发布他们开发的最好质量的、最可靠的且最有使用价值的计算机软件。

本书将介绍开发人员、QA 技术人员和 IT 管理员的经验，他们都认为交付高质量的 Web 使能应用程序需要智能测试代理。本书还介绍分析可扩展性和性能的新方法，以及理解在开发

小组中行为化主题和个性如何影响 Web 使能应用程序的测试和监测。利用适当的技术确定 Web 使能应用程序的可扩展性和性能的索引，其索引用用于设计合适大小的数据中心并监测 Web 使能应用程序以达到服务级的认可。

当传统的上一代软件测试、监测工具和技术用于 Web 使能应用程序的时候，本书将详细说明这会引起什么情况。有时候其结果是没有意义的——或者更糟，甚至会带来误导。这里提到的新软件测试技术将会帮助开发人员权衡利弊从而使用户充分发挥自己的能力以实现重要的目标。本书为 Web 使能应用程序的健壮性列出了新测试工具和易遵守的准则。最后，本书还列出了新数据中心架构和开源测试工具，该工具在当前实际的生产环境下为建立可扩展和可执行的 Web 使能应用程序提供了一个简单且有效的方法。

1.4.1 Web 使能应用程序

十几年前，只有在电路、带宽、连接以及其他硬件资源实现之后，软件开发人员才会开始考虑软件应用程序的设计。一个 1.44KB 的调制解调器可能会在中途发生连接失败。所连接的系统仅仅实现了部分的 Kermit 协议。服务器端的 10MB 硬盘可能发生溢出并且操作系统发生崩溃(这些陈旧的技术将您带到过去了吗？)。这些早期的不完善系统经常出现故障，以致软件应用程序不得不去处理这些失效。

我们知道当底层平台不完善时，软件开发人员就需要关注它。开发人员为此会编写大量的错误处理代码，而不是专心于编写应用程序的表单、流程和逻辑。起初，优良的程序可能包含 300 行代码，但是其中只有 50 行用于处理应用程序中真正的业务逻辑，其余的是用来处理出错的情况。而现在，底层平台被详细地说明并且很好地部署。软件项目拥有与 Internet 相连接的策略、面向对象的编程语言、程序化接口、以及一组用户交互的异常(expectation)。虽然新技术和 API 可能相应地增加了新的复杂度，但是在当今的应用程序中，300 行处理业务逻辑的实现代码只有 50 行用来处理错误处理。

通过为部署和管理系统提供更多的灵活性，Web 使能应用程序给网络和 IT 管理员带来了众多益处。通过在大量单板电路大型机上使用许多小型、互连接的服务器，人们获得了可扩展性。在 Web 使能应用程序的环境下，部署、商业管理和升级维护都变得非常容易。

Web 使能应用程序也使得开发人员编写和维护应用程序的工具得到了改进。目前，开发人员通常使用 IDE 而不是单独的工具去开发具体的应用程序。IDE 通过集成许多单独工具和功能占领了市场的主体。目前，IDE 至少要提供具有上下文帮助的文本编辑器、编译器、调试器、源代码修订系统的客户(client)、部署包、还有使得程序员可以迅速开发新应用程序、模块和类的模板。

IDE 应该提供这样的环境：在编辑器中编写代码，将其编译成可执行代码，然后在调试环境下使用该应用程序。调试器可以逐行执行程序或者运行到某个特定条件以及断点，然后说明当前代码中的变量和数据结构。与过去相比，IDE 简直就是天赐之物。

遗憾的是，IDE 和调试器并不具备在远程服务器上获取 Web 使能应用程序的能力。为了测试这些系统，在软件为开发人员提供有意义的调试信息之前，经常需要应用程序中的业务逻辑到达某个状态。为了求解这些问题，测试图在几年前就出现了，其中包括 Brute Force 测试方法、大型机软件开发生命周期、桌面软件开发生命周期和 Internet 软件开发生命周期。

1.4.2 测试图和生命周期过程

现代软件在生命周期过程中开发。软件通过编写、测试、部署，然后为了维护又重新开始新的轮回。软件开发生命周期过程引来了许多可变的测试方法和测试图。

在台式机以标准形式出现的时候，就确立了 Brute Force 测试模型。Brute Force 方法需要许多工程师坐在台式机系统前，并执行下列任务：

1. 操作键盘和鼠标。
2. 使用诸如 UNIX tail 的程序查看日志文件。
3. 编写调试代码，将应用程序的内部值和状态保存到日志文件中。
4. 检查日志。
5. 每天重复 1000 次。

最后，测试人员用尽了耐心、花费了大量的努力(甚至作为一种爱好)也无法将所测试的应用程序中每隔 3 小时出现一次的大错误排除。Brute Force 测试模型可以通过减少返回值迅速地找到应用程序中的一点。几年之后，软件界意识到需要改变现存的软件开发生命周期。

为了了解软件开发生命周期在过去 30 年是如何发生变化的，让我们试想当新硬件、开发工具和连接性改进出现在这种情形下的时候，会发生什么事情？回到过去大型机的时代，软件测试是在真正的使用模拟器和测试数据的产品大型机上完成的。我认为软件开发生命周期是始于大型机软件开发生命周期的。生命周期中的步骤如下所示：

1. 指定程序、子系统和终端接口菜单。
2. 写出初始的文档。
3. 编写主要模块。
4. 模拟运行在系统上的模块。
5. 分析模拟结果，查找异常。
6. 为小模块编码。
7. 在带有测试数据的产品系统上运行系统。
8. 分析结果。
9. 修改实现中的每个问题。
10. 完成文档。
11. 提出最终的候选方案，并将软件交付给用户。

遗憾的是，最终测试和模拟的效果是一样的。因为仍然使用模拟测试新的维护版本，所以软件的维护同样是不确定的。这些测试也带来了对系统分析员的需求——最后证明系统分析员成为了在用户目标和项目实现的求解方案之间的协调者。

系统分析员系统化地审视软件，也会经常遗漏并忽略软件之后的使用性和性能问题。举个示例，在 20 世纪 80 年代早期，当代理查看顾客的订票信息时，飞机航线订票计算机系统无法查找往返飞行的航程信息。订票应用程序能够正确地显示航线，但是代理不得不退出该应用程序而进入往返飞行的应用程序以获得这种信息。订票应用程序与往返飞行应用程序在质量和性能上都能够符合系统分析员的准则；然而，旅客在机场却不能迅速地实现他们旅行的目标。

到了 20 世纪 80 年代末期，PC 产业的迅猛发展为桌面应用程序软件的出版市场提供了新的机遇。这是 Norton Utilities、dBase II 以及 WordStar 引领的时代。在这个时代，出现了桌面软件开发生命周期，以建立个人计算机软件的生命周期：

1. 以一个特征列表说明程序。
2. 编写软件应用程序。
3. 对应用程序进行单元测试。
4. 修改在单元测试中发现的问题。
5. 质量保障工程师对应用程序进行测试(alpha 测试)。
6. 修改发现的问题。
7. 外部 beta 测试人员对应用程序进行测试。
8. 修改发现的问题。
9. 将应用程序交付给客户。
10. 为维护周期收集带有 bug 的报告。

到了 20 世纪 90 年代，软件 QA 人员成为了开发人员的一部分并且管理桌面生命周期。

当桌面生命周期运转得很好时，用户对新软件版本的需求不少于 6 个月。在版本发布期间，软件发行人选择修复好错误的软件并且向着下个生命周期进展。许多用户遇到了非常严重的错误从而向软件发行人要求更快的维护版本。桌面生命周期太过于严格，以致于不允许采用较短的版本周期。即使很小的维护版本也只能从 6 个月的生命周期缩短为几个星期。另外，维护版本将开发小组的精力主要集中在开发新特征上面。

当软件发行人开始在管理桌面生命周期上更加高效的时候，又出现了分布和销售渠道的问题。苹果计算机能够在不到 6 个月的时间就开发出了维护和新特征的版本；然而，苹果公司的主要产品发行人却每隔 9 个月向公司要求新产品的版本。市场发布系统需要一个高效发行并且具有创新与改进意识的开发节奏。对于苹果公司，在公司工作的零售商、大商场、IT 管理员需要额外的时间，以使为下一版本软件作好准备。

桌面软件的性质会引起较长的生命周期。计算机产业将包含手册和软盘的物理盒交付给台式机的所有者。台式机与 Internet 相连接的优势对软件开发生命周期产生了深远的影响。在这个时代，出现了 Internet 软件开发生命周期，以建立基于浏览器的软件生命周期：

1. 以 Web 站点物理模型说明程序。
2. 编写软件。
3. 对应用程序进行单元测试。
4. 修改在单元测试中发现的问题。
5. 内部人员对应用程序进行测试。
6. 修改测试中发现的问题。
7. 将软件发布到 Internet 上。
8. 为现有服务器快速修复小的错误。

在上面的生命周期中，灵活编程技术交换了第 2 步和第 3 步。灵活编程建议应该首先开发测试，然后再开发应用程序的代码。

Internet 产生了计算的新形式：将许多小服务器连接在一起。Internet 软件的开发同出一辙，它是将许多小模块连接在一起。比起桌面系统管理，驻留在 Internet 应用程序中的业务更适合引入多个检测 bug 的装置。

随着更多的桌面计算机和服务器计算机与 Internet 相连接，软件开发开销降低到一个级别，在该级别个体开发人员能够编写、发布并支持软件。这些开发人员正在使用新方式编写代码，包括如下两个方面：

- **开源开发和分发** 软件来自源代码。用户查找错误既可以修改他们自身的错误，又能够加入开发社区从而宣布错误并修复这些错误。如果开源项目是必要的，那么也能够实现用户所需的更改。
- **极限(灵活或者 XP)编程** 开发人员的群体实现了复杂、庞大的系统，每个开发人员都工作在整个系统的不同部分中。在个体开发人员和小组之间要建立一条协定，该协定用预先定义并互相约定的 API 交付所需的软件模块。

从 90 年代末期开始，计算机产业见证了软件开发项目的迅猛发展。在新软件开发工具、新软件开发生命周期以及编写软件新方法的支持下，软件测试工具、技术和方法在 90 年代仍然是在软件开发中一个保持不变的领域。

1.5 测试方法

软件测试伴随计算机产业本身的出现而出现——不超过 40 年。因此新的软件测试方法每年都会出现，您对此不应该感到惊讶。在 Internet 软件开发的时代，现代测试方法分为如下几类：

- 点击流(click-stream)测试
- 单元测试(状态、边界、错误、特权)
- 功能系统测试(事务、端到端测试)
- 可扩展性和性能测试(负载测试)
- 服务质量测试

上述每种技术都填补了软件开发领域中的一个空白。它们都尽量满足每个软件开发人员的目标，从而交付高可用、高产出、高质量的应用程序。下面几节将对这些测试方法分别定义。

1.5.1 点击流测试

在 1996 年，当时的一个流行的 Web 搜索引擎公司 Excite 通过增加个性化特征赢得了电脑杂志的赞誉，其特征为：Excite 用户可以登录并且根据喜好查看他们家乡的天气情况。这是有史以来最先进的设计。不久之后，其他的搜索引擎 Web 站点也具有了个性化的特征。在这个门户或者网关开放的时代，Web 站点使得用户将站点个性化，并且可以访问比普通用户更深层次的内容。在个性创新出现之前，网页是由 Web 站点编辑手动更新的——Web 功能更像是网络电视机，观众改变频道观看某个电视剧。现在，每个人都希望 Web 使能应用程序能够提供个性化服务，与桌面应用程序操作不同的 Web 站点看上去就不合理了。

早先，测试 Web 站点意味着检查个性化服务，然后检查基础架构以保证站点持续运转。基础架构包括在用户浏览器和驱动 Web 的数据库之间所存在的一切，它具体包括网络路由器、交换机、负载平衡器、服务器硬件、Web 服务器软件、以及公共网关接口(Common Gateway Interface，CGI)程序。CGI 是运行在 Web 服务器上的软件最流行的定义，它为用户提供动态的内容。

起初的 Web 使能应用程序开发人员对点击流测试策略很感兴趣。在点击流中，用户通过点击，会从一个网页转到 Web 站点的另一个网页。点击的次数越多，Web 站点的发行人通过网页的广告和赞助就会赚到更多的钱。点击流工具说明了用户点击的 URL，Web 站点的用户

在当天具体的活动时间和其他数据会记录在 Web 服务器的日志中。对点击流测试策略的流行选择包括 KeyNode 系统 Internet 天气预报、WebTrends 日志分析工具和 NetMechanic 监测服务。

遗憾的是，点击流测试对于用户使用 Web 站点达到他们目标的能力几乎没有作用。例如，一个 Web 站点可以显示一百万个页面，但是其中 33%的页面可能是带有“没有找到搜索结果”的简单页面。使用点击流测试无法说明何时才能达到用户的目标。

1.5.2 单元测试

目前，个体开发人员对开发工具的选取变得不怎么重要了，而基于小组的使用面向对象技术的软件开发变得很平常。从 90 年代开始，计算机编程语言和操作系统平台都采用了面向对象的编程策略。相应地，开发人员也非常熟悉用于创建这些平台的语言和平台。每个小组成员通过自己的工具编写面向对象的代码，开发小组现在可以协同开发软件项目。

在小组中，个体工程师编写代码模块并确保模块的功能。小组通过一系列口头协议和电子邮件协议确定各个模块的功能和接口，从而达到联合开发。每个模块的内部工作由该模块的软件工程师负责。然后，工程师的主要精力通常是确保模块在调用时提供有效的数据，但是工程师并不关心模块内部是如何创建的。工程师保证模块在调用时给出有效响应的这种工作称为单元测试。

在软件开发结束之前，单元测试在模块一级发现问题和错误。通过给模块添加少量代码，使之作出正确的响应，从而完成单元测试。例如，一个模块可以监测空调的温度。用 Java 编写这样的模块，其代码如下：

```
Class checktemperature {
  private int temperature = 0;

  public int gettemp()
  {
    return temperature;
  }

  public Boolean validate( int temperature )
  {
    if ( temperature > 40 && temperature < 110 )
        return true;
    else
        return false;
  }
}
```

监测温度的对象有一个返回空调温度的方法(gettemp)。监测温度的对象也包括有效性验证的方法。空调温度传感器只能够在华氏 40 度到华氏 110 度之间进行操作。如果温度是华氏 20 度，就会返回无效的结果。

当该模块被编译成 Web 使能应用程序时，有效性验证方法为每个代码模块提供了检查其是否为合法数据的方式。这是通过在编译时运行单元测试工具而实现的。除了查看编译的语法错误和创建的问题，单元测试还将验证编译后的新对象。在编译软件模块期间，执行单元测试的编译工具的输出如下所示：

```
Compiling bigproject.
Compiling checktemperature.
Running unit test on checktemperature,input value is 55.
Unit test successful.
Building bigproject file.
```

虽然单元测试给开发人员带来了巨大益处，但一些开发人员却认为已经达到了安全性这种错误的假相。例如，某个开发人员会提出："已经通过单元测试了，因此会有很高的软件质量。"遗憾的是，如果单元测试是用于确定 Web 使能应用程序的健壮性的惟一方法，那么它将没有任何意义。

测试 Web 使能应用程序也必须包括智能的端到端的系统测试。系统测试将检查整个 Web 使能应用程序，包括从用户操作键盘和鼠标一直到 Web 使能应用程序的逻辑，再到数据库和底层的通信系统。如果存在问题，用户将不能实现他们的目标。

1.5.3　功能系统测试

虽然单元测试为开发人员提供了一种非常不错的技术，但用户不会仅操作某个单独的模块。取而代之的是，用户将访问构成整个系统的模块集合。系统测试从端到端适当地检查软件的功能。图 1-7 说明了在 Web 使能应用程序数据中心产品中所涉及到的组件。

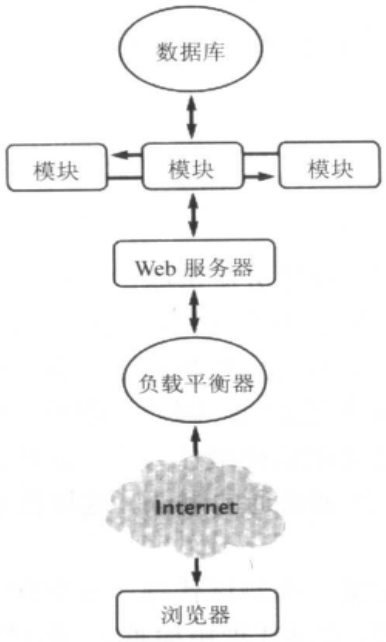

图 1-7　一个 Web 使能应用程序的组件

功能系统测试将为整个应用程序检查从客户端到数据库之间的全部组件，客户端是图 1-7 中所描述的 Web 浏览器，但它并不是在网络连接之上使用开放协议的应用程序。Web 使能应用程序的框架部署 Web 浏览器软件、TCP/IP 网络路由器、网桥和交换机、负载平衡路由器、Web 服务器、Web 使能应用程序的软件模块以及数据库。还可以部署额外的系统，从而提供目录服务、用于音频和视频流的媒体服务器，以及用于电子邮件的消息服务。

测试人员常犯的一个错误是当正在测试系统的单独组件时，他们认为建立了系统测试。例如，如果 Web 服务器返回的页面仅仅含有静态的 HTML 页面，则检查该页面并不是系统测试。相应地，这样的测试仅检查 Web 服务器——而不是系统的所有组件。

1.5.4 可扩展性和性能测试

通过可扩展性和性能测试可以帮助理解系统是如何对多个并发用户引起的负载进行处理的。在 Web 环境下，并发使用是通过在同一时刻用户所作出的请求数目进行度量的。本书中的一个中心主题是：执行功能系统测试可以并且应该对可扩展性和性能测试有指导作用。您所选取的测试工具应该带有功能系统测试而且在服务器上多次或者并发执行以提供负载。这种方法意味着服务器从测试中检查到负载，这种测试要比以往的方法更接近实际产品的环境。

1.5.5 服务质量测试

了解系统处理用户负载的能力是正确提供数据中心的关键所在，然而，可扩展性和性能测试并不能说明真正的数据中心在产品中是如何执行的。本章前面提到同一个功能系统测试可以并且应该重用于监测 Web 使能应用程序。通过长时间运行功能系统测试，结果日志是提交给用户关于服务质量的证据(即使在服务级别仍然很高的情况下，这些日志也可以为提高质量给出基础性的建议)。

本节说明了主要测试类型的定义，也为开发人员、QA 技术员、IT 管理员在测试系统的功能、可扩展性和性能时给出了提高各自能力的实例。

接下来将学习如何将用户典型的行为建模成智能测试代理。测试代理是自动化单元测试、功能系统测试、可扩展性和性能测试以及服务质量测试的关键。下面的章节将给出这些测试方法的定义。

1.6 定义测试代理

以我个人的经验，功能系统测试和服务质量测试是所有测试中最困难的，因为它们要求测试人员了解用户目标。将用户目标转换为测试代理的代码也同样具有挑战性。

当软件使得用户实现了重要目标并且这些目标因人而异时，Web 使能应用程序增加了自身的价值。由于可以根据用户的目标将其分组，因此理解单一用户的目标并且确定 Web 使能应用程序如何帮助用户实现目标是建立测试的最佳方式。这样的测试将能更好地确定 Web 使能应用程序的功能，从而可以执行、扩展普通的测试。这种技术可以将用户目标转换成测试代理的代码。

执行系统测试的正式方法是定义一个测试代理，这个代理为获得具体目标对 Web 使能应用程序的单一用户的操作进行建模。测试代理由检查表、测试过程和报表方法(reporting method)组成，其表示如表 1-1 所示。

表 1-1 智能测试代理的组件

组　件	描　述
检查表	定义条件和状态
测试过程	定义需要执行检查表的事务
报表方法	在完成测试过程和检查表之后，记录结果

假定 Web 使能应用程序为旅行代理提供了在线订单入口服务，该服务可以从包价旅游承办商那里获得旅行手册。订单入口服务每到春天增加新的旅行手册并且删除之前季节的手册。订单入口服务的测试代理将模拟一个订购当前手册和过期手册的旅行代理。测试代理的任务是保证操作要么成功，要么失败。

下列方法通过确定检查表、测试过程和报表方法来实现旅行代理测试这个示例。检查表定义了 Web 使能应用程序能够满足的条件和状态。例如，需要定购旅行手册的购物篮的应用程序，其检查表的形式可能如下：

1. 浏览当前手册列表。会出现多少个手册呢？
2. 订购当前手册。该服务提供认证号吗？
3. 订购过期手册。该服务提示错误吗？

检查表确定 Web 使能应用程序所需的状态。例如，当测试订购当前手册的 Web 使能应用程序时，Web 使能应用程序的状态被设置成持有当前手册的状态；相反，当出现定购过期手册时，Web 使能应用程序处于错误状态。

测试代理过程定义了几个所需的步骤，以初始化 Web 使能应用程序，然后运行 Web 使能应用程序，其中包括遍历检查表。测试代理过程会处理事务。在旅行代理手册订单入口系统中，测试代理需要以下事务：

1. 初始化订单入口服务。
2. 查找手册序号。
3. 定购手册。

事务要求有一定数量的独立步骤。例如，事务 2 需要测试代理签署订单入口服务，提交表明所需的手册号码的请求，确认手册存在，提交订购手册的请求，然后离开。

最后，测试代理必须包括一个报表方法，以定义处理的结果保存在哪里且以什么格式保存。手册订单入口系统测试代理为成功订购的手册和所订购的过期手册打印出报表。

测试代理可以用几种格式来表示。测试代理可以在纸上定义，由人执行。测试代理也可以是驱动 Web 使能应用程序的程序。测试代理必须为 Web 使能应用程序所产生的结果定义重复的方法。测试代理的自动化程度越高，开发管理员就能在更有利的位置上保证为用户创建更完善的 Web 使能应用程序。

测试代理定义会带来如下优势：

- 回归(regression)测试变得更容易。对 Web 使能应用程序软件的每次新更改或者在维护上的变动，测试代理说明哪些功能继续有效，哪些功能无效。
- 回归测试也可以表明 Web 使能应用程序可以被用户访问的程度。回归得越少，开发速度就越快。
- 开发测试代理也为可扩展性和性能测试提供了快捷的途径。因为测试代理为 Web 使能应用程序的个人用户的使用进行建模，所以同时运行相同测试代理的多个备份可以使可扩展性和性能测试变得更加简单。

具有测试代理的可扩展性和性能测试

测试 Web 使能应用程序要比测试桌面软件更加困难。在任意时刻，中等规模的 Web 使能应用程序都需要处理 1~5000 个并发用户请求。在数百个用户带来的负载下，了解 Web 使能应

用程序的可扩展性和性能特征对于管理软件开发项目、建立充足的数据中心以及保证用户正常使用都是非常重要的。通常情况下，Web 使能应用程序的互操作模块被用户激活后才会显示其特征。

可以采用两种方式对 Web 使能应用程序进行分析：可扩展性方式和性能方式。我发现只分析其中一个而忽略另外一个的方法通常会得出无意义的结果。Web 使能应用程序如果具有为 5000 个用户迅速服务的能力，而其中 500 个用户接收了错误页面，这个结果难道很好吗？

可扩展性描述了 Web 使能应用程序在不同负载级别下为用户服务的能力。为了测量可扩展性，运行一个测试代理并测量其运行时间。然后分别在 1、50、500 和 5000 个并发用户的情况下运行相同的测试代理。可扩展性属于功能性测量。可扩展性在负载的条件下测量 Web 使能应用程序完成测试代理的能力。经验表明，一个测试代理应该测试 10 个数据点才能给出有意义的结果，但是测试的次数要依赖于运行测试代理的开销。总结这些测试结果有助于开发管理员预测在特定负载条件下 Web 使能应用程序为用户服务的能力。

表 1-2 说明了一个 Web 使能应用程序的可扩展性的测试结果。表题下的第一行数据显示的是在一个用户的情况下运行测试代理的结果；在该表中，Web 使能应用程序完成测试代理的时间少于 1 秒钟的占 85%；完成测试代理的时间少于 5 秒钟的占 10%；超过 5 秒钟的只占 5%。

表 1-2 说明 Web 服务可扩展性的实验结果

性　能	< 1 秒	2 ~ 5 秒	> 5 秒
1	85%	10%	5%
50	75%	15%	10%
500	70%	20%	10%
5000	60%	25%	15%

当 Web 使能应用程序在有负载的情况下，注意该表发生的变化。当 50 个用户并发运行同一个测试代理时，Web 使能应用程序执行的结果比只处理单用户的情况要差一些。在 50 个用户的情况下，完成相同的测试代理时间少于一秒钟的占 75%。当 5000 个用户开始运行测试代理时，Web 使能应用程序就有些承受不住了。在 5000 个用户的情况下，完成测试代理的时间少于 1 秒钟的只占 60%。

在得到所规定的最少数量的数据点之后，可以推断 Web 使能应用程序的可扩展性结果。例如，如果只有 1 个和 50 个用户的可扩展性结果，那么推断 5000 个用户的可扩展性就毫无意义。然而，至少在 4 个级别负载下运行可扩展性测试可以给出有意义的结果，从中得出的推断才会是有效的。

接下来，我们讨论性能索引。性能是与可伸缩性相反的一面。可扩展性在负载增长的情况下测量 Web 使能应用程序为用户服务的能力，并且假定有效的测试代理都会成功地进行测试。可扩展性对用户经验是没有作用的。与此相反，性能测试的测量是无效的。

性能测试用来估计 Web 使能应用程序正确提交功能的能力。性能测试代理查看测试代理的结果以判断 Web 使能应用程序是否会产生异常结果。例如，在表 1-3 说明的可扩展性测试示例中，性能测试说明了在不同负载的情况下所返回的错误网页的数目。

表 1-3 性能测试代理结果示例

性　能	<1 秒	2~5 秒	>6 秒	总　和
1	1%	5%	7%	13%
50	2%	4%	10%	16%
500	4%	9%	14%	27%
5000	15%	25%	40%	80%

表 1-3 说明了 Web 使能应用程序示例的性能结果，该示例的可扩展性结果在表 1-2 中。性能结果表明了同一个 Web 使能应用程序的不同方面。在 500 和 5000 个并发用户级别上，仅仅注意可扩展性结果的开发管理员可能会作出将 Web 使能应用程序发布给用户的决定，即使表 1-2 说明了在 500 个并发用户级别上 10%的提交页面响应时间很长——在这种情况下最慢的测试代理需要执行 6 秒多钟的时间。开发管理员在看过性能测试结果之后还会发布该 Web 使能应用程序吗？

表 1-3 表明了当测试代理在 5000 个并发用户服务级别上完成测试的时间少于 1 秒钟的情况，Web 使能应用程序在 15%的时间里是失效的。在少于 5 秒钟内完成测试代理增加了 25%的失效，而在多于 6 秒钟完成测试代理又会增加 40%的失效。这样在 5000 个用户并发负载的情况下，开发管理员会预测有 80%的用户将接收到错误页面。

可扩展性和性能测试都需要确定一个 Web 使能应用程序在运行期间对用户提供什么程度的服务。如果单独分析，这两个测试结果可能没有说明 Web 使能应用程序的特征，甚至会更糟，它们会误导最终的结果。

综上所述，可扩展性和性能测试能够表明 Web 使能应用程序的特征。

1.7 为单个用户进行的测试

许多开发人员认为直到测试覆盖了用户群的全部情况，测试才真正结束。另外一些开发人员认为测试高质量软件需要依靠 Web 使能应用程序的原始设计目标，该设计目标是由产品管理员、项目市场销售人员或主要开发人员定义的。然而这些方法并不充分，因为它们仅仅是针对中间人进行测试的。

像中间人测试，它需要作出大量的假设，这些假设是关于用户的聚合小组是如何使用 Web 使能应用程序，以及为小组成员实现共同目标的步骤。但是 Web 使能应用程序并非简单地用于这种方式。实际上，每个用户都有各自使用 Web 使能应用程序的目标和方法。

Intuit 是流行的 Quicken 个人财政管理软件产品，它先识别出每个用户的特殊使用方法。Intuit 开发出“跟我回家(Follow me home)”软件测试方法。Intuit 开发人员和产品管理员到附近的软件零售商店，在 Intuit 产品旁边等待顾客选取 Quicken 软件的副本。当顾客准备买 Quicken 软件时，Intuit 管理员就自我介绍并请求到顾客的家中，从而可以了解到用户安装和使用 Quicken 软件的过程。

Intuit 测试人员留在办公室里，模拟 Quicken 软件用户一般类型的主要交易。相应地，他们开发用户原型——典型的 Web 使能应用程序的用户，这是根据他们所遇到的真实用户以及这些用户的使用方法开发出来的。可以将同样的方式应用程序到开发测试代理上面。使用描述用户的原型要比对 Web 使能应用程序的用户所做出的概括更加有效、更加准确。根据使用 Web 使能应用程序的个体用户的目标和方法建模成测试代理，原型使开发它变得更加容易。

建立一个原型测试代理最好的方法是从一个单用户开始。仅仅选择一个用户，通过 Web 使能应用程序观察这个用户并了解用户希望使用的步骤。然后利用这些信息，根据这个单用户建模原型。对单用户的需求理解得越透彻，原型具有的价值就越高。

一些开发人员将原型的用户方法熟记于心。他们为自己的原型命名并且描述其背景和特征。他们描述原型越详细，开发小组的其他成员理解测试代理就越容易。

例如，设想为我创建的一家名为 Inclusion 的 Web 使能应用程序的软件技术公司定义原型用户。在 1997 年，Inclusion 公司开发了一个 Web 使能应用程序，它可以为跨国公司不同地点的小组提供协同消息服务。像 BP 这样能合并 British Petroleum 和 Amoco 的能源公司，使用 Inclusion 的 Web 使能应用程序建立一个安全私有的大型网络(extranet)，利用它，BP 财政审计组中的职员和签约人可以在正常工作的情况下交流想法和好的经验。

BP 大型网络的测试代理是根据下列原型用户设计的：

- Jack，审计员，22 岁，最近从西北大学毕业进入 BP 公司。未婚，但是有感情很好的女友，从中学开始接触电子表格软件，如果能提高工作效率愿意使用新技术，爱好是摩托车越野赛和滑雪。
- Madeline，总部管理员，42 岁，结婚 15 年并有两个孩子，在 BP 公司的职位稳步提高，在进入管理工作之前从事 IT 行业 3 年，遵从已有的工作方式，但是愿意在使用提高小组生产力的技术的环境下工作。
- Lorette，IT 支持，27 岁，配有两部传呼机和一部移动电话，每天花很长的时间维护系统，为新雇员做系统培训，希望在其他地方参加培训讨论会。

以 Jack 为目标建模的测试代理的注意力放在访问和操作数据上。Jack 经常需要查找以前保存的电子表格。在这种情况下，测试代理登录到 Web 使能应用程序，并使用查找功能定位文档。测试代理修改文档并且确保正确地保存了修改后的文档。

为 Madeline 开发的测试代理注意力集中在使用数据上。测试代理首先使用 Madeline 的高级安全认证登录到 Web 使能应用程序。这需要给出运行使用报表的权限，然后查看在她的组中哪个程序正在充分利用 Web 使能应用程序。当需要性能评定的时候，测试代理对于 Madeline 来说是非常重要的。测试代理也尝试以 Jack 的身份登录并且访问相同的报表。如果 Web 使能应用程序正确执行，那么只有 Madeline 可以访问这些报表。

以 Lorette 为模型的测试代理注意力集中在访问数据上。当 Lorette 离开办公室参加一个培训讨论会时，他仍然需要访问这个 Web 使能应用程序，就像在他的办公室里一样。测试代理使用远程登录的功能访问所需的数据。

理解这些原型对创建测试代理的智能非常关键。例如，Lorettte 的测试代理可能比 Madeline 的测试代理在行为上更加持久。如果测试代理尝试建立远程连接但是失败了，Lorette 的测试代理会继续尝试，然后更换一个不同的访问号码。

1.8 创建智能测试代理

开发测试代理和原型对建立良好的测试数据是一种快速、可预计的方法。测试数据越好，就越能了解在负载情况下 Web 使能应用程序的可扩展性和性能。分析数据说明了可扩展性和性能指标。理解可扩展性和性能可以了解开发和操作高质量的 Web 使能应用程序的开销。

测试 Web 使能应用程序在许多方面与医生为病人看病有很多相似之处。例如，为肺癌病人治疗的肿瘤医生从不会说出病人还有几天生命。因为这不是肿瘤医生最基本的常识，他们的基本常识当是研究并且治疗肺癌。肿瘤医生通过流行病的术语研究肺癌，只有当收集了大量其他病人的统计数据时，个体病人的测试才会有意义。如果医生确认一个病人属于总体病人中的某一类，那么肿瘤医生按照这类病症的情况对该病人进行治疗。通过相同的方式，只有使用系统才能测试 Web 使能应用程序，因此无法保证一个系统将会出现这种或者那种情况。然而，可以观察测试代理的结果并且推断该产品系统的性能和可扩展性。

会计和业务管理员经常引用收益递减(diminishing returns)的规律，它的意思是为一个项目开发一个逐渐增长的功能，相对于功能代价的回报将会越来越少。有些时候这种想法也融入到软件测试项目当中。

可以向自己提问，达到什么程度可以使得测试代理和原型产生有意义的结果？实际上，无法使用充足的测试代理和用户原型确保 Web 使能应用程序的健壮性。因为个体测试代理可以得到有用的信息，因此需要很多测试代理。每个测试代理都要提供使用 Web 使能应用程序的个体用户的目标和方法。综上所述，测试代理表明了可扩展性和性能的模式。

许多测试代理的数据可以用来作出预测。每个新的 Web 使能应用程序或者维护版本可以在预计时间和预计资源上根据测试代理库进行测试。

自动测试

在手动执行的情况下，开发和运行测试代理的开销是很大的负担。在第一个测试代理开发之后，紧接着就是第二个。很可能处于无休止的循环。其结果是，根据个体用户选取的测试模型会带来不断增长的测试代价——或者需要自动化测试。测试自动化使测试代理库将持续增长，而且测试代价可以保持不变。

测试自动化使系统并发地、大量地运行测试代理。许多自动测试代理在实际产品的环境下使 Web 使能应用程序处于正常级别。用户的数目要依赖于希望使用 Web 使能应用程序的用户。经验表明用户数目增加 10 倍，将测试出相反的结果。

每个自动测试代理都包含一个原型的行为。相同测试代理的多个并发执行的副本将得出有趣和有用的测试结果。在实际生活中，许多用户在不同的时间里会做出相同的行为。智能测试代理使我们的测试更加接近于现实世界的产品环境。

1.9 本章小结

在本章中，我们了解了当前计算机产业的状态。知道了软件开发人员、QA 技术人员和 IT 管理员都拥有建立集成 Web 使能软件应用程序的、大量可选用的工具、技术、硬件和知识。还了解到工具、技术、方法的选择，对系统的可扩展性和可用性都会产生一定的影响。

我们学习了建立高质量 Web 使能应用程序所需的工具、方法，并很好地了解了开发小组的行为。在第 2 章中，将详细研究智能测试代理如何提供有用的、有意义的数据，以及在使用 Web 使能应用程序时如何确定用户满足他或她的需求的程度。我们还研究了智能测试代理如何自动运行测试组件，并讨论了测试组件结果所必需的测试环境和测试工具。

本章描述了使得 Web 使能应用程序流行的因素，并且介绍了如何通过新的软件测试工具、新技术以及新方法达到更高级别的生产力。

第2章　应用程序的性能何时会带来问题

软件开发人员通过使用已确立的优良质量方法和标准对分布式系统的性能进行传统方式的量化。本章将介绍用于测试 Web 使能应用程序的众多新方法和标准。新的软件测试技术帮助开发人员、QA 技术人员、IT 管理员更好地达到用户的个人目标。本章介绍的新工具将提供确定 Web 使能应用程序健康性的容易掌握的标准，例如，这些工具包括 WebRubric 和 Web 应用程序点系统(Web Application Points System)。

2.1　什么是标准

Internet 发展到现在，Internet 用户都希望 Web 使能应用程序能够达到满意的程度，但这带来了一种两难的局面：一个用户满意的标准可能是另一个用户无效的定义。另外，某个用户的满意标准也常常随时间的变化而变化。图 2-1 说明了用户标准如何随着时间不断变化。

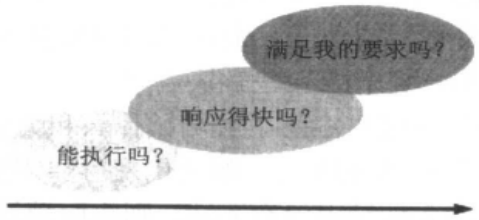

图 2-1　用户变化的需求

图 2-1 说明了尽管根据目前承认的一系列目标启动 Web 使能应用程序是可行的，但是目标会随着时间的推移而变化。过去的 Web 仅仅接收满足用户需求的 Web 网页。之后，只有能作出快速响应的站点才能满足用户的需求。近来，用户通常根据能够快速满足他们个人目标的服务能力来选择 Web 使能应用程序。所显示的页面必须是经过排版的有意义且有用的数据。当然，它还必须能够快速地显示出来。

随着用户需求的不断变化，某天运行系统的结果会出现与第二天完全不同的结果。根据我个人的经验，Web 使能应用程序需要经常测试、监测和观察。在一个零售商店模拟中，Web 使能应用程序的商店每天营业 24 个小时，在任何时刻商店中会有 1~100 000 个顾客。因此，这需要引入智能测试代理！

第 1 章介绍的智能测试代理是在每天中测试 Web 使能应用程序的一种快捷且有效的方式。智能测试代理能够自动完成测试过程，因此测试每天都可以进行。基于用户原型的测试代理，为确定 Web 使能应用程序的能力是否满足用户需求，提供了有效的数据。因为用户的需求目标不断变化，我们可以根据新的用户原型添加新的测试代理。

为了更好地理解用户原型，下面给出一个示例。图 2-2 介绍了一个典型用户原型的描述。

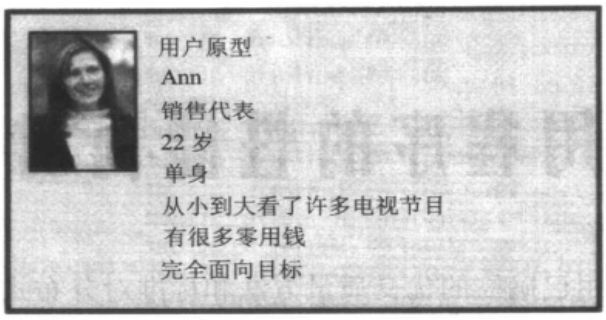

图 2-2 一个用户原型的示例：Ann 是一家制造业公司的销售代表。为用户原型描述越多的个人信息，开发小组就会得到越多的回报

通过这种方式定义用户原型需要依靠您的想象力。花费在定义原型方面的时间越多，小组成员可能越容易理解原型。在小组中使用这种原型技术将获得如下结果：

- 原型在应用程序的原型用户目标与交付应用程序的软件开发人员、QA 技术人员、IT 管理员之间建立了感情上的连接。小组成员会有关于某人的易于理解的示例，这个示例将原封不动地用于 Web 使能应用程序。小组能够指出其功能而不仅仅是讨论在应用中的个体特征，原型通常将利用这些功能实现用户的个人目标。
- 原型引来了在 Web 使能应用程序中对特征、形式和功能的讨论，讨论包括模糊通用的主题到集中的面向用户目标的探讨。例如，小组并不单独讨论用户注册页面的功能，对用户注册页面的讨论将涉及到 Ann 是如何使用注册页面的。
- 原型使小组理解了需要将新的功能、错误和问题安排在全局调度中的什么位置。这是因为 Ann 比其他未知用户更需要一些解决方案。

不仅如此，用户原型还能使小组更容易了解测试什么和为什么测试。例如，Ann 模型的测试代理将注意力集中在实现其目标的速度上。测试代理可能需要驱动一个 Web 使能应用程序，从而提交一个查询，然后如果在 2 秒钟内没有完成，则将取消这次查询——这是因为 Ann 目标需要在很短的时间里完成。将 Ann 模型的测试代理添加到测试代理库中，其中还包括在第 1 章中介绍的 Jack、Madeline 和 Lorette 测试代理。

最后，开发新的用户原型和根据原型行为建模的测试代理会产生自动的 Web 测试，它与系统所服务的真实用户非常接近。而且，用户原型和测试代理使我们可以了解 Web 使能应用程序何时会带来问题。

2.2 Web 性能的标准

在 Internet 出现之前，客户/服务器网络系统在个人网络上运转，并且以可预计的次数、以简单模式被熟知的可预测的人员所访问。在很多情况下，网络通常在具有几个远程地点的独立办公室内运转。

利用现有的路由器、网关、网桥以及其他网络设备，办公室局域网(LAN)内的任何一台设备都可能在任意时刻接收来自其他任意地方的请求。办公室网络可以不直接暴露给开放的 Internet，但是雇员可能在家中访问办公室网络，远程办公室通过网桥访问办公室网络，局域网也可以使用 Internet 语音协议(VoIP)处理电话系统的报文流。局域网上的 Web 使能应用程序将面临由广泛异构的、不可预测的用户组所引发的难以预测的负载模式。

因此，如果公司忽略了负载模式，会有什么损失呢？极少有公司会为 Web 使能应用程序基础架构的正常运行而奖励它。但如果它停止工作，那就非常愚蠢了！过渡负载对公司的账户底线、市场价值、品牌资产净值(equity)都有着严重的影响——但这并没有损及网络管理员的名誉。例如，回想一下 eBay 在 1999 年由于负载相关的数据库错误失效了 22 个小时。公司因而损失了两百万美元，eBay 股票下跌了 10%(详细信息见 http://www.internetnews.com/ec-news/article.php/4_137251)。虽然很多公司的损失没有 eBay 那么严重，但负载模式也不应该被忽视。

注意负载模式并选择适当的测试方法学是这种测试有效性的关键。目前，在 Internet 或者大型网络上发布的 Web 使能应用程序的可靠性和鲁棒性的测试方法学是必须的。使用不良的测试方法学，其结果没有任何用处，在最坏的情况下还会引起误导。

Web 使能应用程序的性能标准每年都会发生变化。利用这么多的信息，很容易使用现有的知识确定并更改过去的标准。通常“老”的测试技术包括 ping 测试、点击流度量工具和服务、HTML 内容检测。

- **ping 测试** 使用 Internet 控制信息协议(ICMP)给服务器发送 ping 请求。如果有 ping 返回，就认为服务器工作良好。不利的方面(downside)是即使在 Web 使能应用程序失效的时候，Web 服务器也持续地回应 ping 请求。
- **点击流度量** 测试为一系列网页发送请求并记录响应的统计数据，其中包括每小时全部的页面浏览，每周全部的点击次数，每周全部的用户会话，以及它们的派生数据。不利的方面是如果 Web 使能应用程序使用多于两倍的网页才可以实现用户的目标，点击流测试表面看起来好像网站很受欢迎，而对于用户来说这个 Web 站点很差劲。
- **HTML 内容检测测试** 向一个网页发送请求，分析 HTTP 超级链接的响应，向相关的宿主请求超链接，并判断链接返回是成功还是失败。不利的方面是在 Web 使能应用程序中超链接是根据用户的请求动态可变化的。几乎无法了解在 Web 使能应用程序中超链接的上下文。如果没有误导，仅仅检测链接的有效性也是没有意义的。

理解并开发 Web 使能应用程序的性能标准需要基于用户每天的经历。仅仅使用 ping 测试、点击流度量或者 HTML 内容对 Web 使能应用程序进行测试的公司，只能得出无用的甚至有误导作用的结果。这些工具只是用来测试静态网站的而并不是用来测试 Web 使能应用程序的。

实际上，当用户在几个 Web 使能应用程序中进行选择时，Web 使能应用程序的性能标准的选择应该基于如下 3 个关键问题：

- 功能有效吗？
- 性能可以接受吗？
- 失效的频率是多少？

2.2.1 功能有效吗

首先汇集基本功能的一个简单列表。这个列表通常可以在创建 Web 使能应用程序的软件开发人员所使用的功能需求文档中获得。记下这个功能列表，设想一下第一次使用该 Web 使能应用程序的用户，还有这个用户准备使用该 Web 使能应用程序的情形。

下面的示例说明了本公司的 PushToTest 所支持的在线服务的基本功能：

- 登录和离开。
- 导航到一个讨论消息。
- 下载一个文档。

- 发出一个消息。
- 使用关键字查找消息。

即使 PushToTest 网站有 480 多个服务对象、13 000 行代码以及一个通用的用户接口，但是它能提供上面列出的 5 个功能以保证应用程序完全正常运行。

2.2.2 性能可以接受吗

检测 3~5 个用户，以确定他们在等待 Web 使能应用程序执行某个功能时，因无法忍受而转向另外一个功能之前的等待时间。花费一些时间直接观察用户，确定执行一个基本功能需要多少秒。

2.2.3 失效的频率是多少

如果日志里的记录是有效数据，Web 使能应用程序的日志就可以显示出失效之间的时间。首先，为用户开发出在可接受范围内的失效百分率还是比较诱人的。失效之间的时间是一个绝对数值。为了更好地估计可接受的数值，需要查看真正的日志以得到正确的答案。2.3 节所描述的 Web Rubric 是一种很好的方法，它有助于理解失效因素和统计结果。

2.3 Web 使能应用程序的度量工具

以我个人的经验，对 Web 使能应用程序的最好度量包括如下几个方面：

1. 失效之间的平均时间(单位为秒)
2. 每个用户会话的总时间(单位为秒)，用户会话有时也称为事务
3. 应用程序的可用性和峰值使用周期
4. 用得最多的媒体元素有哪些(例如，HTML 与 Flash，JavaScript 与 HTML 表单，Real 与 Windows 媒体播放器、QuickTime)

为 Web 使能应用程序的性能开发标准将是一项深奥而且困难的任务。在这里，我们给出一个更实际的、定义性能标准的方法示例。

Web Rubric

在为 Web 使能应用程序的性能开发标准的过程中，测试方法通常包括很多主观性：标准评价有时候可以很好地评定一个 Web 使能应用程序，但是当检查评定的时候，评价标准是模糊的并且性能行为过于主观臆断。这令每个测试人员进入了一种想要尝试验证测试结果的奇怪境地。Web Rubric 是一种可信的评估工具，在 Web 使能应用程序的结果十分复杂且主观时，它在评定 Web 性能标准上特别有用。

可信的评估是一个科学术语，比科学术语更好的名称可能是“根据实际”。几年前，人们根据性能来评估学徒(apprenticeship)系统。在可信的评估中，教师在实际的工作中观察学生，提出反馈意见，再观察学生使用反馈意见的情况，然后相应地调节指示和评价。

Web Rubric 设计模拟现实生活从而提供一种可信的评估工具。因为它是软件开发生命周期的一部分，所以它是评估的一种格式化形式。它也包含开发人员本身，开发人员评估 Web 使

能应用程序是如何执行的。随着时间的推移，开发人员可以协助设计 Web Rubric 的后续版本。可信的评估使得开发人员、测试人员和用户的界线变得更加模糊。

表 2-1 是电子邮件协作 Web 服务的 Web Rubric 的一个示例。

表 2-1 一个电子邮件使能的 Web 应用程序的 Rubric

根据系统使用情况的评估标准	第 1 级 开始	第 2 级 开发	第 3 级 标准	第 4 级 标准之上
基本功能有效	第一次使用时只有少数功能正确	许多功能无法运转。需要一些缺失的功能来完成工作	大多数功能可以运转。所提供的工作区 (workaround) 可以完成工作	每次使用时，所有的功能都能正确运转
运转速度	许多功能从不完成	大多数功能在用户失去耐心之前完成	大多数功能在 3 秒之内完成	所有的功能都在 3 秒之内完成
正确的运转	极少数功能成功地返回一个错误情形	某些功能结束于一个错误情形	大多数功能正确地完成	所有功能都正确地完成

注：从这个 Web Rubric 中可以得出的正式标准如下所示：在标准中最高、最一致的分数一定在第 3 级，第 1 级没有分数。对于使用这 3 个标准的 Web 使能应用程序，其中两个必须在第 3 级，剩余的那个标准也要比第 1 级高。

在评估中使用 Rubric 的益处有如下几点：

- 评估更加客观和一致。
- Rubric 可以帮助测试人员将注意力集中在通过具体的术语阐明的标准上面。
- Rubric 可以清楚地为开发人员描述如何评价他的工作以及他所期望的是什么。
- Rubric 可以提供测试程序，开发人员可以利用它度量和文档化进展过程。

可以以不同的形式和不同级别的复杂度来创建 Rubric；但是，它们都包括执行下列功能的一些通用特征：

- 集中测量声明的目标(性能、行为或者质量)
- 使用一个范围评价性能
- 包含特定的性能特征，这些特征有着不同的级别，用于说明标准所达到的程度

虽然 Web Rubric 可以从 Web 使能应用程序的性能标准中消除主观因素，但是可以采用其他重要的方法来定义 Web 使能应用程序的性能标准。

2.4 优秀性能的四种测试

Web 使能应用程序的性能可以通过如下 4 个方面进行度量：

- **并发性** 当一个以上的用户操作 Web 使能应用程序时所采用的度量方法。当 Web 使能应用程序能够处理大量并发用户对功能的使用和同时发出请求时，可以认为 Web 使能应用程序的并发性是优秀的。使用负载平衡设备通常可以解决并发问题。
- **延迟** 测量一个 Web 使能应用程序完成处理请求的时间。延迟有两种形式：Internet 网络中从浏览器到服务器的数据传输延迟，以及 Web 使能应用程序完成处理请求的软件延迟。
- **可用性** Web 使能应用程序可以接受请求的时间的度量。许多“高可用性”计算机产业软件发行商和硬件制造厂商会声称其可用性达到 99.9999%。作为一个可用性的示例，设想一个运行在服务器上的 Web 使能应用程序每周需要停机 2 个小时以进行维护工作。计算可用性的公式是(全部时间-停机时间)/全部时间。因为一周有 168 个小时(7 天×24 小时/每天)，每周 2 小时停机的结果是 98.8095%的可用性[(168 - 2)/168]。
- **性能** 两次失效之间时间总和的简单平均数。例如，一个应用程序在上午 10:30，11:00 和 11:30 出现错误，那么性能度量是 30 分钟。

通过几年实践，我发现很多时候这些术语被混淆而且被交叉使用。我也犯过类似的错误！记住这 4 个优秀性能的测试的简便方法就是 CLAP。CLAP 是并发性、延迟、可用性和性能英文单词首字母的缩写，这样可以帮助我记住测试方面的名字。

2.5 优秀测试代理的组成部分

Web Rubric 提供了测试 Web 使能应用程序的系统化方法。通过使用 Web Rubric 定义测试标准可以避免复杂性和主观臆断。利用现有的 Web Rubric，为了正确地实现标准，必须特别小心。主观臆断和松懈都会影响测试。

像古老的谚语所说：有垃圾进就有垃圾出(garbage in, garbage out)。

测试 Web 使能应用程序的现代表达方式可以为：谁在监管垃圾(who's watching for garbage)?

这是因为对于软件测试来说仍然没有明确的职业。目前，很少有人从事软件测试这样的职业。而且很少有测试人员希望整个职业生涯只做软件测试。我所认识的许多软件测试人员将他们的工作视为进入软件开发与管理的台阶。

为 Web 使能应用程序性能开发标准时，测试人员和测试软件需要说明很多特有的性质，包括严格性、系统化以及可重复性。这些特性无法确保成功。然而，识别出优秀软件测试人员和优秀软件测试代理的特性对于保证 Web 使能应用程序测试标准的正确测试将是一个重要的因素。表 2-2 说明了我所找到的在软件测试人员中的主要特性，以及这些相同的特性是如何在智能测试代理中发现的。

识别出优秀软件测试人员和优秀软件测试代理的特性对于保证 Web 使能应用程序测试标准的正确测试将是一个重要的因素。

表 2-2 测试代理和测试人员中发现的特性

智能软件测试代理	软件测试人员
严格性——智能软件测试代理是“多途径的”。当驱动一个 Web 使能应用程序时，如果一条路径在错误条件下中止了，严格的测试代理在放弃之前将尝试第二条或者第三条路径	**严格性**——测试人员的本能驱使他们通过任何方法改正软件工作中的错误。他们是有创造性的
系统化——测试代理自动运行。一旦安装并且指示它们做什么，它们会不遗余力地遵守测试标准	**系统化**——测试人员遵照测试标准以相同的过程启动和终止；然而，他们集中精力发现系统所没有要求查找的问题
可重复性——测试代理遵照详细的指令，它描述了系统运行测试标准所必须进入的状态	**可重复性**——测试人员每次运行测试标准的时候可以发现新的内容，即使每次测试都完全相同

2.6 Web 使能应用程序的类型

在 Internet 和 Web 发展的初期，Web 使能应用程序好像是软件开发的另一种形式。然而，现在，所有软件很明显都包括 Web 连接的形式。甚至 PC 视频游戏都包括接收软件自动升级的功能，而且玩家通过 Web 连接可以和其他玩家对战。

因为所有的软件都包括 Web 使能应用程序的性质，所以 Web 使能应用程序的软件分类也就消失了。这种状况对软件性能的开发标准有着深刻的影响。例如，为连接速度性能提出的单独标准已经消失了。Web 使能应用程序启动一个 TiVo 视频硬盘记录设备，它在每天早晨连接中心服务器并下载电视节目的收视指南。如果连接速度下降到平时传输速度的一半但仍能完成下载的进程，谁还会真的在意呢？连接速度性能标准仅仅用于视频硬盘记录设备。没有人会为视频记录设备和 PC 视频游戏开发两者都能使用的标准，因为它们根本就是软件的完全不同的分类，而且都有各自的性能标准。

那么，这就意味着每个软件程序都会有自己的标准吗？因为 Web 使能应用程序软件的主要分类出现了，每种分类将包括用于确定性能标准的资源和用于评价标准性能的测试工具。技术产业领导宣布集中在一起，目的仅是为了发现分歧以及异构系统中使用的数据中心，这样的机会我错过了很多次。

目前，Web 使能应用程序软件的主要分类有：

- 桌面产品软件
- 工具软件
- 电子商务软件
- 网络连接软件
- 协作软件
- 数据库软件
- 目录和注册软件
- 中间件
- 驱动、固件(firmware)以及工具

- 存储软件
- 图形和用户接口软件

每种类型都有各自的 Web 使能应用程序策略和性能标准。

在评价一个新软件类型时，应该查找有助于确定性能标准的一些工具。

- 随着系统的增长，诸如性能退化、损坏的功能、以及非认证的内容等问题的风险也随之增加。在系统增长的时候，需要确定测试工具可继续使用的能力。
- 单个工具的特征可以相互覆盖，但是每类工具被优化以解决不同类型的问题。确定测试工具最佳的组合以覆盖标准。
- 诊断软件用于定位损坏的功能、通过错误消息和故障代码保护用户。找到能很好处理错误条件的测试工具。
- 查找检查 Web 使能应用程序完整性的产品。在测试上下文中的完整性意味着安全和功能性检查。例如，可以登录并且访问无特权的控制吗？
- Web 使能应用程序的功能和内容通常需要分开的测试软件。例如，尽管一个按钮的功能可能正确，但是它的标签有可能拼写错误。另外，有具体需求的用户可以合理地要求应用程序的发布者检查应用程序的可访问性。例如，美国政府 508 部分的法案就是为有具体需求的人们提供全部访问性的众多规章之一。
- 复杂系统可以管理用户对数据和功能的访问。测试工具在执行 Web 性能标准测试的时候需要检查认证。
- 性能监测工具需要度量 Web 使能应用程序的性能。
- 跟踪浏览器行为的工具有助于优化 Web 使能应用程序的内容、结构和功能性。

最终，主要的 Web 使能应用程序分类成为了我们通用语言的一部分。表明完备性的 Web 使能应用程序将说明不需要该服务的印刷手册。相应地，服务的功能和使用将达到用户的一定期望值，因而也不需要指令了。

软件测试是用来模拟用户期望并且确保软件在一周 7 天、每天 24 小时满足这些期望的。为 Web 使能应用程序的性能开发标准是达到这个目标的关键所在。

2.7 Web 使能应用程序点系统

测试 Web 使能应用程序可以同时将一个开发小组引向不同的方向。例如，单元测试可以说明 10 个软件模块中的 3 个失效了，响应时间延长了 10 秒钟，而且 Web 使能应用程序每天可能有 7 次将系统的错误消息返回给用户。一个含有上述问题的 Web 使能应用程序说明其已经失效了。经验表明用户会发现无法接受含有上述特征的 Web 使能应用程序。

同时，这样的结果也会立即在许多方向上牵制开发管理员。在有限的资源下，哪个问题的优先级最高？Web 使能应用程序点系统(Web-enabled Applocation Points System，缩写为 WAPS)是一种好的解决方案，它可以帮助开发管理员理解 Web 使能应用程序的健康性而且可以将开发小组的精力集中在解决问题和维护代码上。当为了测试系统的性能而使用 Rubric 时，WAPS 是一种理想的度量。

WAPS 为概括 Web 使能应用程序的性能和可扩展性提供了一种快捷、方便的方法。通过降低点的数目，开发小组可以为用户增加 Web 使能应用程序的服务能力。在表 2-3 中，描述了点系统所包括的 3 种度量。

表 2-3 Web 使能应用程序点系统

标　准	描　述	点(每 24 个小时重新设置)
功能	错误状态、边界错误以及权限问题的数目	为说明功能性错误的每个日志入口设立一个点
性能	提交动态网页的秒数	为每个编译超过 10 秒钟的页面设置两个点
失效	用户看到错误消息的数目；例如，“对不起，数据中心发生数据库错误”	为 Web 使能应用程序失效设立 3 个点

每隔 24 个小时统计点的数目。利用表 2-4 中的分类，检查 Web 使能应用程序的健康性(health)。

表 2-4 点系统分析

点	健　康　性	应该采取的措施
1~10	优秀	今天可以放松了，明天再检查点的数目
11~25	较好	改正问题，以便达到优秀的标准；然而，需要将功能、性能以及失效的问题结合起来。应该避免只集中于上述问题的某一个分类
26~100	较差	Web 使能应用程序尚不能被用户使用。如果 Web 使能应用程序准备交付，那么应该返工，修改出现的问题，向用户解释这些步骤
≥101 个	极差	不能自暴自弃，这是交付健康的 Web 服务的方法。评价代码编写的水平，项目的复杂度，主要问题的范围。选择一周以后的某一天，看看是否将 Web 服务从极差程度提高到较差的程度。然后每天再检查点的数目

WAPS 提供了标准，它可以帮助开发人员理解软件项目相应的健康性——在初始开发阶段以及之后的维护阶段。开发小组的有效交流是 WAPS 成为高效工具的关键。可以自动生成 WAPS 报表，这样开发人员每天都可以获得这样的报表。WAPS 数据集合中包含的规则越少越好。

WAPS 的日常测量已经为开发人员提供了大量的信息；但是沿着时间轴作点数的图对于维护软件项目的健康性可能是没有意义的。PushToTest(一个软件测试自动解决方案的公司)使用 WAPS，并为 Web 使能应用程序记录随时间变化的数据，如图 2-3 所示。

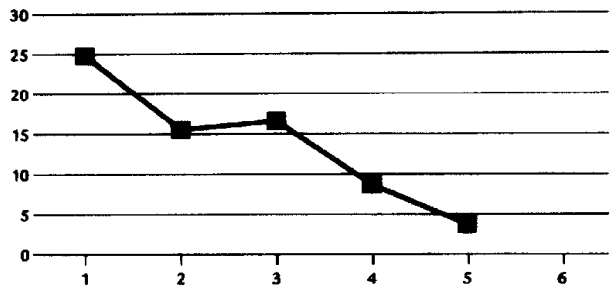

图 2-3　为一个网站作出 WAPS 系统结果随时间变化的线图

在每周的开发人员会议开始的时候，列出 WAPS 结果。PushToTest 工程师会注意到 WAPS 着重解决用户每天遇到的问题。因为 WAPS 是由很多不同权重类型的问题所组成的，所有开发人员将注意力集中在目标问题上，而不只是记录问题出现次序的列表。

那么，WAPS 如何将规模扩大到网站上呢？设想一下将 WAPS 应用在一个支持免费音乐下载的公共 Internet 网站。每月可能有数千万个用户访问这个网站。如果面对一千万个用户，Web 使能应用程序处有 120 个点并且处于极差的状态下，还会使用这个应用吗？答案是会的！可以根据所评估的系统规模为点表选择权重；然而，与根据百分比相反，统计点是很重要的。

由于 Web 使能应用程序处于极差的状态，遇到应用问题的用户无法满足个人的目标。WAPS 认为每个单独的用户都是成功的关键。当管理员忽视个体用户而着重解决重要的群体用户目标问题的时候，这时也可能是一名重要杂志记者、投资资本家、商业开发执行官、或者关键客户访问该网站的时候，那么，所有开发的努力以及公司的信誉都受到了损失。

WAPS 应该成为公司语言的一部分。即使对于公司外面的开发小组，公司也应该通过查看日常 WAPS 报告逐渐了解 Web 使能应用程序是如何执行的。即使根据软件遇到点的数目进行加班，为用户提供服务和支持的工作人员，甚至是市场销售小组都应该知道何时应该提供更多的资源。

WAPS 的另一个明显优势是它对开发人员有着影响力。Web 使能应用程序的构建、部署和管理是复杂、错综和令人厌烦的。从开发人员的角度上看，建立一个 Web 使能应用程序就好比是强迫攀登冰山——一旦打滑，一切就都完了。这使得许多开发人员过于谨慎而且太希望这些快点结束(当然，严格的管理员会根据 WAPS 给出诱人的工资！)。

开发人员将注意力集中在以下目标：

- **设计**——软件模块如何交互
- **特征**——执行用户驱动功能的代码
- **性能**——软件完成用户驱动命令的速度
- **可扩展性**——软件在不同使用级别上如何运行

明显缺少的目标是具有可用性的。由谁来负责用户使用 Web 使能应用程序的经历呢？

开发人员最好的愿望是交付可扩展、运行正常且有价值的软件。但是当一个软件项目超出预算，难以调度，以及过于复杂的时候，许多开发人员会变得保守而且稳健。开发人员甚至会陷入经常引用的谚语之中：质量、功能和调度，只能选择其中的两个。

在这样的情况下，我建议使用 WAPS。通过打分系统，开发小组就拥有了一个可以返回原始调度和合理管理预测的工具。无论项目的规模和复杂度有多大，WAPS 都提供一个可度量的平衡，以解决问题并交付具有完全可用性的已完成的 Web 使能应用程序。

2.8 Web 使能应用程序的框架

当系统可以从 1 个用户扩展到 10 000 个用户而且不需要改变软件的实现或者设计时，该 Web 使能应用程序就是成功的；当所有的用户都可以很好地使用，而且不管用户使用多少功能时，Web 使能应用程序就是成功的；当 Web 使能应用程序无论何时都能够为用户提供服务时，Web 使能应用程序就是成功的。在现实条件下，测试 Web 使能应用程序可以预测一个 Web 使能应用程序的成功性，还可以监测 Web 使能应用程序何时需要维护。为部署 Web 使能应用程序的系统，选择正确的组件可以有效地确定一个 Web 使能应用程序的成功性有多大。

记住上面的说明，本节将介绍用于建立和部署 Web 使能应用程序的架构，介绍如何为 Web 使能应用程序提供智能测试代理的指令，以及称为 TestMaker 免费开源工具集和脚本语言，TestMaker 有助于建立自动的智能测试代理。其结果是得到一个能够更快速开发软件的系统和方法学。

2.8.1 饼体系结构

在我作为 Web 系统的设计员和编码员期间，我发现了一个类比，它很恰当地描述了 Web 使能应用程序体系结构是要部署许多通过负载平衡器访问的小服务器，从而为前端提供一个功能强大的数据库服务器。这个类比如下：

饥饿的人(Web 用户)到(开发人员的)餐馆吃早餐。来吃早餐的人越多，就需要向平锅中加入越多的饼(服务器)。一些人想要吃香蕉味的饼，一些要草莓味的，而一些人想要香草味的普通蛋糕，还有要混合口味的。在 Web 使能应用程序环境下，一些用户要求访问运行 Java servlet 的服务器，一些用户需要应用服务器，一些需要.NET 和 SOAP 的 Web 使能应用程序，而一些用户希望使用目录服务器，但是大多数用户还是需要一些服务器的组合。

所有的饼(flapjack)都用相同的盘子盛放。在这个类比中，盘子的职责就是一个数据库，它支持永久存储、查询和索引等功能。

服务员接受用餐人的要求(用户的请求)并将它们交给正确的厨师(好比是制作草莓饼的厨师)，这更像是负载平衡器将 Web 浏览器路由到正确的服务器上(如果必要，可以是备用服务器)。

图 2-4 说明了“饼体系结构。”

这种类型的体系结构很流行。在 eBay、Yahoo!、HotMail 和其他大型公共 Web 站点提供服务的系统都是根据饼体系结构设计的——大多数称为N层体系结构。企业应用软件提供商(包括 Microsoft 的.NET 架构、IBM 的 WebSphere、BEA 的 WebLogic)都建议使用饼体系结构。

饼体系结构为 Web 使能应用程序，特别是专用内部网络系统带来了许多优势。用户更愿意从小的、廉价的服务器上获得更快的性能和更少的等待时间。在一批小的服务器中，负载平衡系统能够使一组服务器保持在就绪状态——加载线程并运行，内存分配，准备数据库的连接——从而当下一个功能请求到来的时候，小服务器可以立即提供服务。

因为任何时刻只有极少数的线程在运行，所以软件工程师可以很容易地进行调试。大规模多处理器系统在已安装的处理器之间管理并协同线程。廉价的小服务器只运行本地 Web 使能应用程序所需的线程。

因为公司财务管理员可以购买大量廉价的小服务器而且可以避免大系统的花销，所以他们也愿意使用饼体系结构。在计算机产业中，这类小服务器很快占领了市场。低端服务器定价使

得财务管理员与服务器制造商有了更多协商的余地，因而可以构建并且装配具有大内存和硬盘容量的多处理器的服务器。

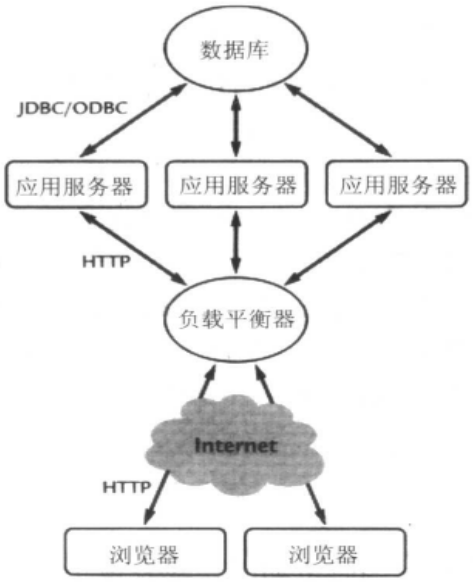

图 2-4 饼体系结构

最后，因为所有的小服务器都提供了很大的灵活性，所以网络管理员也喜欢使用饼体系结构。正如一个 IT 管理员所说，“如果系统在某个早上坏了，我可以到邻近的计算机大超市，买到替代的部分。”另外，小服务器可以方便地替代其他的损坏设备。在交换过程中，负载平衡器将数据流引向其他正在工作的服务器。

2.8.2 采用饼体系结构和智能测试代理

虽然饼体系结构和智能测试代理对于建立可扩展的 Web 使能应用程序是非常理想的，但是这种理想并不是在业务的不同要素中都是那么明显。新公司也要求一些销售中要带有饼体系结构/智能测试代理服务器基础架构的策略。

表 2-5 考虑了用户以及他们的目标。理解饼体系结构是如何影响公司中的主要成员小组，对于成功使用它是非常关键的。

表 2-5 饼体系结构的用户和目标

用　户	需　求	问　题
开发人员	单元测试的新源代码模块	在正常的状态和输入情况下，模块能返回正确的结果吗
质量认证技术员	模拟真实的产品加载环境	应用的性能和可扩展性方面如何？什么特征使得之前修复过的错误又重新出现呢
IT 管理员	描述一个新的 Web 使能应用程序数据中心的需求或者对一个 Web 使能应用程序的更新	数据中心会满足该应用的需求吗？可以使用监测器查看 Web 使能应用程序失效的情况吗

饼体系结构与智能测试代理的结合是满足每个用户需求的有效解决方法。例如，开发人员使用智能测试代理可以将应用程序返回到已知的状态。质量认证(QA)管理员使用智能测试代理可以查找再次出现但已修复的错误。而且产品管理员使用测试代理可以保证数据中心服务器的安全性，从而支持用户的需求。

1. 开发人员

Internet 软件开发生命周期为软件测试带来了一个新概念：快速应用程序测试。桌面应用程序和 Web 之前的应用软件将软件开发小组与直接来自用户的响应相隔离。随着开发人员与用户之间的隔离逐渐在 Internet 上消失，Web 使能应用程序的发行商可以直接从用户那里得到软件的错误。用户通过电子邮件消息提出错误，或者通过鼠标的点击直接离开这个 Web 使能应用程序。在 Internet 软件开发生命周期中，在用户之前发现错误对于商业的成功运作起到越来越重要的作用。

当第一批用户错误报告到来的时候，习惯于 Internet 之前软件开发生命周期的开发人员通常会感到震惊。公司希望对用户错误报告进行实时处理。使用智能测试代理的开发人员可以查看以前的代理日志，从而可以确定问题在代码中的根源以及位置。不使用智能测试代理的开发人员只能通过查看 Web 使能应用程序的日志文件获得帮助。

Web 使能应用程序服务器与智能测试代理两者创建的日志文件之间是有所不同的。

```
127.0.0.1 - - [22/Jan/2002:00:12:41 -0600]  "GET /mailman/
listinfo/oukids HTTP/1.1" 200 5358
127.0.0.1 - - [22/Jan/2002:00:12:41 -0600]  "GET /icons/mail-
man.jpg HTTP/1.1" 200 2022
127.0.0.1 - - [22/Jan/2002:00:12:41 -0600]  "GET /icons/gnu-
head-tiny.jpg HTTP/1.1" 200 3049
127.0.0.1 - - [22/Jan/2002:00:12:41 -0600]  "GET /icons/
PythonPowered.png HTTP/1.1" 200 945
127.0.0.1 - - [22/Jan/2002:00:14:27 -0600]  "GET / HTTP/1.1"
200 1489
```

这是标准的 Apache 风格的日志文件。该文件包括服务器处理的每个请求的入口。试想仅仅有日志文件就可以帮助减少并且修改服务器的错误吗？

智能测试代理根据原型用户的需求进行建模——就像是第 1 章的 Madeline 或者本章之前的 Ann。测试代理日志为查找、修复错误提供了更多的信息。

```
Agent name, Task,  Results, Module, Date:Time

Madeline,  Sign-in, Ok,  com.p2t.signin,  10:21:22:10
Steps taken: chose random user,  lookup user id/password,
sign-in, all modules ran within tolerances

Madeline, Find product 7, OK,  com.p2t.findit,  10:21:22:31
Steps taken: chose product previously Ordered,  find product
module notes: At 80% of catalog capacity warning

Madeline, Choose product 7, OK,  com.p2t.formchk,  10:21:22:45
Steps taken: Check credit module warns of credit overrun
```

```
Madeline, Add quantity, OK, com.p2t2.quanchk, 10:21:23:15
Steps taken: Add product to shopping basket module. All mod-
ules run successfully

Madeline, Sub quantity, Fail, com.p2t2.subchck, 10:21:23:25
Steps taken: Subtract product from shopping basket module.
Shopping basket module returned invalid integer value 871
```

Apache 网络服务器日志文件是面向点击流的，它回答这样的问题：Web 使能应用程序处理什么样的请求？智能测试代理日志文件是面向用户目标的，它回答这样的问题：用户达到其目标了吗？

查看日志文件对于合理维护 Web 使能应用程序是非常重要的。利用智能测试代理，可以将日志文件数据汇总成一张可见图表(meter)。图 2-5 说明了这种图表的形式。

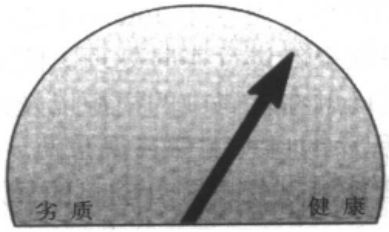

图 2-5 本图总结了运行智能测试代理的结果，直观地说明了 Web 使能应用程序的健康性

因为智能测试代理为需要执行新源代码模块单元测试的开发人员提供了有价值、适时的数据，所以智能测试代理使 QA 管理员能够模拟真实产品的负载环境，下面将会对此进行介绍。

2. 质量认证技术员

最好的 QA 技术员是有别于软件开发人员的。他们将用户的目标记在心中并且使 Web 使能应用程序达到这样的目标。QA 技术员每天的大部分工作看起来都很无聊，但是许多 QA 技术员很喜欢发现问题，而且愿意为开发人员提供解决方案。很多优秀的 QA 技术员会为开发人员指出引发问题的代码，因此 QA 技术员必须同时熟知代码、错误和可用性。

因为 Internet 软件开发生命周期要比其他非 Internet 软件开发生命周期更强调代码文档，所以软件模块的功能是 QA 技术员频繁遇到错误和问题的地方。开发人员对一个模块进行改动，编写改动的标注，然后将代码交给 QA 技术员。标注通常可以有不同的解释，因而可能会带来错误。

试想一个模块新的改动标注常常伴随着开发人员编写的智能测试代理。测试代理在描述模块上要比改动标注的功能更多，更加准确。相应地，测试代理成为了在开发人员与 QA 管理员之间有效的通信手段。

3. 信息技术(IT)管理员

IT 管理员在初始的开发结束之后开始维护系统。IT 所关心的就是维护和监测问题。

即使 Web 使能应用程序只有一行代码，服务也需要不断地维护。Web 使能应用程序开发人员的工作会受到业务变化的影响。即使原来的开发人员还在继续开发，IT 管理员有些时候也

需要调试 Web 使能应用程序中遇到的问题。设想对于 IT 管理员哪种情形会更好些：是通过查看源代码的注释查找有故障的模块，还是通过阅读智能测试代理查看模块应该如何执行？智能测试代理为开发人员向 IT 管理员说明 Web 使能应用程序和软件模块的正确行为提供了一种功能强大的方法。

监测 Web 使能应用程序需要智能测试代理。在 Internet 的初期，如果 Web 使能应用程序返回一个页面，那么就认为它在运行。当返回的页面中包含错误消息或者无效数据的时候，会发生什么情况呢？智能测试代理了解 Web 使能应用程序功能的上下文，可以向 IT 管理员发出警告——通过电子邮件、传呼或者日志文件中的记录。

智能测试代理是开发人员、QA 管理员和 IT 管理员之间的通信媒介，它也是开发和建立 Web 使能应用程序非常有效的方式。将每个人的注意力集中在智能测试代理上，所有小组成员都知道智能测试代理可以提交新的代码——它首先能够支持灵活(扩展/XP)的软件开发的测试方法。一旦某个组织采用了测试代理，那么所有的开发项目都将包括测试代理。

一个组织选择的测试工具决定了小组持续应用测试代理的可能性。当测试工具受到 QA 技术员的青睐但没有得到开发人员的认可时，持续应用智能测试代理的可能性不大。

2.9 在饼环境下建立智能测试代理

将一个 Web 使能应用程序移植到产品环境中需要保证高可用性和持续的优良性能。测试代理从它们创建开始时就要建立这些保证。因此，测试工具的选择就很重要了。当为测试 Web 使能应用程序选择工具的时候，开发人员、QA 管理员以及 IT 管理员需要记住下面推荐的这个检测列表：

- **状态测试**——当使用 Web 使能应用程序设置数值的时候，服务器稍后是否响应正确的结果？
- **权限测试**——当每个用户都尝试访问只为管理员授权的一个控制项的时候，会发生什么情况？
- **速度测试**——Web 使能应用程序响应的时间是否很长？
- **边界定时测试**——当使能应用程序需要超时或者的确需要很长的响应时间时，会发生什么情况呢？
- **回归测试**——一个新功能破坏了现有的功能吗？

对每个软件应用来说，这些都是很平常的测试。为理解在不同级别负载下的 Web 使能应用程序的性能特征，表 2-6 显示了测试矩阵的一个测试模板。

表 2-6 测试矩阵模板

测试类型	1 个用户	50 个用户	500 个用户
状态测试	❑	❑	❑
权限测试	❑	❑	❑
速度测试	❑	❑	❑
边界定时测试	❑	❑	❑
回归测试	❑	❑	❑

在评价商业测试工具的时候，许多开发人员都很有些怀疑。总是带有这样的想法：我自己就可以编写测试代理，公司为什么要花费这么多钱买测试工具，为它还要学新的指令和语言呢？

自己编写测试工具带来的问题归结为两个字：维护。就像其他的软件应用，测试工具也需要维护。当测试工具的开发人员不在公司工作的时候，谁来维护这些代码呢？

自己编写测试工具的硅谷开发人员作出了一句简短但并不很科学的总结，维护工具由一些小的烦恼可以发展到令人头痛的大问题。普通开发人员编写工具的初次尝试一般是 Java 类或者 VB 方法的鲁棒性集合，它们向 Web 服务器发送 HTTP 请求，请求饼环境执行一些结果的简单分析。编写一个测试代理包括按顺序编写调用正确测试对象的 Java 代码。

例如，从在线论坛消息集合中读取信息的测试代理的可能形式如下所示：

1. 读站点的第一页(包括讨论消息的 URL)。
2. 随机读出一个讨论消息。
3. 如果该消息有回应，读取回复消息。
4. 重复第 3 步直到没有任何回复消息。

因而，将测试代理编写成一个小的 Java 应用程序。在一些点，读取站点首页的 URL 形式可以变化。这需要对 Java 测试代码进行维护。

作为基本事实，对 Web 使能应用程序的每个新改动都会带来测试代理的一些小变动。每个改动都会使开发人员重新查看测试代理的代码。因为测试对象——获得 Web 页面、登录、测试正确的返回结果——在不同的测试代理中都是一样的，所以调用次序就应该有所不同。

遗憾的是，Java 和 VB 都不能将一个脚本模型定义为设计的一部分。在脚本模型中，功能组以根据参数评估的发生分支的具体顺序进行调用。利用 Java 和 VB 使得程序员对具体顺序调用对象的方法负责。因此，许多程序员在他们代码内开发了一种简单的脚本语言，它可以实现业务规则或者代理的逻辑。一些更高级的应用甚至将脚本语言提供给用户。在整个计算过程中，上千的应用程序逐个地使用它们自己的脚本语言。

查看测试代理的代码，将脚本语言添加到测试对象库中的装配测试代理会有意义吗？测试对象执行很少需要变动的个体测试程序。脚本语言将用来装配并控制对象的参数——脚本语言可以很容易地改动不同级别和不同类型的 Web 使能应用程序的测试。

对所有开发人员、QA 技术员、IT 管理员来说，其好处是程序员一次可以编写更多的测试对象并且根据需要改变脚本语言。脚本语言使得开发人员、QA 技术员、IT 管理员更方便地编写他们自己的测试代理，而不是使用核心的面向对象的代码去编写。脚本容易读懂，因而可以方便地和其他脚本进行共享与交换。

现代脚本语言，例如 Jython(详见 www.jython.org)使得 Web 使能应用程序出现本书所描述的测试代理形式。脚本为测试代理脚本的编写方式、变量和标注的程序流程提供了一种通用的参考。

2.9.1 脚本语言和测试代理

多数软件工具的成熟在于充分考虑了脚本语言(很多时候，脚本语言的出现标明了它是成熟工具)。然而，使用脚本语言的不利之处在于每个脚本语言只能应用在它自身的小范围之内。相应地，可以发现公司中每个组都使用了不同的脚本语言和工具。表 2-7 说明了几种常用的选择。

表 2-7　测试和监测 Web 使能应用程序的脚本语言和工具的示例

软件开发人员	QA 技术员	IT 管理员
Java 对象	Python	Perl
jUnit——单元测试框架	OpenSTA	PHP
C#	TestMaker	HP OpenView 管理控制台
VB.NET	Ruby	Intermapper

表 2-7 只是目前使用 Web 使能应用程序的公司中涉及的许多脚本语言和工具中的很小一部分。因此，寻找每个人都同意的单独脚本语言和测试工具的可能性是非常小的。然而，对交付运行在通用实时环境下的脚本语言功能有着增长的趋势。Jython 就是这个趋势的一个很好的示例。

Jython 是用 Java 实现通用的 Python 编程语言的一个开源项目。它将 Python 脚本命令编译成 Java 字节代码。Java 字节代码是 Java 用于处理本机命令的中间语言。Java 虚拟机在本机处理器上执行字节代码。字节代码使得 Java 应用程序可以在任何运行 Java 的主机上运行。如图 2-6 所示，使用通用运行环境的方法也可以用于其他脚本语言。

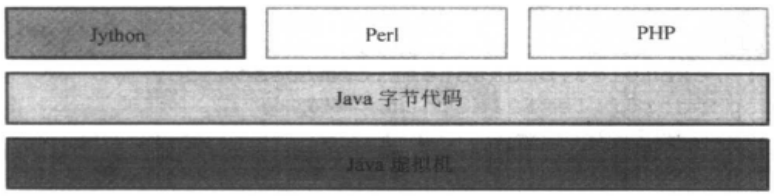

图 2-6　将脚本语言用于通用运行环境有着增长的趋势。Jython 是这种可能的一个示例。它将 Python 脚本命令编译成 Java 字节代码，因而可以在任何运行 Java 的主机上运行。利用相同的方式，Perl、PHP 以及其他脚本语言在相同的通用运行环境下也可以使用

使用通用运行环境的最大优势是脚本语言命令可以访问本地的脚本语言对象，也可以访问运行环境所提供的任何公共对象。在 Jython 的示例中，任何 Jython 脚本都可以使用所有的 Python 对象，同时脚本还可以访问任意 Java 对象。这在 Jython 脚本编写人员和 Java 对象开发人员之间架起了一座桥梁。

为了查看 Jython 和 Java 对象的协作，下面的脚本是一个使用两者的测试代理的示例。

在阅读下面的脚本时，要记住如果对脚本或者软件代码不感兴趣，那么诸如 TestMaker 这样的测试工具将提供图形化界面和测试代理记录工具，从而不必编写脚本代码。但是，理解一个典型测试代理脚本的功能则有助于了解测试工具的图形化界面所提供的基本功能。

```
# Agent name: Example Form Test
#
# Purpose:
# Calculates a Transaction-Per-Second index for a
# Web-enabled HTTP application

from com.pushtotest.tool.protocolhandler import \
  ProtocolHandler, Header, Body, HTTPProtocol,  \
  HTTPBody, HTTPHeader
from com.pushtotest.tool.response import Response,  \
```

```
  ResponseLinkConfig,  SimpleSearchLink
from java.util import Date
from java.lang import Exception

# The test class implements the steps recorded while
# watching a user operate the application using a browser

class test:

    # Set-up
    def _init_(self):
        self.steps = 0          # Steps in transaction
        self.errors = 0         # Counter of errors
        self.passed = 0         # Value is 1 if no exceptions

    # Test
    def run(self):
         self.steps = 0;
         start_time = Date().time
          http = ProtocolHandler.getProtocol("http")

    # Request the welcome page with the html form

    http.setUrl( \
       '''http://examples.pushtotest.com/responder \
       /htmlresponder?file=file2.html''')
    http.setType( HTTPProtocol.GET )
    response = http.connect( 0 )
    self.steps += 1
    code = response.getResponseCode()
    if code <> 200:
         print  '''Error response code:''  ,self.code
         self.passed = 0
         self.errors += 1
    else:
         self.passed = 1

    # Submit the form values to the host

    http.setUrl( \
      '''http://examples.pushtotest.com/responder \
      /htmlresponder''')
    body = ProtocolHandler.getBody(  'http'  )
    http.setBody( body )
    http.setType( HTTPProtocol.POST )
    body.addParameter('''firstname''',  '''Frank''')
    body.addParameter('''lastname''',  '''Cohen''')
    body.addParameter('''phone''',  '''408 374 7426''')
    body.addParameter('''account''',  '''10381838''')
    body.addParameter('''amount''',  '''1838.12''')
    body.addParameter('''Transfer''',  \
      '''Transfer Funds''')
    response = http.connect( 0 )
    self.steps += 1
    code = response.getResponseCode()
    if code <> 200:
        print '''Error response code:''',self.code
```

```
            self.passed = 0
            self.errors += 1
        else:
            self.passed = 1

    # Getter methods for various timing values
    def getTotaltime(self):
        return self.totaltime

    def getSteps(self):
        return self.steps

        def getTransactions(self):
            return self.transactions

        def getErrors(self):
            return self.errors

        def getPass(self):
            return self.passed

        # Clean-up
        def cleanup(self):
            print "Done."

    # Transactions-per-second Index Test
    # Run the test 100 times and then print the results
    print "  ..................................................  "
    print "Transactions-per-second Index Test"
    print "brought to you by TestMaker from PushToTest"
    print
    print "  ..................................................  "
    print "Starting test"

    t = test()                      # Create a test object
    transactions = 0                # How many transactions completed
    exCount = 0                     # Count of exceptions thrown
    count = 50                      # Number of times to run the test

    starttime = Date().time         # record the current time

    print "Test running"
    print

    for i in range( count ):
        print  "-",
    print

    for i in range( count ):
        try:
            t.run()
            transactions+=1
            print "-",
        except Exception,  ex:
            print "Encountered an error:",ex
            exCount += 1

    totaltime = Date().time - starttime
```

```
print
print
print "................................................."
print "Results"

tps = float( transactions ) / float( totaltime )  * 100
print "Transactions Per Second (TPS):  %9.2f" % tps
print "Total transactions processed:  ",  transactions
print "Total time to process all transactions:  ",  totaltime
print "Each transaction had",  t.getSteps(),"steps"
print "Exceptions thrown during the test:  ",  exCount
print "Errors encountered during the test:  ",  t.getErrors()
print

t.cleanup()
```

由于脚本格式的简明风格，像 Jython 这样的脚本语言使得测试代理逻辑更加明显。因为脚本语言关心测试代理的逻辑，所以底层的测试对象可以处理与主机服务的实际通信。

所给出的示例利用了脚本语言和 TestMaker 中的测试对象库，TestMaker 是测试 Web 使能应用程序的性能和可扩展性的一个开源工具。TestMaker 和本书的所有示例都可以通过 http://www.pushtotest.com 免费下载，而且使用 Apache 风格的免费开源许可证。

为了在建立测试代理的时候更好地理解脚本语言的功能，请详细地查看下面这个测试代理脚本：

```
from com.pushtotest.tool.protocolhandler import \
   ProtocolHandler,  Header,  Body,  HTTPProtocol,  \
   HTTPBody,  HTTPHeader
from com.pushtotest.tool.response import Response,  \
   ResponseLinkConfig,  SimpleSearchLink
from java.util import Date
from java.lang import Exception
```

import 语句标识了脚本语言编译器的测试对象库。测试对象库处理与 Web 使能应用程序主机通信的很大一部分。例如，HTTPProtocol 是通过 HTTP 协议处理与 Web 主机通信的对象。另外，该脚本使用了 Java 的 Date 对象，并且通过 Java 的 Exception 对象报告错误。

```
class test:

   # Set-up
   def _init_(self):
       self.steps = 0          # Steps in transaction
       self.errors = 0         # Counter of errors
       self.passed = 0         # Value is 1 if no exceptions
```

该脚本使用 Jython 的面向对象方法创建一个名为 test 的新类。测试类初始化 3 个新变量：steps、errors 和 passed。steps 变量跟踪在每个事务中所使用的 HTTP 请求数目。在本例中，事务由两个步骤组成：获得一个包含表单的 Web 页面，然后将表单的数值提交给主机。

```
def run(self):
        self.steps = 0;
        start_time = Date().time
        http = ProtocolHandler.getProtocol("http")
```

测试类只有一个名为 run 的方法。该方法通过完成单个事务从而测试主机。我们会进一步介绍测试代理脚本如何多次调用 run 方法以获得系统评价的吞吐量。

run 方法首先使用 TestMaker 提供的 ProtocolHandler 对象得到一个新的 HTTP 协议处理程序对象。协议处理程序会与使用 HTTP 的 Web 服务器进行通信，获得一个带有 HTML 表单的页面。用浏览器查看下面的 URL，能更好地理解这个 Web 使能应用程序：http://examples.pushtotest.com/responder/htmlresponder?file=file2.html。

```
http.setUrl( \
    '''http://examples.pushtotest.com/responder \
    /htmlresponder?file=file2.html''' )
http.setType( HTTPProtocol.GET )
response = http.connect( 0 )
self.steps += 1
code = response.getResponseCode()
if code < > 200:
    print  '''Error response code:''',self.code
    self.passed = 0
    self.errors += 1
else:
    self.passed = 1
```

这个脚本告诉 http 对象使用所提供的 URL，并且使用 HTTP Get 协议。http 对象提供了一个 method 方法，该方法返回一个 response 对象。response 对象包含从主机那里接收的 HTML 代码、HTTP 头信息以及 cookie 值。response 对象提供了关于有效响应数据的许多方法，包括确定从主机接收的结果码返回值的 getResponseCode 方法。HTTP 协议将响应码 200 定义为正常响应。在响应码大于 200 的时候，脚本将响应码打印到输出面板中。

```
http.setUrl( \
    '''http://examples.pushtotest.com/responder \
    /htmlresponder''')
body = ProtocolHandler.getBody( 'http' )
http.setBody( body )
http.setType( HTTPProtocol.POST )
body.addParameter('''firstname ''', '''Frank''')
body.addParameter('''lastname ''', ''' Cohen''')
body.addParameter('''phone ''', ''' 408 374 7426''')
body.addParameter('''account ''', ''' 10381838''')
body.addParameter('''amount ''', ''' 1838.12''')
body.addParameter('''Transfer ''', \
    ''' Transfer Funds ''')
response = http.connect( 0 )
```

下一组脚本命令执行 Web 使能应用程序主机的 POST 命令。这相当于用户填写一个网页上的表单项，然后单击 Submit 按钮。

```
# Getter methods for various timing values
    def getTotaltime(self):
        return self.totaltime

    def getSteps(self):
        return self.steps
```

```
    def getTransactions(self):
        return self.transactions

    def getErrors(self):
        return self.errors

    def getPass(self):
        return self.passed

    # Clean-up
    def cleanup(self):
        print "Done."
```

这些命令将几个帮助方法添加到测试类中。特别地，它们为测试类提供了向其他类返回测试结果数据的方法。我们将在下一部分代码中介绍它们。它们包括需要创建测试类的脚本代码。现在，我们开始使用测试类了。

```
t = test()          # Create a test object
```

该测试脚本实例化一个测试对象，该测试对象通过使用变量进行引用。用 Jython 创建对象就是这么容易！

```
transactions = 0          # How many transactions completed
exCount = 0               # Count of exceptions thrown
count = 50                # Number of times to run the test
starttime = Date().time   # record the current time
```

测试脚本建立了几个变量，它们稍后将会报告出以事务/秒为单位的主机吞吐量。

```
for i in range( count ):
    try:
        t. run ( )
        transactions+=1
        print  "-"  ,
    except Exception,  ex:
        print "Encountered an error:",ex
        exCount += 1
```

该测试脚本使用 for 命令重复调用测试对象中的 run 方法。Count 变量的初始值为 50，因此测试代理将运行测试类 50 次。

如果还想知道测试脚本的格式，Jython 利用间距为代码块划定界线。间距是很重要的。如果想要马上使用这个脚本，本人建议到 http://www.pushtotest.com/ptt 上进行下载，从而可以节省输入这个代理脚本的时间。

```
tps = float( transactions )  / float( totaltime )  * 100
print "Transactions Per Second  (TPS):  %9.2f"  % tps
print "Total transactions processed:  ",  transactions
print "Total time to process all transactions:  ",  totaltime
print "Each transaction had",  t.getSteps(),"steps"
print "Exceptions thrown during the test:  ",  exCount
print "Errors encountered during the test:  ",  t.getErrors()
```

最后，这个代理脚本在事务/秒的报告中计算并打印测试的吞吐量。

2.9.2 产生有意义的数据

智能测试代理通过像用户使用那样的操作软件对 Web 使能应用程序进行测试。对于 Web 使能应用程序来说，来自测试代理的请求被发送给合法的用户。Web 使能应用程序根据请求运行并且返回数据。多个并发运行的测试代理可以模拟用户在同一时刻访问系统的真实产品环境。所有这些操作创建了下面形式的原始数据：

- 测试代理创建日志文件以存储请求的细节，包括需要多久建立和处理请求，返回多少个错误。
- Web 使能应用程序的主机创建日志文件，从而说明处理请求的数目以及一些结果的细节信息，包括超时和断开连接。
- Web 使能应用服务器的硬件包括存储使用信息的操作系统代理(通常使用简单节点监测协议，或为 SNMP)，它包括网络带宽、硬盘空间、CPU 的使用和内存的使用。
- 网络路由器和其他基础架构创建跟踪资源分配和使用服务的日志文件。

每个原始数据都可以成为智能测试代理技术的有效数据。有效利用大量原始数据的关键是定义良好运行系统的参数。Web 使能应用程序点系统是一个很好的方法学。最后，当自动技术完成 Web 使能应用程序中监测并观察性能优劣的必要任务时，就产生了有意义的数据。

2.10 本章小结

随着 Internet 技术、带宽、服务级别的提高，Web 使能应用程序开发项目有了明显的进步。现代 Web 使能应用程序的开发小组包括了通过使用新的软件开发工具、新的软件开发生命周期以及编写软件的新方法学而达到新的生产水平的开发人员。软件测试工具、技术和方法也相应地发展。开发人员、QA 技术员、IT 管理员发现软件测试必须是智能的，否则测试结果会没有任何意义，甚至产生误导。新的软件测试技术帮助开发人员更好地使用户达到他们的重要目标。

本章介绍的新工具为确定 Web 使能应用程序的健康性提供了易于遵守的标准，例如 Web Rubric 和 Web 使能应用程序点系统。新的数据中心框架和开源测试自动工具 TestMaker 为建立在真实产品环境下可扩展和可执行的 Web 使能应用程序提供了简单、方便的方法。

工具、基础架构和方法学的选择对测试代理在高级服务上定位问题并且监测 Web 使能应用程序上带来了很大的不同。经验表明，当现代的 Web 使能应用程序开发小组达到了测试工具的这些目标时，Web 使能应用程序也会满足用户的目标。

- 不论工具的价格和公司的标准如何，软件开发人员、QA 技术员、IT 管理员需要采用正确的工具。
- 测试代理需要智能化。
- 测试 Web 使能应用程序和基于 SOAP 的 Web 服务需要智能驱动应用的测试，就像用户平时使用的情况一样。
- 测试代理需要在接近产品负载的情况下测试系统的可扩展性和性能。
- 要使用测试工具。

在本章中，我们学习了如何在集成了 Web 使能应用程序的系统中量化系统的性能。在第 3 章中，将介绍功能、可扩展性和性能的测试与用户的成功是如何联系在一起的。

第3章 模拟测试

“我不作测试，”这是我的一个朋友对我说过的一句话。这并不是因为她反对测试。她可能认为软件设计和体系结构是同一回事。以她的观点，测试只是测试员处理的晦涩、难学的过程。如果您还不熟悉测试，本书采用了每人测试(everyone tests)的思想对测试进行讲解。我相信每个人——用户、管理员、软件开发人员、体系结构设计人员、质量保证技术员、网络管理员和管理人员——在他们每次使用 Web 使能应用程序的时候都会进行测试。

在分布式和互操作的 Web 使能应用程序环境下，其窍门是发现一种将专门测试变为有用、有意义的数据的方法。有用并且有意义的测试数据对于一些问题的排序很有价值，这些问题包括：需要解决的错误和疑问、确定在维护和升级方面的预算、能为用户提供更好的使用环境。

为了产生有意义的测试数据，测试必须确定在使用软件的时候满足用户需求的程度是什么。本章将说明如何模拟用户的行为，以及如何实现在智能测试代理中的行为。智能测试代理是自动化测试组件最有效的方法。理解为建立智能测试代理使用的测试环境和测试工具是理解测试组件结果的必要前提。

第 2 章为描述执行 Web 使能应用程序的有意义测试所需的因素提供了基础；本章将继续说明所有的参与者——服务器、路由器、数据中心、Internet 服务提供商(ISP)和软件——在交付高可用、运行良好的 Web 使能应用程序中是如何扮演各自的角色的。本章并不描述一个单独的方法，而是描述可以应用在很多测试情况下的方法学，并且说明在确定基础架构需求和测试方法学上人们是如何起到关键作用的。

3.1 为有意义的测试结果模拟用户行为

业务需要知道建立、部署和维护 Web 使能应用程序的解决方案，而测试方法学和策略给出了解答。例如，一个业务可能会问道，“我需要再买几个服务器才能满足新客户的需求？”电子表格和模型可以为数据中心的需求提供主要的预测；然而，模型最终需要硬数据来验证每一个假设。由模型引出问题的答案通常需要描述操作环境的数据——数据中心、用户界面与部署技术。测试 Web 使能应用程序为规模、带宽和功能问题提供了有意义的解答。

业务开发和交付 Web 使能应用程序遇到了相同的问题。表 3-1 列出了我最频繁回答的问题。

表 3-1 在开发 Web 使能应用程序时常询问的问题

问 题	所需的数据	使用的测试解决方案
在建立了数据中心而且设置了使用预测的时候，您的数据中心可以使用吗	测试从而确定数据中心是否就绪	使用智能测试代理确定 Web 使能应用程序的可扩展性和性能指数(SPI)。SPI(本章稍后会说明)测试数据中心是否就绪
数据中心需要什么组件和带宽	测试从而估计数据中心的需求	使用智能测试代理的 Web 使能应用程序仿真将给出答案(测试代理将在本章稍后描述)

(续表)

问　题	所需的数据	使用的测试解决方案
当用户将一个物品放在购物篮中时，它一个小时之后还会在吗？该物品没有出现在您的购物篮中，反而出现在别人的购物篮中吗？系统允许这种权限错误吗	测试以发现状态和边界问题	智能测试代理的单元测试有利于利用有效或者无效数据测试 Web 使能应用程序的功能。结果说明了 Web 使能应用程序的一些部分不能正常工作。智能测试代理自动化单元测试从而使测试流水化并且降低测试开销
当需要更高期望的用量时，Web 使能应用程序将如何运行	测试从而为更高期望的容量作好准备	并发运行的智能测试代理网络将说明在过度使用的情况下，Web 使能应用程序如何作出处理
因为软件需要维护，原有的错误可能又会出现。什么原因使得修改过的错误再次出现了呢	测试从而发现软件的退化	智能测试代理通过检查 Web 使能应用程序的每一个功能来监测它。当使用新软件时，监测器测试以前可用的功能是否继续可用

测试 Web 使能应用程序在解决公司业务问题上具有很重要的作用。通过确认测试是如何解决业务问题的，测试人员可以了解到对重要问题有价值的答案。

几年来，我懂得了设计高质量的 Web 使能应用程序系统需要测试与维护。良好的系统设计需要解决下列主题：

- 组件失效的频率是多少？有没有临时可以替代的部分？替代一个组件需要哪些步骤？
- 维护系统需要哪些预计的步骤？例如，一个数据密集型的 Web 使能应用程序需要索引，将数据重建成表格以得到空闲的磁盘空间和内存。如果重建索引，系统会对用户还可用吗？
- 新组件添加到系统的哪个部分？是否需要更多的物理空间才能容纳新的计算机硬件？在哪里安装新软件？
- 如果经常评价效率和性能，改进哪些方面会有更好的结果？我们希望改进内存、CPU 和存储技术。系统应该做出具有这些改进的设计吗？

理解开发 Web 使能应用程序的生命周期对于回答业务问题以及准备维护具有全局意义。

3.2 生命周期、项目和人的本性

人的本性在决定基础架构需求和测试方法学上扮演着很重要的角色。人是根据过去的经验、可信度、对事实的理解、数据表示的风格以及其他一些因素作出决定的。当我们设计产品生命周期、新的体系结构和测试的时候，应该记住人的本性。例如，设想在一家运输公司里的一名网络管理员。公司决定使用 Web 使能应用程序公布费用并且调度信息，这些信息放在一个已建立的数据库驱动系统上而且用户可以通过呼叫中心进行访问。公司需要估计购买服务器的数量，数据中心的 Internet 带宽。作为网络管理员，设想将以松散且专门的方式收集到的测试结果数据交给上级管理员，这才是一个严谨、有等级的管理风格。

通过理解业务管理风格，我们可以进行最高效管理的测试。在本节的稍后部分，定义了 4 种管理风格并且说明它们对设计和测试有什么影响。

以我个人经验，大多有意义的测试数据来自使用理解性很好的软件开发生命周期的测试小组。Web 使能应用程序的软件开发作为项目管理，在生命周期内进行开发。项目需要定义人、工具、目标和调度。生命周期描述通用的 Web 使能应用程序项目的标记和检测点。

Web 使能应用程序借鉴了传统的软件开发方法，从而形成了一个 Internet 软件开发生命周期。用户的直接方式——他们不只使用电子邮件传递消息——将具体的方法添加到传统的软件开发生命周期之中。以下是典型的 Internet 软件开发生命周期：

1. 以 Web 网站物理模型说明程序。
2. 编写软件。
3. 对应用程序进行单元测试。
4. 修改在单元测试中发现的问题。
5. 内部人员对应用程序进行测试。
6. 修改测试中发现的问题。
7. 将软件发布到 Internet 上。
8. 为现有服务器快速修复小错误。

在步骤 8 将软件发布到 Internet 与接收用户的第一次反馈之间有一小段的时间间隔。反馈通常使得业务以快速的方式忙于用户反馈。软件的每个改动都预示着新的生命周期的开始。

生命周期包括每个人开发 Web 使能应用程序的任务。看待生命周期的另一种方式是理解开发的过程，如下所示：

1. 写下需求说明。
2. 验证需求。
3. 完成项目。
4. 对应用程序进行单元测试。
5. 对应用程序进行系统测试。
6. 预先部署应用。
7. 开始制作产品阶段。

为 Web 使能应用程序项目定义阶段和生命周期可以表明项目将以合理的、正确构想的和适当的步骤进行运行。高级管理、用户、销售商、服务提供商、销售和市场人员、财务管理员都不参与其中！他们会因特定的利益而左右项目，直到项目成为图 3-1 描述的那样。

最佳布置的计划通常假定开发小组成员相互合作，包括内部的和外部的。但是，实际上所有这些要素都需要 Web 使能应用程序。许多软件项目是从定义完善的 Web 使能应用程序的项目阶段开始的，但是在考虑所有的项目需求时，项目可能看起来一团糟(图 3-1)。

面对这些杂乱的标记和连接，软件项目管理员通常将其分为两大阵营，用最好的方法建立、部署、维护高质量的 Web 使能应用程序。一个阵营重点放在项目小组对 Web 使能应用程序大规模的改动上。新的软件版本需要大量的工作才能赶上发布日期。另一个阵营重点放在“分治”改进的长串列表上。需要开发一系列成功的小改动，而不是作出大的改动。

在创建 Web 使能应用程序的企业中，软件项目管理员希望时常通过增加许多小的改进和改正错误来维护软件，在管理上给软件开发小组施加一定的压力以获得一个易理解的新版本。

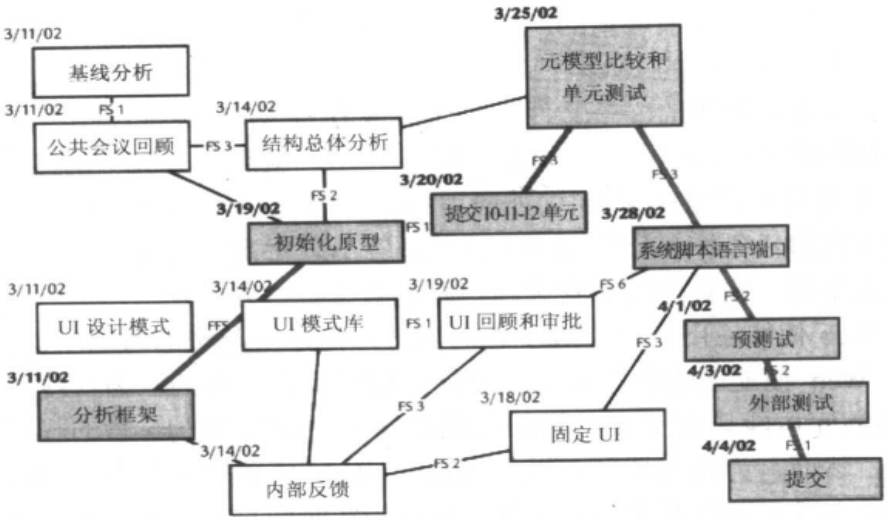

图 3-1 为开发典型的 Web 使能应用程序需要管理复杂且相关联的标记，对于软件开发小组如何完成项目有着很大的影响

3.3 Mircomax 生命周期

Micromax 是为现有软件项目部署许多小改进的一种方法。许多大公司都使用了 Micromax 方法，而且得到了很好的效果，例如 Symantec 和 Sun Microsystems。Micromax 定义了 3 种技术：一种用于对问题进行分类和排序的方法，一种为开发小组分配任务的方法，还有测试并验证改动的自动化技术。项目管理员通过可预测的调度和合理的资源分配从 Micromax 获得了益处。因为项目是自包含的，并且为开发人员进入项目提供了机会而不是处理众多的目标，所以开发人员也能够从中受益。QA 技术员通过了解测试和解决问题的最佳顺序也得到了好处。

3.3.1 问题分类

Micromax 定义了对问题进行分类和排序的一种方法。用户、开发人员、管理员、分析员可以报告所遇到的问题。其目标是开发一种能够理解并解决问题的方法。输入越多，效果越好。

问题可以是错误、改动、改进的请求、愿望以及没有文档化的特征。选择最适合自己小组的术语，小组要包括工程组之外的人员。Micromax 中的一个问题在于有利于用户或者公司的改动声明。但是，问题报告需要按照对用户的影响程度进行分类。表 3-2 描述了 Micromax 定义的问题分类。

表 3-2 Micromax 定义的问题分类

分 类	解 释
1	数据丢失
2	功能丢失
3	间断的功能丢失
4	工作区功能丢失
5	速度受损
6	适用性相互抵触
7	装饰(Cosmetic)

分类 1 的问题报告通常是最严重的。每个人都想在生活中作出标记，几乎没有人希望将他的标记删除。当一个在线银行的 Web 使能应用程序丢失了您最近储蓄记录的时候，当远程文件服务器删除了包含重要报告的一个文件的时候，或者当 Web 使能应用程序本来是要删除一条消息但结果却是删除了所有电子邮件消息的时候，这些就属于分类 1 的问题。

分类 2、3、4 是用来说明在 Web 使能应用程序中没有起作用的特征或者功能。Web 使能应用程序没有完成它的任务——分类 2——或者任务不是每次都完成——分类 3——或者功能没有执行但是可以使用一些其他的步骤达到相同的效果——分类 4。

分类 5 说明 Web 使能应用程序完成了它的任务，但是所花费的时间不能满足用户的需求。经验表明，每个 Web 使能应用程序定义的可接受时间都是不同的。为在线用户提供单点登录的 Web 使能应用程序可能只有 1~2 秒钟的可接受速率。需要 12~15 秒的相同单点登录时间可能是无法接受的。然而，为化工场每天提供报表的 Web 使能应用程序可以接受数秒或者数分钟的响应时间，因为报表浏览者并不需要每秒更新的实时数据。分类 5 在分类列表中的地位通常是先由软件开发人员编写软件的功能，然后修改软件使得可以执行得更快。

分类 6、7、8 问题是最难确定的。它们更加接近于主观臆断。对于每个错误位置的按钮，屏幕上无法理解的指令列表，本应存在但却奇怪消失的功能是需要开发人员解释的，使用所有能用到的原因解释为什么将软件建成这个样子。当对问题分类的时候，要记住用户的目标。

分类 6 确定了 Web 使能应用程序能够充分完成任务。但是，任务需要多个步骤，需要太多的上下文用户知识，使得用户无法完成更大的任务，或者它是最笨的用户界面设计。软件用户在任何时候都会遇到使用性的抵触。例如，打印机在运行的时候缺纸了，它会询问用户是“继续打印”还是“终止打印”。用户的目标是终止打印，但是他需要给打印机添加纸张以便继续打印。分类 6 问题减缓或者阻碍了用户要达到的目标。

分类 7 确定了包括出现在错误位置的图标、颜色选择、用户界面元素的问题。当用户抱怨无法满足他们的目标或者无法确定如何使用 Web 使能应用程序的时候，可以归为分类 8 问题。

Micromax 系统将软件问题归纳为 8 级问题，不可操作、无法使用、难用的界面、速度很慢的性能——这些对于用户都是很头痛和低效的。

3.3.2 问题排序

虽然问题分类有助于理解 Web 使能应用程序的性质并且可以直接辨别，但是分类自身可能产生误导。如果 Web 使能应用程序丢失了一个用户的数据而所有的用户遇到了响应速度慢的问题，这就需要一些其他的分类。给问题排序是一种好的解决方案。表 3-3 描述了 Micromax 定义的问题优先级。

表 3-3 Micromax 问题优先级

优 先 级	描 述
1	不可接受的业务风险
2	产品成功所需的紧急处理
3	需要解决的问题
4	具有时间限制的需要解决的问题
5	对业务目标是低风险的

优先级为 1 的问题表明严重的损失、业务风险和可能发生丢失——我听说过它被描述成“某人的头发着火了。”该问题需要马上解决，否则公司将面临倒闭的危险。例如，一家 Web 使能应用程序的公司每年花费市场利润的 40%用于一次交易会，当公司标识语在数百名议会者前无法正确显示的时候，公司可能会遇到装饰(分类 7)的问题但是可以将它的优先级设置为 1 从而避免这种错误。

不利的方面是优先级为 5 的问题。这些问题通常是“不合逻辑”的。因此，它们被列为跟随系统而且很少跑题的问题——这并不一定是坏事，因为以它们的性质，它们不会给公司、产品和用户带来太大的风险。

3.3.3 报告问题

在 Micromax 中，用户和项目管理员都对问题进行分类。项目管理员经常与内部小组讨论错误的优先级分配。“那个问题的确需要马上解决吗？”通常是其中讨论的问题。

Micromax 依赖用户对分类的理解以及对问题正确理解的应用。依赖用户对问题进行分类有好的一面，用户减少了小组对问题内在化所需的时间。当然，必须认真考虑用户可能应用的级别从而确保与用户级别相一致。项目管理员为问题设置分类并且将用户输入作为内部小组理解的另一个数据点。

3.3.4 评价问题的准则

通过使用现有的 Micromax 系统，项目管理员就有了一种分类并排序错误的方法。管理员使用的准则是非常重要的。成功的准则可以解决用户对 Web 使能应用程序的目标。例如，当系统开始使用的时候，一个为文档共享化提供安全、私有的 Web 使能应用程序要遵从下列准则：

- 分类 1、2 的问题不能具有优先级 1、2、3、4
- 分类 1、2、3、4、5 的问题不能具有优先级 1、2、3
- 分类 6、7、8 的问题不能具有优先级 1、2

在这种情况下，使用性和装饰问题对于发布的版本是可以接受的，但是，数据丢失问题对于发布的版本却是不可接受的。

另一个示例是，一个使用 Web 使能应用程序的电视宣传节目(TV talk show)开始组合其内容，这对于建立它的品牌要比内容的正确交付更具有商业公平性。发布的准则如下：

- 分类 7 的问题不能具有优先级 1、2、3、4
- 分类 6、7 的问题不能具有优先级 1、2
- 分类 1、2、3 的问题不能具有优先级 1、2
- 分类 5 的问题不能具有优先级 1、2、3

在这个示例中，电视宣传节目将重点放在解决装饰和速度的问题上。虽然它也希望有更多的功能，但是将重点放在建立 Web 使能应用程序上可以获得更好的设计。

Micromax 系统对于项目管理员、QA 技术员、开发管理员等都很有用。开发管理员为开发人员确定分配问题的准则。例如，给新加入小组的开发人员分配低优先级的问题降低了开发人员对 Web 使能应用程序项目作出不适当的设计的风险。该准则也定义了一个协定，开发人员根据说明书交付软件的功能。我们在本章稍后可以看到，这个协定在单元测试和灵活(也称为可扩展编程，或者 XP)开发过程中扮演了很重要的角色。

在有很多开发人员的情况下，选择以小幅度增量提高并维护 Web 使能应用程序的时候，Micromax 是一个很好的工具。使用 Micromax，调度更具有可预测性，开发小组可以紧密地协作，用户也会对他们所使用的 Web 使能应用程序出现的改进大加赞赏。

3.4 Web 使能应用程序的测试

正如我在第 2 章指出的那样，影响 Web 使能应用程序的功能、性能、可扩展性的问题通常与所编写的实际代码联系很少。本章在下面几节将说明需要查找什么，如何量化性能，并介绍设计和测试 Web 使能应用程序的方法。

3.4.1 功能和可扩展性测试

为 Web 使能应用程序投资的业务会将功能提交给用户、客户、发行人、雇员和协作伙伴。在本节中，我以一个为其雇员提供在线书店以发布公司出版物的 Web 使能应用程序为例，讨论功能和可扩展性测试方法的目标。然后说明如何测试系统的功能和可扩展性。

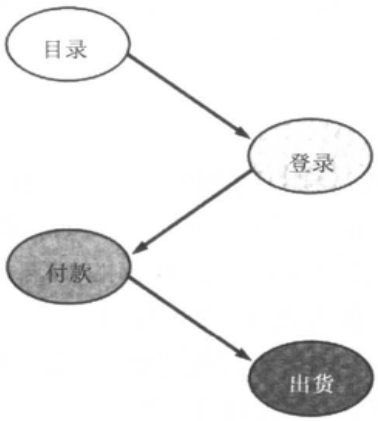

图 3-2 组合多个服务，从而建立一个系统

图 3-2 说明了系统设计。当后端系统由 4 个 Web 使能应用程序组成时，公司为用户提供了完整的、印象深刻的单一服务。

在线书店的示例使用目录服务通过标题查找图书。用户一旦选择了一本书，登录服务确认并认证该用户。接下来，付款服务负责用户为图书的付款预算。最后，出货服务取得用户交付的信息并且填写表单。

对于用户，系统成为单一的应用。在后端，图 3-3 说明了用户在 4 个完全独立的系统中移动，这些系统构成了一个单独的 Web 使能应用程序。这种联合发生在设立商标的阶段(色彩、标识语、页面布局)，在安全阶段提供对整个应用的单点登录，在数据共享阶段系统共享与订单相关的数据。其结果就是用户使用 Web 使能应用程序的整个流程。

用户从目录服务开始使用。通过 Web 浏览器，用户在目录服务中进行搜索、选择。用户选取一本书，然后单击“购买”按钮。浏览器就向登录服务发送一个 HTTP Post 命令。Post 命令包括所选取书的表单数据。登录服务显示询问用户身份号码和密码的 Web 页面。登录服务通过使用 LDAP 向目录服务发送一个请求。LDAP 是根据功能强大的 X509 安全标准指定的一

个流行的协议。目录服务根据雇员的身份号码作出响应。如果它是合法的雇员身份号码，登录服务将用户的网页重定向到付款服务中，同时向付款服务器发送 SAML 确认的调用。

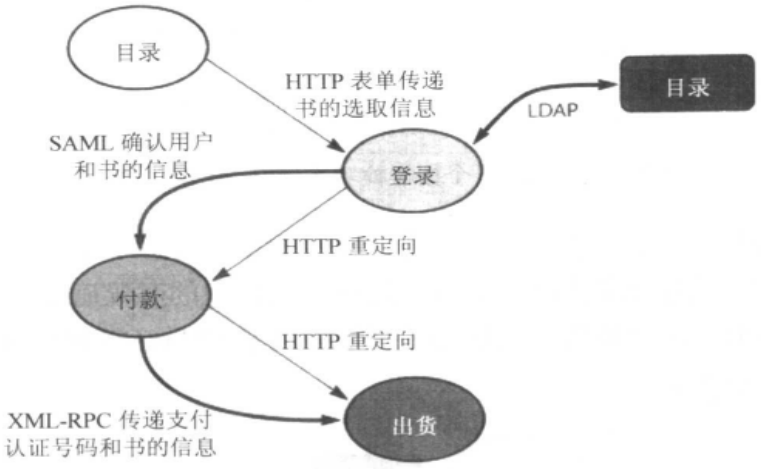

图 3-3 使用 SAML 的书店的示例。XML 远程过程调用(XML-RPC)、LDAP 以及其他协议，将独立的系统联合成用户的整个操作过程

直到现在，用户的浏览器才从目录和登录服务中得到重定向的命令。重定向命令将事务涉及的数据(也就是选取的书)加入 URL 中。遗憾的是，这个技术受到浏览器所能处理的 URL 最大长度的限制。另一种可选的方法是使用 HTTP 重定向命令和服务间的异步请求。书的标识、用户的标识、账户信息以及用户购买信息通过这些异步调用可以从一个服务转向另一个服务，用户浏览器通过使用对话标识(例如浏览器 cookie)也可以从一个服务重定向到另一个服务。

一般的服务仅仅支持受限数量的服务，因此这个示例使用了 SAML 的登录和支付服务，又使用了 XML-RPC 的出货服务。用户一旦确定他们的支付信息，支付服务就将用户浏览器重定向到出货服务中，向出货服务发送一个 XML-RPC 调用以确认所定购的书。

看到这个系统却令我感到疑惑：您会怎样测试该系统呢？像这个书店示例的互操作系统每次都需要无缝的协作。功能测试为我们提供了对支持无缝操作系统的有意义的测试数据。可扩展性测试将说明该系统可以处理每次数量都不断变化的用户群组。

3.4.2 功能性测试

功能性测试要比可扩展性和性能测试困难一些。可扩展性测试在系统并发地增加用户数量的时候，回答如何影响性能的问题。性能测试回答关于系统失效而无法满足用户目标的频率问题。功能性测试回答这样的问题："整个系统能够完成用户的目标吗？"

功能性测试确保 Web 使能应用程序的特征按所设计的那样运行。Web 使能应用程序返回的内容是有效的，Web 对应用的改动是恰当的。例如，设想一个转售产品的业务。当一个新的转售商登录的时候，公司管理员使用 Web 使能应用程序激活转售商的账号。这一步初始化了几个处理过程，包括为转售商建立一个电子邮件地址，为转售商设立全部价格的计划，为转售商建立一个销售的限额/预计。如果管理员可以单击一个按钮从而检验转售商的电子邮件、价格、限额是否建立成功，这将是多么方便的事情啊！图 3-4 给出了一个功能性测试的示例。

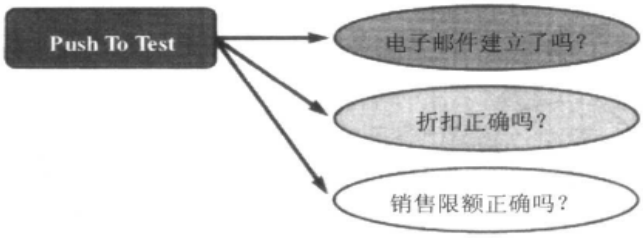

图 3-4 单击一个按钮检验系统是否已建立好

在这个书店示例中，需要一个不同类型的功能性测试。设想 4 个独立的外购公司提供书店支持服务。在这种环境下的功能性测试的目标是在出现问题的时候确定系统问题的根源。设想当已部署的服务使用来自多个销售商提供的服务时，IT 管理员必须履行什么任务。图 3-5 中的测试代理技术就是一个答案。

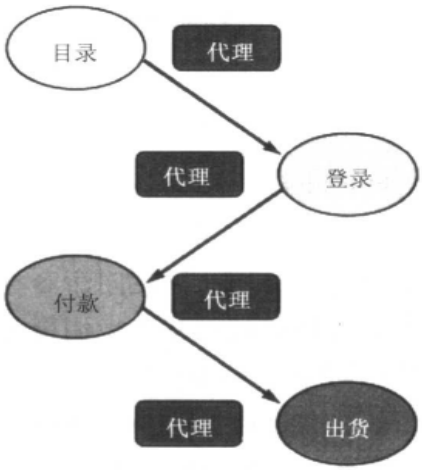

图 3-5 智能测试代理为每个服务提供功能性测试，从而为 IT 管理员指出问题所在

图 3-5 说明了如何部署智能测试代理以引导每个服务的功能性测试。测试代理监测整个书店系统中的每个服务。一个单独的控制面板协调测试代理间的行为，为保持并且分析测试代理数据提供了一个公共的位置。

这些测试代理模拟系统中每个 Web 使能应用程序的真实使用情况。代理将行为和结果记录在公共的日志服务器中。它们在组件一级的测量系统中操作。当一个组件失效的时候，系统管理员使用测试代理监测到的数据恢复失效的 Web 使能应用程序。因为测试代理数据也是满足可接受服务级别的证明，因此它具备双重作用。

3.4.3 可扩展性测试

直到现在，书店测试示例都是通过模拟单一用户在底层服务所支持的功能上的操作步骤来检验系统的功能。可扩展性测试表明当许多用户同时操作系统的时候，系统将如何执行。智能测试代理是测试系统可扩展性的理想技术，如图 3-6 所示。

在这个示例中，创建的测试代理用来执行功能测试，它们也可以重用于可扩展性测试。测试代理通过驱动系统的功能从而完成用户的行为。通过并发地运行多个测试代理的副本，我们可以得知系统是如何通过分配资源和带宽来处理负载的。

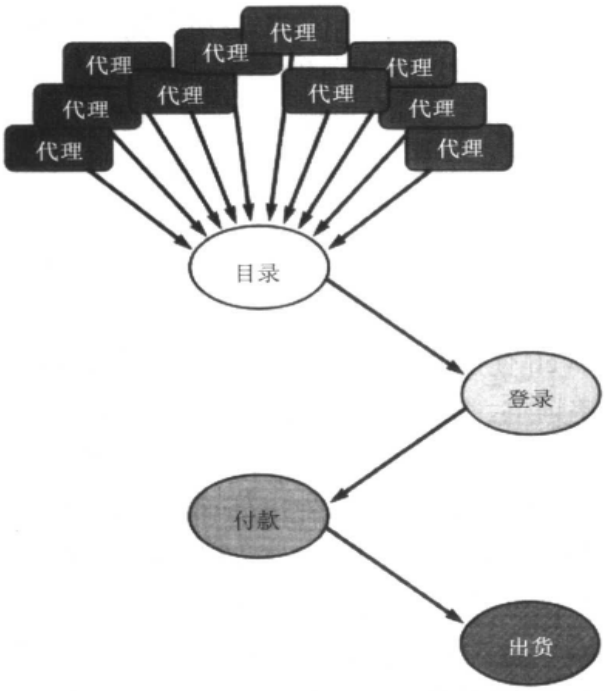

图 3-6 使用测试代理完成可扩展性测试

3.4.4 测试功能和可扩展性的模块

理解系统为用户提供服务的能力的另外一种方法是在 Web 使能应用程序提供服务的模块上执行功能和可扩展性测试。处理 Web 使能应用程序的计算机成为了互连系统网格上的节点。这些系统利用许多小组件有效地设计。我并不是说过去风格的大规模巨型机已经过时了，而是说它们也成为了网格上的一个节点。我们需要确定整个系统中每个部分的可靠性。

第 2 章介绍的饼体系结构是 Web 使能应用程序的宿主模型，其中的负载平衡器将 Web 使能应用程序的请求分配给应用服务器。因此，饼体系结构为我们提供了测试功能和可扩展性模块的一些有意义的机会。特别地，我们可以驱动在整个系统中的每个模块进行测试，而不是通过从客户端驱动系统进行测试，图 3-7 作出了说明。

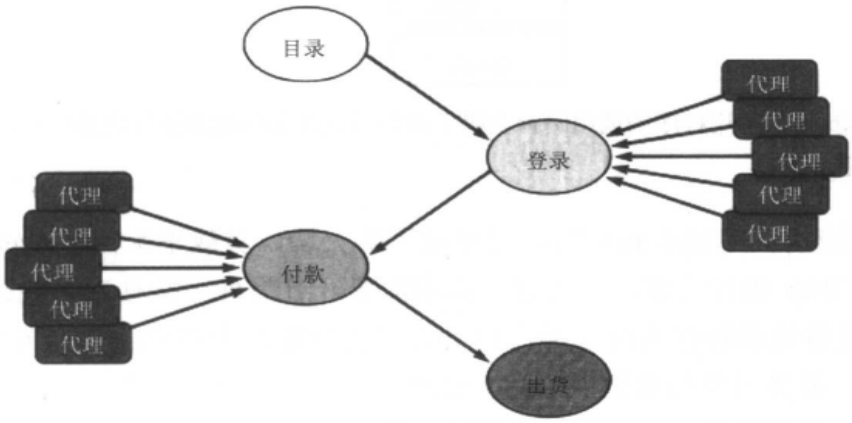

图 3-7 在饼环境下的功能和可扩展性测试使我们可以测试组成系统的模块。测试代理使用每个模块固有的协议发送请求，验证并且测量响应可以了解瓶颈和受损功能的位置

饼体系结构使用标准模块可以提供高质量的服务和低廉的成本。模块通常包括位于数据库服务器前端的一台应用服务器。负载平衡器使用 cookie 管理会话并且执行加密套接字协议层(SSL)安全通信的加密/解密。测试位于饼中心数据库的 Web 使能应用程序有如下好处：

- 即使在测试期间，负载平衡器可以使系统动态地增加更多的功能。这种灵活性能够为具有不同级别的负载和可用服务器的系统更加容易地计算 SPI，SPI 在本章已经介绍过了。另外，应用服务器可以提供可变的特征，包括处理器配置、速度、不同的内存、存储器配置的排序。
- 部署在内部网中的 Web 使能应用程序——与公共的 Internet 不同——通常需要认证和加密，通常使用数字证书，有时使用公钥基础架构(PKI)。在饼环境下测试内部网应用可以使用加密系统的可扩展性索引，它和系统的其他部分是相分离的。
- 使用负载平衡器和 IP 层路由——通常使用外部网关协议(BGP)——使得整个数据中心成为数据中心网络的一部分，负载平衡器在峰值负载时刻会降低流量但仍然保持连接。在这种环境下进行测试可以比较网络分段的性能。

以另一个角度看待 Web 使能应用程序，它甚至可以为测试和优化系统提供更多的机会。处理 Web 使能应用程序请求的调用堆栈为收集测试数据提供了几个特有的位置。调用堆栈包括图 3-8 描述的层次。

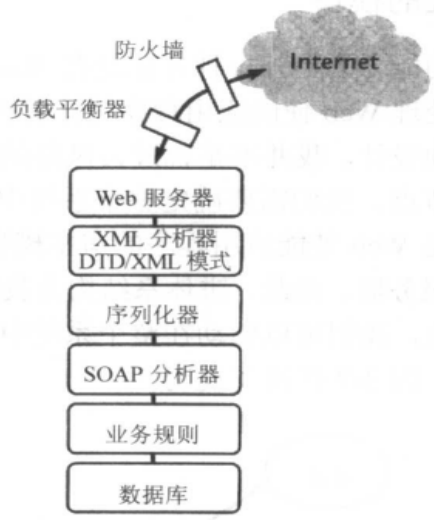

图 3-8 典型 Web 使能应用程序的调用路径说明了我们可以在哪些地方更好地测试和优化可扩展性、功能和性能

当 Web 使能应用程序请求到来时，它通过了防火墙、负载平衡器和 Web 服务器。如果它是基于 SOAP 的 Web 服务请求，那么这个请求需要经过另外一些处理，包括 SOAP 分析器、XML 分析器以及将请求转化为内部平台和语言支持对象的序列化器。业务规则指示应用程序作出适当的响应。业务对象与数据库连接，以查找响应请求所需要的存储数据。请求从数据库中返回系统堆栈之前的所有内容，最终向请求应用发送一个响应。Web 使能应用程序请求堆栈中的每个阶段都需要收集测试数据，包括如下阶段：

- **Web 服务器** 大多数 Web 服务器保存几个日志文件，其中包括页面请求、错误/异常消息、servlet/COM(组件对象模型)对象消息的日志文件。记录位置和内容在很大程度上是可以配置的。
- **XML 分析器** SOAP 分析器处理与 Web 使能应用程序主机的通信，而 XML 分析器负责处理大量的查看、验证 XML 文档的工作。
- **SOAP 分析器** 诸如 BEA WebLogic 和 IBM WebSphere 的应用服务器都包括集成的 SOAP 分析库，因此 SOAP 分析器处理数据可以在应用服务器的日志中找到。另一方面，许多 Web 使能应用程序在它们各自的应用服务器上运行。在这个示例中，绑定的 SOAP 分析器——例如，Apache Axis——在日志文件夹中保存处理的数据。
- **序列化器** 它利用请求中的 XML 代码为局部操作环境创建内部对象。序列化器将它们操作的数据记录在日志文件中。
- **业务规则** 它们通常由一组 servlet 或者分布式 COM(DCOM)对象实现，在 servlet 或者 DCOM 容器中运行，例如 Apache Tomcat。可以查看应用服务器的应用日志文件。
- **数据库** 数据库服务器在它们操作、优化以及其他工具的本地机器上维护扩展的日志文件。

收集所有测试数据的不利方面是结果数据过多。所有这些数据会令人感到无从下手！集成几个模块的系统，例如上面的书店示例，在默认情况下会产生大量的结果数据。用于 Web 使能应用程序的子系统包括商业和开源软件包，它们可以创建描述发生事件的日志文件。例如，一台应用服务器在默认情况下可以将到来的请求、应用程序具体的日志数据以及错误记录在日志文件中。当然，也在局部文件系统中保存日志数据。这也可能在 Web 使能应用程序的环境下出现问题，其中之一就是系统的一些部分无法从局域网上访问。

许多商用软件包都内置有数据收集工具。用于收集并分析简单 Web 使能应用程序(HTTP 和 HTTPS)的工具也有很多。使用 Internet 搜索引擎将会找到所选取的大量数据集合和分析工具。

迄今为止，已经介绍了用于检验功能、可扩展性和性能的智能测试代理驱动系统。然后，代理将记录中心日志文件用于以后的分析。毕竟代理所访问的是将它们的操作记录在一个 Web 使能应用程序中所需要的 Internet 协议。

在一个智能代理环境中，收集结果数据需要下面的因素。

1. 收集什么数据

数据收集要依赖测试准则。验证功能准则会像事务那样，通过成功比率执行一组功能从而进行数据的收集。测试验证可扩展性准则通过单独步骤说明哪个功能具有很好的可扩展性，从而完成数据的收集。验证性能准则是收集错误和异常状态发生的数据。

测试代理所收集的最基本数据包括时间、位置、每个事务处理任务的基本信息。例如，当验证一个 Web 使能应用程序功能的时候，测试代理会将下列数据记录到日志文件中：

代理	任务	结果	模块	持续时间
Stefanie 1	Sign-in	Ok	com.ptt.signin	00:00:00:12
Stefanie 1	Run Report	OK	com.ptt.report	00:00:08:30
Stefanie 1	Send Results	OK	com.ptt.send	00:00:00:48

对于功能测试，其结果需要正确地说明整个测试各部分的功能，同时也要表明完成每个步骤所需要的时间。一些测试协议将整个系统中的一个功能描述为一个用例，其中定义了设置参

数、使用功能的步骤以及预期的结果。当验证可扩展性的时候，Stefanie 代理将下面的结果数据记录到日志文件中：

```
代理        任务              结果      时间          持续时间
Chris 1   Sign,report,send Ok       14:20:05:08 00:00:09:10
Chris 2   Sign,report,send Ok       14:25:06:02 00:00:06:12
Chris 3   Sign,report,send Ok       14:28:13:01 00:00:08:53
Chris 4   Sign,report,send Ok       14:32:46:03 00:00:05:36
```

可扩展性测试有助于了解系统为用户处理的速度。结果数据表明了每个测试代码的开始时间和完成整个用例的所有步骤所需的时间。

2. 在哪里保存数据

默认情况下，Web 使能应用程序的软件包将结果数据记录到局部文件系统中。在许多情况下，这会成为无效数据。试想在网状服务器的网络中跟踪一个 Web 使能应用程序的远程日志文件！可以查询有效的数据，但是这需要很多标记。另外，一旦定位结果数据，数据的分析就能够给出耗费的时间。

以我个人的经验，结果数据最佳的保存位置是在一个集中的关系数据库中。数据库——商业的和开源的——有很多是可以使用的，考虑廉价的选择以及内置的分析工具。数据库的选择包括带有结构化查询语言(SQL)功能的关系数据库，它可以为管理员提供在桌面计算机上运行的简单文件数据库。

3. 理解明显的失效

作为一名测试人员，持有怀疑态度是很重要的。我不是在建议 X-Files 级别的怀疑，但是应该注意有缺陷测试的测试结果。在这种情况下，测试数据可能是没有意义的，甚至有误导作用。在一个 Web 使能应用程序的环境下，下列问题可能会引起测试失效。

受限的网络带宽 许多测试假定网络带宽是无限的。但是实际上，许多网络在代理的适当程度上会出现饱和。设想如果在代理和系统之间的连接超过 T1 网络连接，网络将只能处理 16 个并发请求，每个请求传输 8Kb 的数据。表 3-4 表明了网络实际处理的流量。

表 3-4 用于处理生成测试代理数据的网络容量；性能的变化很剧烈

结果数据的大小	T1 请 求	以太网请求	T3 请 求
1 KB	132	854	3845
2 KB	66	427	1922
4 KB	33	213	961
8 KB	16	106	480

注：这些数字是通过将结果数据大小乘以每秒最大的处理能力计算得出的。T1 线路每秒传输 1Mb。100Mb 的以太网线路每秒可以传输 7Mb。T3 线路每秒可以传输 30Mb。

没有足够的数据库连接 饼体系结构环境下的系统用多个 Web 使能应用程序服务器为前端提供一个功能强大的数据库服务器。数据库连接池和对高级事务的支持在任意时刻减少了激活数据库连接的数量。数据库连接池是用 Java 数据库连接(JDBC)2.0 规范定义的，而且得到广

泛的支持，其中包括了 Microsoft 公司的 DCOM。然而，一些数据库的默认设置没有充足的数据库连接，因此无法避免经过长时间的使用而断开的后果。

无效的输入和响应 Web 使能应用程序具有多个方法，它们位于接收输入和提供响应的软件对象中。使软件应用失效最简单的方法是提供无效输入或者提供引起无效响应的输入数据。Web 使能应用程序允许出现相同的输入和响应的问题。一个优秀的测试人员会查看表示测试失败的无效数据。测试人员也会保证如期地完成错误处理。

负载平衡器成为失效的一个方面 在系统中使用负载平衡器可以集成 Web 使能应用程序，但它也会导致系统失效。当负载平衡器失效的时候，整个系统也就失效了。流行的负载平衡解决方案提供一个同时运行的负载平衡器，从而可以避免失效。

到目前为止，已经介绍了技术因素——检验功能和可扩展性——用于对 Web 使能应用程序进行测试。接下来，将介绍管理风格是如何影响测试的。然后说明期望的测试结果是如何对测试 Web 使能应用程序的方式产生影响的。

3.4.5 管理风格

在使用一个新的 Web 使能应用程序时，我的感受与电视主管看到新的电视节目的感受应该有些相似。使用一个新程序是令人激动的，但是如果它无法满足许多用户的期望时却是一件十分令人痛苦的事情。

许多业务管理员普遍曾经感到过很难满意他们的 Web 使能应用程序。TCP/IP 连接在 Internet、内部网和外部网上普遍存在，它通过浏览器或者 Web 使能应用程序软件满足每个人的需求。这给管理人员、测试人员和开发人员等带来了很大的压力。

我发现管理风格对于 Web 使能应用程序的设计和测试如何将价值体现在业务上有着很大的影响。理解管理风格和个人风格对于建立高效的设计和测试是非常重要的。表 3-5 描述了在几种管理类型的设计和测试中体现的管理风格、特征以及影响。

表 3-5 管理风格、设计和测试策略

风　格	特　点	影　响
分层的	策略为上，战术为下。基本信仰：“我们不是去问为什么，而是去做，直到死亡。”	层次中最高级的管理员已经选取了服务器、网络设备、销售商和位置。然后，设计仅仅是将销售商提供的组件粘合在一起。基于测试解决方案的智能测试代理同定义了数据搜索的参数和用于交付的可接受时间框架的管理层次一样，都能很好地运转。开发人员、测试人员和 IT 管理员应该重用之前创建的或者为过去项目购买的测试代理自动工具，从而得到更高的效率
系统化的	将一个问题从整体中分离出来，为其开发一个解决方案，然后将问题交给小组实现该解决方案	系统管理员可以使用设计工具和测试自动化工具，当它们拥有独立工具命令的时候，系统管理是非常有效的。测试工具使得系统管理员可以编写提交所需数据的测试代理。在系统管理员分配项目之前，针对测试自动化工具进行训练是非常重要的。为接收和存档测试代理提供一种方便的机制对于开发公司是非常关键的

(续表)

风　格	特　点	影　响
企业化的	想尽可能多的拥有业务机会。节省企业资金	企业通过将现有资源集成起来解决未解决的问题来发现机会。设计通常是对现存系统的修改。测试能够证明一个新的业务方法或者技术是否达到其潜能。测试应该将重点集中在系统如何运转的证明点上面
无经验的	经常失效、不重视、或者忽略公司所使用技术的业务效率	设计主要由商业解决方案的性能/价格比决定。测试可以为业务带来益处，它会节省资金或者增加税收。可以说商业利润与技术术语和版本是无关的

从表 3-5 中列出的风格可以看出管理员的风格，它会鼓励您设计并测试数据，然后识别出自身的风格。风格差异的益处是可以为您带来业务上的职位升迁。忽略管理风格可能是危险的。例如，将设计改进和设计策略的企业列表交给分层管理员其结果可能会是令人失望的。

列举一个真实的示例：Symantec 公司的一名测试管理员明显表现出了企业化风格，并和一名分层风格的产品管理员一同工作。该测试管理员识别出了他自己的企业化风格，并且改变了他的方式，从而可以和分层风格的管理员协同工作。该测试管理员并不将重点放在即将开始的软件项目上，而是注意如何重用现有的测试自动化工具和代理以节省公司资金并能解决以后的销售问题。

某些风格有与其他风格相冲突的趋势。试想企业化风格的执行官与系统化风格的测试管理员一同工作。当执行官需要测试数据时，系统化风格的测试管理员可能不在——他与其他小组一起解决另外的问题。理解管理风格和管理风格之间如何融合可以提供一个更好的工作环境和更高效的测试。

3.5　服务级别约定

外购 Web 使能应用程序的需求在当今的商业中每天都会发生。商家从 ISP 那里购买 Internet 连接和带宽，从配置供应商那里购买服务器主机设备，从应用程序服务供应商(SAP)那里购买应用程序。高级 ASP 供应 Web 使能应用程序。每个商家都依靠外购公司而获得可接受的服务级别。一个公司安全方案的公共部分是要求外购公司提交一个服务级别约定(SLA)，它可以在预先定义的级别上确保性能。SLA 要求服务供应商以适时的方式对问题的响应和失效的处罚作出承诺。表 3-6 给出了 SLA 的一些通用方面。

表 3-6　服务级别约定术语

目　标	描　述	如 何 度 量
正常运行时间(uptime)	Web 使能应用程序能够接收和响应请求的时间	每周中正常运行的小时数除以一周的总时间(168 个小时)。结果接近 100%。例如，如果系统在一周有两个小时失效，服务就达到了 98.80952%的正常运行((168 - 2)/168)。结果越接近 100%，表明运行得越好

(续表)

目　标	描　述	如何度量
响应时间	开始解决问题之前的时间	从开始报告问题到技术员开始解决之间所用的平均时间，以分钟为单位。技术员是指经过训练且可以解决问题的人员
恢复	解决问题的时间	以分钟为单位，从开始报告问题到问题解决之间的最长时间
延迟	在供应商的网络中，网络流到达其目的地的时间	从 Internet/内部网到服务设备之间的慢速网络连接的测量；报文到达目标服务器的平均时间
维护	维护周期的频率	服务供应商维护周期的总次数，以月为单位
事务	全部请求和响应次数的索引	系统处理的请求/响应对的全部数量。数量越多表明越好
报表	关于监测、条件和结果的统计数据	以 30 天为周期，产生报表的全部数量

SLA 实际上给商家提供了如下保证：

- 服务供应商必须同意提供良好服务的准则。在现实环境下，由于服务术语的不统一，使得报告的问题无法解决。SLA 要描述所交付良好服务的所有必要的方面。
- SLA 成为服务供应商降低日常风险中的一部分。无法提供良好服务可能很快会影响到服务供应商的财政问题。当服务供应商无法满足 SLA 的时候，提供商将退还为商家提供服务的费用。依靠 SLA，甚至更多违反 SLA 的服务通常会使供应商为顾客的损失支付现金。

Internet 发展到这一步，应该适当地利用 SLA。但是，Web 使能应用程序为 SLA 增加了额外的需求。目前的企业交付系统是将几个 Web 使能应用程序集成为一个整体系统。例如，为雇员提供一个共同的门户，它将一个 Web 使能应用程序中的公司运输报告和另一个 Web 使能应用程序中的销售商目录集成起来。如果不同的供应商要完成这个应用，如何将 SLA 应用到整个系统中呢？其需求就是 Web 使能应用程序服务级别协定(WSLA)。

WSLA 的目标是了解在整个系统中哪个 Web 使能应用程序运行的效果不好。WSLA 要求每个服务供应商为测试 Web 使能应用程序提出一种方法和查询记录数据的一种标准化方法。测试必须说明向 Web 使能应用程序发出请求的内部协议。例如，Web 使能应用程序中的公司运输报告可以使用 SOAP 协议响应这个报告。测试 Web 使能应用程序需要服务供应商向现有的 Web 使能应用程序发出真实数据的 SOAP 请求。检测响应数据的合法性并将结果记录下来。

WSLA 为远程接收记录的数据和在给定时刻下记录数据的数量定义了一种标准方法。根据活动级别和记录数据的数量，WSLA 应该要求服务供应商至少保存记录数据 24 小时。商家和服务供应商需要协同记录数据的格式和查询机制。查询记录数据的流行方法是使用 FTP 服务、电子邮件附件以及基于 SOAP 的 Web 使能应用程序。

WSLA 的含义是当用户使用系统遇到问题的时候，业务通过集中、易访问的方法可以确定每个 Web 使能应用程序都发生了什么事情。

当然，只要使用服务供应商的产品，就要面对 Web 使能应用程序技术的现实。服务供应商用物理设备、雇员、设备、带宽、次序表应对每天的商机。增加一个智能测试代理机制——需要交付 WSLA——服务提供商不必参与这项任务。

随着当今 Internet 技术的进步，我们开始发现 Internet 计算有些像互连计算机之间的网格。智能测试代理技术在网格计算中非常适合服务供应商的发展。

3.6 网格计算和智能测试代理

在微型计算机和计算机产业发展的 25 年中，通常需要单击 Reset 按钮，计算机才可以解决系统问题。虽然 Reset 按钮对于功能性的解决并不是很完美——却一直很有效——但是当单击 Reset 按钮的时候，系统操作员希望知道出现了什么错误。如果无法雇佣足够多的系统分析员，我们如何保证一个系统是在峰值情况下运转的？让系统自己知道其如何运转而且不依靠功能瓶颈还需要做很多工作。

如果我们可以建立并且管理能够提供简单安装、良好使用、低成本维护的计算机基础架构，这样难道不好吗？这是由网格计算技术和自维护(self-healing)技术的倡导者提出的新的基础架构。Internet 用户已经从网格计算的初期中获益了。

网格计算采用了目前的 Internet 环境并将其模块化。网格希望每个数据中心都有相同的基本组件：存储器、处理器、交换机和内存。网格使用标准的开放协议使数据中心进行通信。通过标准化的通信，数据中心成为了商品，它们可以容易地替换和交换。例如，如果 New York 的数据中心不能正常工作，那么可以通过 San Jose 的数据中心进行处理。取消网格上的一个数据中心节点也允许数据中心激活系统自我诊断，在很多情况下，可以进行自维护。自维护就意味着很少单击服务器上的 Reset 按钮。高级的自维护系统明白它是如何使用系统的，而且会采取适当的步骤来提高系统的性能和有效运行时间。支持网格计算的协议还不成熟，仍然需要进行大量的工作来解决。

网格计算的大量工作是围绕建立基础架构的各个部分——路由器、设备和服务器——相关的技术包括：对等(peer-to-peer)(P2P)技术，如 sun 的 JXTA(http://www.jxta.org)，和自适应网络技术，如 sun 的 Jini(http://ww.jini.org/.)。JXTA 使得对等系统可以创建一个虚拟网络，即使当一些对等系统和资源在防火墙之后或者有不同的网络传输的时候，任何对等体都可以同其他对等体和资源之间进行交互。Jini 网络技术是一个开放体系结构，它可以使开发人员创建网络中心服务——无论是由硬件还是软件实现的——都对变化有很好的适应性。现在谈谈 Web 使能应用程序，它采用现有的服务器功能并且向外提供服务，以及在松耦合、细粒度的系统中进行互操作。Web 使能应用程序对于网格计算来说是一个重大进步。

3.6.1 简单安装、使用和维护的方法

计算机产业无法为系统提供简单的安装、使用和维护，这也通常是挑剔 Microsoft 的重要话题之一。其观点是 Microsoft 这样单独一家公司无法满足全世界所有计算机用户的需求，这是因为用户需求是无休止地变化的。Microsoft 垄断了桌面操作系统，并且采用封闭的开发技术，以阻止对 DOS 和 Windows 的有效改动，软件开发人员无法访问其代码，且无法发布改动。

在联邦法庭的详细审查下，Microsoft 才开始开放 Windows 源代码，Microsoft 在软件开发群体中扮演着最差的角色。将精力用于提高开源软件的最优秀的工程师是不能在 Microsoft 工作。

DOS 和 Windows 中最过分的问题是从像 Network Associates 这样的第三方软件发行商工具软件产品提升到整个产业。日常的用户可能在 DOS 或者 Windows 中发现问题，能够查找零售店对工具软件进行修复。通过零售发布系统进行交付软件有其自身的问题，其原因是销售产品所需的时间总量。最终用户等待工具软件发行商对问题的解决方案，而这些问题是绝对不会出现在操作系统上的。零售商发布软件会引起产品不定性，以及需要更多的时间才能投放到市场。

Microsoft 和其他平台供应商正在逐渐依靠工具软件发行商解决操作系统中的问题。他们将操作系统遗漏的部分描述成为“第三方的机遇”。计算机产业将其归为发布新操作系统技术的过程。相应地，用户在操作系统投放之后的一段时间里继续使用有错误的、功能不完全的操作系统。

然后，又有惊人的事情发生：主体平台供应商采用 Web 使能应用程序创建下一代平台。大多数操作系统和软件应用迅速庞大，从而需要对它们像小领域自身那样进行管理。在 Microsoft 的 Windows 中包括数千个库、工具程序、存储格式和接口。控制的权力是由平台供应商制定的，其中一个小组可能或者不能与另一个小组协作。例如，Microsoft 的 Internet Explorer 小组需要多媒体小组的支持。因此，即使在单独操作系统内部也可能有“第三方的机遇”存在，其中所需的功能对于操作系统的完成还是有遗漏的。平台供应商——Microsoft、Oracle、IBM 以及其他公司——内部采用 Web 使能应用程序的协议，包括可扩展标记语言(XML)、远程过程调用(RPC)、SOAP、WSDL，以及全局描述、发现、集成(UDDI)，以在内部的小组中增加互操作性。

采用 Web 使能应用程序的协议还有另一个好处，它使得平台供应商能够更容易地开放他们的系统。暴露 API 需要大量的工作，以交付一个完整的软件开发工具包(SDK)。Web 使能应用程序通过使用 XML 来进行自我描述。使用 Web 使能应用程序的协议一旦创建出一个内部 API，平台供应商仅需要选择将哪些 API 提供给想要加进新功能的开发人员。

随着开发人员在创建自己的软件产品中广泛地采用 Web 使能应用程序协议，现在可以制定出网格计算和自维护技术所需要的协议。

3.6.2 自维护系统

试想一下自维护系统是个什么样子。对于一家为其职员运行人力资源门户的公司，用户可能发现一个集成的系统，该系统具有为职员制定退休金计划、控制税收和安排度假时间的一些管理服务。图 3-9 给出了通过职员浏览器看到的这个门户的基本形式。

像大多数企业应用软件一样，门户显示了从许多服务中收集到的数据。PeopleSoft 的 Web 使能应用程序提供退休金计划功能，美国薪水银行的 Web 使能应用程序提供控制税收的功能，内部数据库提供度假时间安排的功能。一个应用服务器协调 Web 使能应用程序的请求，它需要为职员显示出单一且统一的 Web 界面。

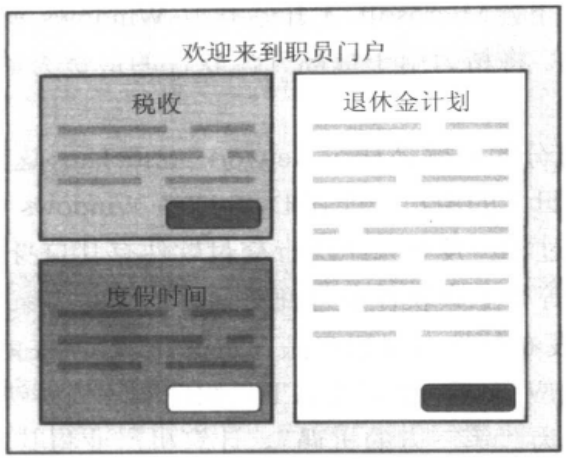

图 3-9　一个人力资源门户中有关职员的场景

在这种情景下，试想当度假时间数据库失效的时候，会发生什么情况呢？驱动门户的现代软件将检测到错误，并且尽可能多地显示出有效数据。图 3-10 表明了当一个 Web 使能应用程序失效的时候，职员会遇到的情况。

图 3-10　错误状况下的人力资源门户

职员有一种选择：接受无法运转的功能或者要求某人为此负责并且解决这个问题。我们都有在服务台解决问题的工作经验。这些经验至少不会使我们感到头痛。

自维护系统可以理解门户页面的内容。系统可以理解数据何时发生遗漏、不准确或者错误。自维护网格计算环境的关键是让系统知道正确操作与问题操作之间的差别。

软件开发人员、IT 管理员、QA 技术员都承认自维护系统可以为用户提供更好的操作，提供更高的可用性，以及很少的维护。然而，一个系统中通常还有对自维护技术负责任的一些争论。开发人员认为自维护技术应该是单元测试的扩展，IT 管理员则认为它是服务级别监测的一部分，而 QA 技术员却认为它是正在进行的软件回归测试中的一部分。

虽然自维护系统可以在软件、测试或者操作级别上实现，但经验表明，诸如驱动人力资源(HR)门户的应用，其软件应该对管理应用作出控制，但是在应用程序执行不正确的时候，软件应该建成一个外部系统——一个使用智能测试代理的系统，它可以根据性能和可扩展性准则连续检验系统。由于智能测试代理对其测试的环境可以作出可编程性的响应，因此，也可以通过编写测试代理解决所发现的问题。

3.7 理解性能和可扩展性准则

对于建立大小合适的数据中心和建立更好的 Web 使能应用程序，以便能更好地满足用户来说，测试和监测一个系统是一种有效的技术。为了测试并监测系统是否有意义，需要理解具有良好性能和可扩展性的准则。可扩展性和性能准则(SPC)定义了获得良好或者不良操作所需的测试。SPI 是根据 SPC 准则测试系统的一种有效技术。SPC 定义了我们怎样知道系统运行正常，SPI 根据这个准则对系统进行验证。

3.7.1 定义 SPC

准则可以解释我们所深刻了解的几乎所有内容。有经验的 Internet 用户可以在不到 1 秒钟内判断出一个 Web 使能应用程序在何时运行情况很差。目前计算机的使用是如此的普遍，以至于每个计算机用户都具有有效功能、可扩展性和性能的最小准则。我发现 Web 使能应用程序的性能基本上可以归纳为以下 3 点：

1. 基本功能有效吗？
2. Web 使能应用程序执行得有多快？
3. 请求失效的频率是多少？

通过上面的说明，可以看出这 3 个问题都是主观的。试想一个 Web 使能应用程序其基本功能在电子商务应用中正确运行的时间占 95%。这个准则等价于在电影列表的 Web 使能应用程序中的基本功能正确运转时间占 95%的情况吗？遗漏一个电子商务功能，业务将会有所损失。遗漏一个电影列表可能没有什么大的损失。

为优秀的 Web 使能应用程序操作定义准则需要许多主观参数；但是，可以规范化所有准则，以评判任何 Web 使能应用程序的功能、可扩展性和性能。SPC 通过下面几种方法对准则进行规范化。

1. 奖励成功

虽然使用 SPC 测试的系统是多个 Web 使能应用程序的复杂且多方面的集成，但到了最后，测试仍然是系统帮助用户达到其接近目标的一种测量方法。SPC 对用户的成功使用加以奖励。用户目标满足得越多，系统的操作性就越好。

2. 奖励节省时间和防止缓慢

回顾一下古老的谚语“时间就是金钱”，直到现在，对于用户快速操作 Web 使能应用程序来说仍然是有效的。所有的用户目标都有这样的含义：用户可以在合理的时间内实现其目标。

Alan Cooper 被誉为 VB 之父，他也是 Microsoft 的一员，他将术语 taction 描述为不良创建的软件接口引起用户减缓达到其目标的特征。Web 使能应用程序也有 taction。当一个在线银行应用程序响应检验账户余额查询的时间超过 10 秒钟的时候，用户就被减缓达到其目标。在相同余额的查询要求用户访问几个 Web 页面从而获得账户余额的时候，这也是 tation 的一个示例。

根据对 California 和 New York 的 Web 使能应用程序用户进行的不科学统计，大多数 Web 使能应用程序用户在浏览器上单击 Stop 按钮之前，他们希望等待时间少于 5 秒钟。用户根据

完成功能的时间少于 5 秒钟对 Web 使能应用程序进行评估，而且不理会耗时的 Web 使能应用程序。多于 10 秒钟的等待足以使得用户取消操作。SPC 奖励快速操作，而防止缓慢操作。

3. 防止失效

气愤、失望、疲惫的用户要远远多于得到满足的用户。当建立在 Web 使能应用程序上的系统失效的时候，用户通常会有这些感受。失效可以发生在某个协议级别——系统在该级别下无法处理请求——或者发生在一个内容级别下——在该级别下返回的数据是错误的。例如，在一个职员门户中的内容失效可能是显示了另一个职员的退休金计划信息。SPC 在失效操作的几何比例中统计否定点，并与成功的操作相比较。

SPC 也根据失效操作所占时间的比例来防止失效。一个用时少于 1 秒钟的内容失效没有用时为 10 秒钟的内容失效那样严重。虽然两个操作最终都失效了，但更快的响应减少了用户的等待时间。没有什么比一个应用消耗很长时间但最终失效更令人厌恶了。

4. 奖励增加的价格/性能比

虽然购买设备、带宽、主机服务的成本在过去十年中大幅度降低，但这些成本仍然是商家主要预算的一部分。在一组给定的设备下，能够处理更多用户的系统可以通过系统代价与处理用户的数量的比率进行描述。计算机制造业引入了术语“价格/性能”比用于描述这个比率。SPC 通过查看某一时刻所处理的并发用户数量来奖励更高的价格/性能比。

5. 理解内部用语(jargon)

计算机产业使用了大量的内部用语和习惯用语。在 Web 使能应用程序测试世界中，术语“并发”、“同时”、“同步”、“一致”和“伴随”都有了新的生命，并具有特定的意义。理解并发和其他术语之间的差异有助于构建 SPC。

- **并发(concurrent)**——暗指特征或者时间长度内的并行性：系统能够在 1 个小时内处理 1000 个并发用户。
- **同时(simultaneous)**——狭义地说明了在同一时刻许多事件的发生：用户同时测试 Amazon.com 和 barnesandnoble.com。
- **同步(synchronous)**——指在一个较短的时间间隔内，事件之间的一致性(correspondence)：测试代理执行一系列的同步请求。
- **一致(coincident)**——对发生在同一时刻的事件的应用，事件之间不需要具有关联：库存控制系统的减速与信用卡验证减速几乎是一致的。
- **伴随(concomitant)**——表明事件在时间上的一致(coincident)，时间之间的关联明显，以至于一个事件好象伴随另一个事件：测试代理在数据库中发现的问题与该问题伴随的库存控制问题在时间上非常接近。

虽然上述每个术语对于定义测试准则都是有效的，但 SPC 更加强调并发测试。并发测试在某个时间区间内测量许多请求的吞吐量，这与同时和同步测试有些不同。在 Web 使能应用程序的环境下，请求可能在任何时刻发生并且需要一段时间才能完成。

3.7.2 运转中的 SPC

执行 SPC 将产生一个 SPI，SPI 是一个能够在多个方面使用的度量标准，包括如下一些方面：

- 鼓励软件开发人员为用户更好地优化代码
- 帮助 IT 管理员设计并部署适当大小的数据中心
- 为管理标准提供对用户满意的判断
- 帮助 QA 技术员了解新软件是具有前面已经解决的问题

虽然 SPC 是一个很好的准则，它可以很好地进行平衡从而测试 Web 使能应用程序，但是根据所测试的 Web 使能应用程序，SPC 需要特定的输入。让我们看看 SPC 的一个示例，这个示例描述了美国一家大保险公司的报价策略系统。这家保险公司在美国的西部和东部都有办公地点。他们通过规章要求保存记录，这些记录描述了他们赢得新的商业帐目所制定的报告。

系统为报价策略系统提供了一个 Web 浏览器前端。保险代理商使用他们办公桌上的 Web 浏览器为商业客户提供建议。系统也提供商业的市场数据，代理商可以将这些数据用于建议之中。

出于冗余和访问速度的原因，系统在两个不同地点的数据中心上运行。系统将报价单保存在备份的数据库系统中，备份数据库可以被每个数据中心所访问。系统使用一家专业调查公司所提供的 Web 使能应用程序，从而可以对商业的市场数据进行调查。

在日常的操作中，代理商登录到系统上，并且开始一个新的报价单会话或者查询一个已有的报价。在开始新的报价单会话的情况下，系统使用预定的顾客业务名称和 SICS 代码去检索相关的商业市场数据。保险代理商会选择在报价单文件中包含的的市场数据。该代理商可能有选择性地将市场数据同报价单附在一起，从而将该报告作为支持代理商的报价单的证据。图 3-11 给出了系统的体系结构。

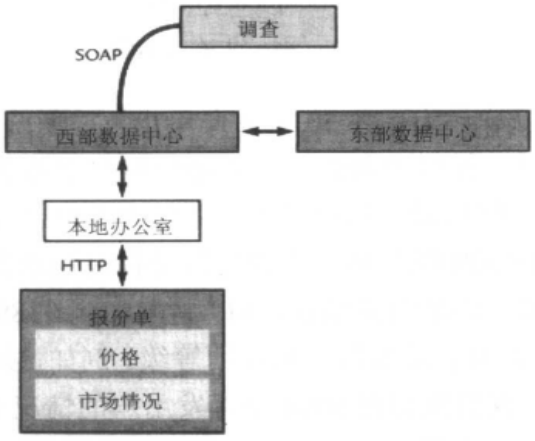

图 3-11 保险报价策略系统

一家通过基于 SOAP 的 Web 使能应用程序进入数据中心的调查公司提供了市场数据。代理商输入一个业务 SIC 代码和使用 Web 浏览器的预定顾客名称。本地办公系统将一个基于 SOAP 的 Web 使能应用程序请求发送给数据中心，它依次向调查系统发送请求。其响应包括可获得的调查和可获得的价格。

该系统的体系结构是现代 Web 系统的典型，其中系统是由不同 Web 使能应用程序集成的。虽然这些系统很常见，但它们对于 IT 管理员来说仍然具有挑战性，IT 管理员必须回答正常运行时间、数据中心大小、问题分辨率(problem resolution)等问题。SPC 在首次展示之前表明了这个复杂系统的性质，它是成品阶段系统性能的一种度量标准。

首先，我们看看系统的 SPI 结果，然后，研究如何发现这个 SPI。图 3-12 给出了这个保险报价策略系统的 SPI。

Part 1: Measurements—15 minutes at 50 concurrent users

Response time	Responses	Weighting	Weighted Responses	Errors	Weighting	Weighted Errors
<1 second	12520	+4	50080	0	0	0
<2 seconds	4050	+3	12150	0	1	0
<3 seconds	3100	+2	6200	16	1	16
<4 seconds	2600	+1	2600	25	2	50
<5 seconds	0	-1	0	43	2	86
<6 seconds	0	-2	0	0	3	0
<7 seconds	0	-3	0	38	3	114
<9 seconds	0	-4	0	45	4	180
<9 seconds	0	-5	0	3	4	12
9 or more	0	-6	0	0	5	0
Total	22270		71030	170		458

Part 2: Analysis

Total weighted responses	71030	
out of possible weighted responses	89080	
got this close to the goal	79.73731%	
However, total weighted errors	458	
out of the unweighted response	22270	
reduces the overall rating	2.05658%	
Giving an overall SPI rating of	77.68074%	
Concurrent users tested	50	
The SPI is	50.77681	

图 3-12　保险示例应用的 SPI

报价策略系统的 SPI 计算包括两部分：第一部分描述了 50 个并发用户使用系统 15 分钟时的性能和所记录的错误测量。第二部分描述了确定 SPI 的结果分析。在这两部分中，以数字表示 SPI。图 3-13 说明了如何解释这个 SPI。

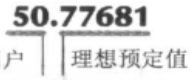

图 3-13　进一步查看 SPI

报价策略系统的 SPI 可以读作为“在 50 个并发用户下，系统执行到.77681 的预定值。”一个接近理想的比率是 50.99999，在这个级别下，为每个用户服务的时间少于 1 秒钟而且没有错误。每个超过 1 秒钟的响应时间和错误都将减少理想的.99999 预定值。

如何可以改进这个系统的 SPI 呢？最大的改进将是提高请求的响应速度。消除超过 3 秒钟的响应会将 SPI 增加到.67802。如果消除错误，SPI 将增加到.93890。

标记为 50.77681 的 SPI 表明了即将测试系统的等级。例如，如果系统安装在可以处理流行公共 Internet 站点的设备上，我们也想在 5000 个并发用户的情况下测试系统。在这种情况下，SPI 将会是 5000.77681。在另一方面，一个小部门服务器可能仅仅只能支持 7 个并发用户，在这种情况下，测试系统可能的结果会是：SPI 为 7.77681。通常，理想预定值在更大、更快的设备上会有所增加。

SPI 是确定一个 Web 使能应用程序系统在具体负载级别下处理并发用户程度的一种有效方法。SPI 的设计使得 IT 管理员、软件开发人员、QA 技术员通过下面问题就可以作出保证满足用户目标的假定，这些问题是：

- 我们有满足期望的用户数目的适当服务器的数目吗？
- 当用户得到错误响应的时候，响应时间的速度可以接受吗？
- 我们应该为服务器和带宽进行投资还是投入更多的软件开发人员？

这些问题在现实的 Web 使能应用程序环境中是很难回答的。它们主要依赖于数据中心的大小、软件性能和用户的期望来解决。SPI 是确定满足用户目标和管理 Web 使能应用程序系统的开发、测试和维护的一种有效工具。

到目前为止，已经介绍了通过 SPC 检验的报价策略系统，并学习了 SPI，下面将继续讨论如何为每个基于 Web 使能应用程序的系统规划 SPC 和 SPI。

3.8 模拟用户目标

图 3-12 中 SPI 的第一部分测量了响应时间。例如，在图 3-12 中，系统在 2~4 秒钟内处理了 22 270 个请求。但是系统实际上响应的内容是什么？响应是一个事务、会话、单独请求，还是其他什么东西？如何将一个响应转化为达到目标的用户？

在智能代理使用系统的时候，SPI 对系统进行测量。代理通过模拟用户的目标执行任务。对于系统来说，代理看上去好像是一个真实的用户通过键盘和鼠标使用一个浏览器或者软件应用程序。为了与现实产品环境更加接近，代理可以模拟几个用户。模拟的过程定义了用户原型、模拟技术和代理技术。

用户原型是系统的典型用户。在报价策略系统中，这些用户原型可能是：

- **Brigitte**——部门经理；她每天通常喝 4~5 杯咖啡，这引起了她的同事对她精神状况的评论。Brigitte 需要从在部门工作的其他代理中查询报价策略。她查看这些报告单的完整性。她很少自己制定新的方案。
- **Stefanie**——索赔(claim)管理员。Stefanie 和 Brigitte 鼓励了其他人饮用咖啡。Stefanie 需要向公司总部报告合计的报价策略总结。由于 Brigitte 对方案的内容很感兴趣，Stefanie 监测了交易流。比起 Ed 或者 Brigitte，Stefanie 缺乏组织性。她通常等到事情成堆的时候才开始工作。
- **Ed**——保险代理。Ed 已经制定方案 30 年了，并且 5 年之后就会退休。Ed 使用计算机的速度很慢。他几乎没有输入过错误的数据，但是需要 10 分钟或者更久才能完成一个新的报告单。

通过用户原型可以了解一个典型系统用户的背景、工作习惯和目标。原型方法对于通常的失误有着矫正的作用，开发人员、QA 技术员、IT 管理员通过修改方法可以解决大量问题。试图解决大量用户遇到的问题有一种散焦效应，在该效应下小问题可以得到改进而重大的用户问题却没有得到解决。

通过定义原型，实现智能测试代理变得更加直接。例如，一个基于 Brigitte 原型的代理会不断地检测系统中的新报价策略。这个代理在同一时刻请求多个报价策略，并将这些请求捆绑在一起，在短时间内发出。由于 Brigitte 多次去喝咖啡，请求的数量每隔 90 分钟就会有所增长。

正如真实的保险代理一样，原型用户在它们工作的时候并发地保存业务时间和操作。当 Brigitte 智能测试代理正在快速查询方案时，基于 Stefanie 和 Ed 的测试代理也将如此。Stefanie 代理每隔 90~180 分钟就会运行一次批报告脚本。Ed 代理在新的预期客户中查询市场数据，并每隔 45~60 分钟缓慢地制定一个新的报价策略。Ed 在开始制定下一个报告单之前需要用 5 分钟抽一支香烟。

代理有助于建立指导测试的接近真实的环境。启动一个已定义的代理并且开始运行。并发地运行多个代理会增加系统的负载。高效的测试会开发 3 个或者更多的用户原型并且并发地运行每个原型代理的多个副本。例如，图 3-12 中的结果并发地用到了 30 个 Ed 代理，15 个 Stefanie 代理和 5 个 Brigitte 代理，共计 50 个代理。

原型通过声明一个用户想要通过系统所达到的目的，来模拟用户的目标。这可能会很困难，因为一些目标实际上仅仅采用部分步骤就能够达到。例如，在保险公司的示例中，Ed 代理可能有登录系统的一个目标，但这只是达到发送一个新报价策略目标的一个步骤或者只是跟踪一个预期用户接受建议的一个步骤。理解一般的代理状态对于建立有效的测试是非常重要的。

3.8.1 测试状态

智能测试代理通过原型用户来模拟系统的预期使用情况，因此，一个测试代理有可能采用多个功能步骤从而完成原型用户的目标。代理是有“状态的”。虽然每个代理随着原型的不同而变化，但是所有的测试代理都满足下列状态：

- **设置(setup)**——在这个状态下，代理设置一系列初始值以及需要与 Web 使能应用程序通信的对象。设置状态也可能包括代理实际的登录，或者执行初始化，或者系统的验证行为。
- **建立(establishment)**——代理使用一些步骤将宿主服务设置成完成行为之前所需的状态。Ed 代理登录并且查询新业务的市场调查。这些步骤足以将系统建立成行动测试的状态。
- **行动(action)**——代理请求 Web 使能应用程序创建新的对象、创建新的数据，或者修改已有的对象。Web 使能应用程序在行动的状态下执行完整的事务——与之前的步骤不同，它可以回滚到以前的状态。
- **分析(analysis)**——代理了解请求的结果的时间。代理检验响应数据以确定这个响应对于请求是否合适。
- **条件重现(conditional recurrence)**——代理查看响应数据并选取一定的行动。代理可能回到以前的状态，获得建立-行动-分析的状态。

软件测试人员非常渴望能够理解测试的状态并且建立测试脚本。虽然理解测试的状态是很重要的，但理解测试代理如何从一个状态变化到另一个状态也是相当重要的。开发人员很久之前就使用了统一建模语言(UML)，它对于确定测试代理状态来说是非常理想的工具。

3.8.2 使用 UML 和代码注释对测试进行建模

对于软件开发人员来说，UML 是用来描述软件应用中的行为和状态的。UML 对于描述测试代理的状态来说是非常理想的工具。UML 用一系列小的符号描述了系统的流程和状态。例如，图 3-14 显示了 Stefanie 代理的 UML。

UML 定义了几个简单的符号以解释程序的状态、转变、使用和功能。图 3-14 通过用例图说明了一个简单的 UML 图。包围椭圆的方框表明了 Stefanie 代理的边界。施动者符号(actor symbol)表示作为 UML 施动者的用户原型的行为，要求达到他的目标。带箭头的实线表明了用例之间的依赖关系。实线箭头的含义是一个用例依赖于它之前的用例。例如，代理直到登录系统之后才可能运行报告。

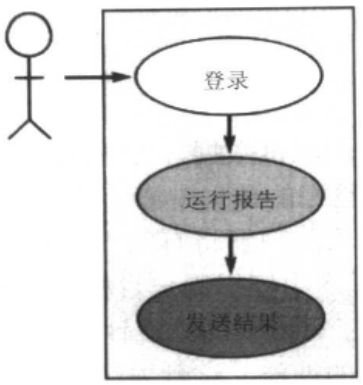

图 3-14 UML 为标注工作流提供了非常简单的方法。在这个示例中，UML 图说明了 Stefanie 测试代理的登录、运行一个报告以及发送结果的流程

UML 易学易用，许多指南和书籍都可以帮助您学习 UML。在 Web 使能应用程序测试环境中，UML 很值得学习和使用。

到现在为止，代理的行为和代理的状态都已经介绍过了，这样我们就可以开始建立系统和实例数据，为智能测试代理编码并确定 SPI 了。

3.8.3 整体考虑测试

编写代理，安装并配置服务器软件，然后定义测试状态，再编写用户原型。测试就可以开始了！当第一个代理开始的时候，测试也就开始了。准确的期限和测试的范围依赖于准则的细节。为了得到图 3-11 保险报价策略系统的测试结果，下面一些步骤是必需的：

1. 运行 50 个并发代理：30 个 Ed 代理，16 个 Stefanie 代理，还有 5 个 Brigitte 代理。
2. 运行这些代理 60 分钟。
3. 通过代理在最后 15 分钟测量系统的活动性。
4. 在一台与系统主机相连的 100Mb 以太网的中等服务器上运行这些代理。

图 3-12 的测试结果描述了系统的 SPI。这些结果给出了系统中的重要信息，也说明了怎样指导开发小组改进这个系统的软件。

虽然测试运行了 60 分钟，但是只有最后 15 分钟的结果对于测试才是最重要的。在这 15 分钟内，系统处理了 22 270 个代理操作：Ed 代理创建了新的报价策略并且在创建报告单之间间隔了较长的时间；Brigitte 代理检测报告单的有效性；Stefanie 代理向总部进行报告。SPI 是根据 SPC 对系统进行测试的。如果所有的 22 270 个代理操作都在 1 秒钟之内完成，加权响应度量将会是 89 080 个操作。系统将会长期地损失性能标记。平均起来，系统中少于 1 秒钟的响应数量占总数的 56.2%。

图 3-12 的 SPI 示例中的数据也为系统的错误进行了加权。在 22 270 个代理操作中，系统出现了 170 个响应错误。SPI 减少超过 5 秒钟完成的错误响应的权值。每个错误根据它返回结果的所用时间按比例确定权值。带权值的错误有 458 个错误响应，占整个无权值响应的 2.06%。

从 79.73%的权值响应预定值中的错误预定值减去 2.06%得到了 50.7768 的 SPI。SPI 主要是针对缓慢移动、较差性能内部网或者 Internet 系统的。它甚至可能本质上是很苛刻的。然而，50.77681 的 SPI 在保险报告单系统中是正常的，因为系统对代理请求的三分之一(33.4%)响应用了 3 秒钟或者更长的时间。通过统计这 3 秒钟或者更大的无权值响应时间，可以看到系统的性能。

3.9 本章小结

我们主要的目标是开发有意义的测试数据，这些数据可以帮助我们指导软件开发人员创建更好的软件，帮助 IT 管理员建立更好的数据中心，并使用户更好地实现他们的个人目标。对软件、数据中心、测试代理更深层次的理解将有助于测试并监测项目和小组人员。

本章列出了在 Web 使能应用程序中模拟用户目标的一种方法。通过用户目标定义了 SPC。SPC 在 Web 使能应用程序中定义了可接受的系统性能。根据 SPC，我们开发了一个称为 SPI 的标准，它可以根据准则测量系统的性能。这种方法可以应用到任何 Web 使能应用程序中。

第 4 章将介绍测试和度量 Web 使能应用程序的数据中心所需的组件。本章的 SPI 将应用到新的数据中心设计方法中，以确定一个示例系统的 SPI。

第 4 章　Java 开发和测试自动化工具

前面几章讨论了开发和测试 Web 使能应用程序的实际情况。可靠地建立并运行 Web 使能应用程序需要艰苦的工作，以及软件开发人员、QA 技术员和 IT 管理员的协同努力。服务于工业、政府和我们日常生活的每个部分的商业业务增加了软件的开发速度和 Web 使能应用程序的可靠性，从而可以大大节省资金并且创造出新的利润。

在过去 15 年中，计算机产业逐步向着自动化软件开发和测试的方向发展。我们已经经历了 3 次自动化的潮流。这使得大多数软件开发人员、QA 技术员和 IT 管理员需要对此有着深刻的了解，然后从几个相互竞争的方法学和工具之间作出选择。

本章将介绍软件自动化过程中的 3 次潮流，以及如何评估一个自动化方法学，如何组装一系列软件开发和测试自动化工具，从而交付经过改进的可靠 Web 使能应用程序。然后，第 5 章将介绍一个免费的开源测试工具 TestMaker，我编写了它的测试自动化框架和工具。

4.1　3 次潮流

从 20 世纪 70 年代的微型计算机诞生开始，计算机产业经历了 3 次主要的软件开发和测试自动化潮流。这些自动化过程在桌面应用程序、客户/服务器应用程序和目前的 Web 使能应用程序 3 者之间平行发展。表 4-1 说明了这 3 次潮流。

表 4-1　软件开发的 3 次潮流

潮　流	应 用 类 型	开 发 风 格	测 试 风 格
1	桌面	事件驱动框架包括个体过程性的功能。一般的风格是列出一系列命令的层次化菜单	根据编写的功能说明测试每个功能
2	客户/服务器	结构化编程，将命令组织成层次化菜单列表，每个客户端都有各自的代码。一般的风格是将下拉菜单按钮与图形化窗口容器控件相结合	根据编写的功能说明测试每个功能，并测试服务器的吞吐量
3	Web 使能	可视的集成开发工具简化了面向对象的设计模式。一般的风格是提供多个路径来实现任务(网络效果使得无法穿越一系列层次菜单，无法得到相同的测试结果)	获得/记录/观察如何使用一个应用程序，然后提供报告以比较重做的记录与原始记录有何不同

4.1.1　桌面应用程序的开发和测试自动化

伴随着微型计算机的发明，桌面应用程序的开发始于 20 世纪 80 年代。早期的设备在本质上是专用系统，它能够运行应用程序软件的很小一部分库，而且这些应用程序软件是在软盘上

发布的，并且通过零售软件商店和邮件定购的方式获得的。我们之中的一些人甚至可以从电子公告栏系统(BBS)的网络中进行下载。软件为信息工作人员提供了友好的界面，这些信息工作人员包括电子数据表操作员、制作报表的商业人员和游戏玩家。因为桌面计算机同其他的信息资源相连接，所以桌面软件可以分类成电子表格、字处理器、数据库和娱乐。虽然存在网络，例如使用慢速的调制解调器与电子公告栏系统这类信息资源连接，但却很少有桌面应用软件与网络信息资源的连接。极少数桌面应用程序软件用到以太网。

桌面应用程序使用键盘，之后使用鼠标通过窗口和下拉菜单进行操作。在桌面系统软件包内部，用户可以发现包括单个处理功能的事件驱动架构。应用程序从逻辑上分开管理窗口和菜单事件。应用程序命令操作系统打开一个窗口、初始化一个下拉菜单，然后等待操作系统将用户选择了菜单命令或者移动了窗口的情况告知应用程序。在本地，当操作系统“发出一个事件”时，应用软件作出响应。响应事件的桌面应用软件在本质上是过程性的。

过程性软件遵从一种简单的单一途径，从初始化开始一直到完成最后的单一目标：回应用户的命令。例如，当用户选择下拉菜单中的“打印”按钮时，操作系统运行应用程序的事件处理程序代码以打印当前文档。事件处理程序代码查找当前文档，格式化文档并将格式化后的数据发送到打印机。操作系统可能会提供多个任务——例如，在打印现有文档的同时，还允许用户创建一个新文档——但打印的事件处理程序代码仅仅关心打印文档。

使用事件驱动框架建立的桌面应用程序软件与目前面向对象的软件开发相比要容易得多。例如，Java Server Page 在单独的应用程序中为软件开发人员提供 3 个对象模型：从 Java 应用程序中绑定数据的对象模型，在服务器端动态创建表明行为对象的 HTML 标注元素的对象模型，以及为关系数据库中所保存的数据进行建模的第三方对象。在事件驱动框架中，一个事件处理程序很少需要调用它本身代码之外的函数。事件驱动设计模式对于软件开发和测试自动化工作有着非常重大的影响。

桌面应用程序的软件开发自动化注重编码和建立软件所需的时间。桌面应用程序框架为软件开发人员编码提供了一种快捷的方法。这个框架为初始化操作系统提供了库调用，并建立应用程序的初始窗口和菜单，处理大部分应用程序中通用的功能(例如，文件管理、打印和内存管理)。测试自动化主要节省用于测试一个桌面系统功能所需的时间。测试工具链接到桌面应用程序中并尝试每个命令，就像是用户访问菜单和窗口命令一样。

除了下拉菜单命令和出现在窗口中的文本元素之外，几乎不必为测试工具解释一个命令的上下文是什么。因此，大多数测试桌面系统的 QA 技术员会将所有菜单和窗口的功能与已有的功能说明文档进行比较。文档中不同的性能之处表明桌面应用程序相对的健壮性。

桌面应用程序框架和功能测试工具目前非常流行。大多数集成开发环境都包含一个新建应用程序向导——或者类似的命名工具——从它内部已有的框架中建立一个桌面应用程序的基本结构。例如，现在流行的开发工具——例如，Microsoft Visual Studio 和 Sun ONE Studio——会询问开发人员几个关于他们建立应用程序的意向问题，然后产生几百行代码实现应用程序的框架。目前流行的测试工具——例如，Mercury Interactive WinRunner 和 WinPerl Guido (http://www.winperl.com/)——在一种桌面应用程序中自动地获得、验证并重复用户的交互。这些测试工具为 QA 技术员提供了一种自动化的方法，它能够确定缺陷，为用户保证业务处理的正确性和可靠性。

4.1.2 客户/服务器应用程序的开发和测试自动化

客户/服务器应用程序的原始目的是将表示逻辑同业务逻辑分离。在一个理想的系统设计中，客户端负责表示用户的界面、命令元素(下拉菜单、按钮和控件)，并显示一系列窗口、图表、饼状图的结果信息。与服务器相连的客户端处理功能，而服务器对数据作出响应。在实际中却很少有这么明显的分工。客户/服务器设计将功能部署在服务器和客户端上。目前的客户/服务器可以认为是与中心服务器通信的通用桌面应用程序。因此，客户端代码可能与服务器上的代码一样复杂。

因为客户/服务器应用程序的集中特性使得系统的维护、部署和开发具有很好的性质，因此客户/服务器应用程序非常流行。客户/服务器应用程序开发为引导第二次工具和方法学潮流的软件开发和测试自动化提供了新的可能性。

在客户/服务器环境中，需要明确地定义协议，从而使所有的客户端都使用相同的协议与服务器进行通信。这简化了代码重用和远程数据源中提供的信息，并为智能客户功能提供了可能性。相应地，软件开发自动化工具销售商扩展他们的框架以提供客户端和服务器端的两种框架。客户端框架提供了与桌面应用程序框架相同的功能，而且还提供了与服务器通信所需的命令代码以及从服务器接收自动更改客户端的代码。服务器端框架提供了从多个用户接收到的和需要处理请求的代码，还有为数据持久性与数据库和远程信息供应商连接的代码。另外，这些框架需要处理有状态的事务和间断的网络连接。有状态的事务需要多个步骤才能完成一项任务。例如，登录到一个 Web 使能应用程序，因此就不必在每个子任务中再次登录。

客户/服务器应用程序通常在本质上是具有事务性的，而且需要与用户进行几次交互才能完成一个单独的请求。例如，在股票交易应用程序中，用户执行的事务是首先验证其在服务器上的身份，查看订单代码，然后提交购买股票的请求。客户端应用程序将请求发送给服务器，接收结果并将其呈现给用户。客户端应用程序通常了解一些事务——例如，客户端应用程序在正常情况下将保存用户身份和会话代码，就像通过用户与基于服务器应用程序交互的记录值。因为一个请求中的每个步骤都可以在客户应用程序中进行优化，所以当客户端应用程序了解事务的时候，用户更希望出现这种情况。例如，在之前股票交易的示例中，客户应用程序在不需要与服务器进行通信的情况下就能够在本地计算出一次股票交易的委托。

客户/服务器应用程序测试自动化提供了比桌面应用程序测试自动化更多的功能，主要包括：

- 客户/服务器应用程序在网络环境下运行。测试不仅需要检验一个应用程序的功能，还需要测试应用程序处理为何缓慢或者为何会出现间断的网络性能。
- 自动的测试对于确定客户应用程序的数目是非常有效的，服务器可以在任何给定的时刻有效地对客户应用程序进行处理。
- 服务器通常位于客户应用程序与几个数据源之间的中间层。在与数据源进行通信的时候，自动测试需要检验服务器功能的正确性。

客户/服务器应用程序带来了与桌面应用程序测试自动化相同的挑战：在客户与服务器之间很少能够表明测试自动化工具的请求上下文。它看起来仍然像是在网络中流动的位和字节，而不是股票交易事务、购买机票的请求以及房屋抵押的研究。这给 QA 技术员测试一个客户/服务器应用程序带来了巨大的麻烦，这是因为测试需要回答下面这些问题：

- 在客户应用程序中，将显示给用户的所有菜单和窗口命令的功能与已有的功能说明文档进行比较。
- 确定增加并发用户请求数目对服务器性能的影响。
- 当某个客户发出请求时，确定他对其他客户功能的影响。

因为测试自动化大大提高了 QA 技术员的生产效率，所以它在客户/服务器环境下是非常理想的工具。然而，由于客户/服务器应用程序更加难以开发，因此测试自动化就变得更加必要。

客户/服务器应用程序框架和功能测试工具目前非常流行。大多数企业集成开发工具都包含根本的客户/服务器框架。目前流行的工具——例如 Borland Jbuilder、IBM Rational Rose，以及 Compuware OptimalJ——都为客户和服务器组件提供了编写代码的框架。目前流行的测试工具——例如 Segue SilkTest、Empirix e-Test、Mercury Interactive LoadRunner，以及 RadView Web FT——提供了工具集，它们可以指导在客户与服务器之间协议的测试自动化工具，验证交互，引导服务器的测试，然后监测系统的服务质量。

4.1.3 Web 使能应用程序的开发和测试自动化

Web 使能应用程序带来了软件开发和测试自动化的第三次潮流。Web 使能应用程序建立在开发、部署、维护桌面应用程序和客户/服务器应用程序的基础之上。Web 使能应用程序在以下领域进行了扩展：

- Web 使能应用程序意味着无状态。HTTP 的设计就是无状态的。来自 Web 使能应用程序的每个请求是原子的，而且与以前的请求没有关联。这对于系统的体系结构和数据中心的供应是非常有好处的。虽然请求是无状态的，但任何服务器都可以响应这个请求而且在任何服务器上的每个请求处理器都可能为该请求提供服务(当然，在本书的下一个部分我们将会看到，没有哪个应用程序是完全无状态的)。
- Web 使能应用程序是与平台无关的。客户应用程序可以在 Windows、Macintosh、Linux，以及其他能够实现命令协议和服务器网络连接的操作系统上进行开发。
- Web 使能应用程序希望客户应用程序提供显示功能和简单脚本的能力。客户应用程序通常是浏览器，但是也可以是专门的客户应用程序，例如零售现金出纳机、基于 Windows 的数据分析工具，或者移动电话中的电子地址通讯录。

因为 Web 使能应用程序为客户/服务器提供了许多优势，所以 Web 使能应用程序在商业中是非常流行的——高效的系统维护、部署以及开发——另外，它们降低了客户端软件维护的负担，为大量的小服务器增加了数据中心体系结构的灵活性，而且成熟地应用了面向对象编程技术。

面向对象编程对于软件开发人员非常有吸引力，这是因为这种编程技术可以在不考虑整个系统的情况下很方便地交付各自的软件对象。每个对象负责管理并封装各自的数据。另外，软件对象的应用编程接口为对象的调用提供了上下文。例如，StockTransactions 对象有一个 BuyStock 方法，它接收一个变量名是 Shares 的整数值并返回一个 TransactionNumber 长整数，该方法为软件开发人员提供了理解对象的目的以及如何调用的能力。遗憾的是，却没有为对象上下文作出解释的等价测试自动化的工具。

在Web使能应用程序的测试自动化中遗漏的上下文表明了软件开发人员和QA技术员必须为每个 Web 使能应用程序手动编写测试。另外，他们还要随着应用程序的改动继续维护这些脚本。Web 使能应用程序测试自动化工具的主要目的是将脚本的编写和维护任务变得更加容易。流行的测试自动化工具提供了以下功能：

- 一个友好的图形化界面，并集成了记录、编辑和运行脚本的功能。
- 一个记录器，利用它可以查看应用程序是如何使用的，并且可以编写测试脚本。
- 一个重新操作(playback)的工具，它可以通过处理测试脚本和记录日志来驱动 Web 使能应用程序。该工具也可以重新并发地运行检验系统的可扩展性和负载测试的相同脚本的副本。
- 一个报表工具，它可以表明重新操作与原始记录有什么差异。差异可能是较慢或者较快的性能时间、错误以及未完成的事务。

Web 使能应用程序的框架和功能测试工具是非常有利润的产业。然而，目前所有的流行工具——例如，商业工具中的 Wily Introscope、Mercury Interactive LoadRunner、Segue SilkPerformer、RadView WebLoad 以及 Empirix eLoad，还有开源工具中的 PushToTest TestMaker，The Grinder 以及 Apache Jmeter——都提供了上面的 4 个基本功能：图形化界面、记录器、重新操作和报表工具。它们都需要编写并且维护测试脚本。

对于测试脚本的维护问题，有两个尚需完成的重大工作：自动生成测试的应用程序建模工具和通过 XML 定义 Web 使能应用程序中工作流模型的 Web 服务第二代标准。

1. 从模型到测试

许多软件开发和测试自动化工具的公司都发布了建模语言，这些语言可以为软件设计师和开发人员描述已有的软件应用程序。例如，Compuware OptimalJ 实现了对象管理组织(OMG)的模型驱动体系结构(MDA)。MDA 使得您能够可视化地描述一个应用程序，而且 OptimalJ 能够自动编写实现应用程序的 Java 代码。MDA 描述告诉 OptimalJ 应用程序中每个函数的所有上下文。因为它们可以将模型正确地转化为代码，因此产品很自然地也将模型转化为验证应用程序的函数、可扩展性以及性能的测试组件。

模型驱动系统的另外一个示例是 IBM 的 Rational Rose。它实现了 OMG 的统一建模语言(UML)。UML 是定义、可视化、建立和文档化软件系统产品的工业标准语言。UML 使软件开发人员为软件构建“蓝图”。之后，IBM Rational 的产品就可以根据这个蓝图编写测试组件。

使用商用软件开发和建模工具的不利因素是销售商一直垄断着这个产品。如果 IBM Rational 无法满足对建模工具的改动需求，那么就需要用其他工具重新进行编写——再为其他的新工具进行投资。

2. 从工作流到测试

几种并行的努力正在进行之中，以提高软件开发人员在 Java 平台下的 Web 使能应用程序方面的产出率。考虑表 4-2 列出的技术和工具。

表 4-2 使得 Web 使能应用程序的开发更加高效

项 目	描 述
Java Server Faces	为简化 Java Server 应用程序 GUI 的创建和维护而定义的体系结构和 API。使得 Web 页面最终可以成为一个 Web 使能应用程序。详细说明请访问 http://web1.jcp.org/en/ jsr/detail?id=127
Java Page Flow	一个工具集，用以建立基于 Struts 的 Web 使能应用程序编程模型，它可以调节 Struts 的功能和可扩展性，同时也减少了建立基于 Struts 应用程序的难度和挑战。Java Page Flow 包括对 Web 使能应用程序编程模型的运行环境的支持以及使得开发人员迅速、方便地建立基于该模型的应用程序的工具 Java Page Flow 的核心概念和结构是页面流。一个页面流在本质上是 Web 应用程序文件的目录，这些文件可以协同实现一个 UI 功能。详情请访问 http://developer.bea.com/articles/Dew.jsp
BPEL4WS	Web 服务的业务流程执行语言(BPEL4WS)是 OASIS 组的一个开发标准，它定义了基于 XML 业务流程执行语言的公共标准，为包括需要业务协议和可执行过程模型的多使用模式建立了必要的技术基础。详情请访问 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

表 4-2 列出了提高软件开发人员产出率的数十个努力中的 3 个，所采用的方式是将建模语言和表示法引入软件的开发过程。这对于测试自动化是非常有利的。为测试这些应用程序的自动生成的测试脚本带来了服务器的工作流表示法。即使没有这些新的工作流表示法技术，我依然相信这是软件开发与测试自动化工具和技术的第四次潮流。

4.2 troika——第 4 次潮流

我处在这样一个位置，在我日常的工作中经常与软件开发人员、QA 技术员和 IT 管理员进行协作。以我个人的经验，这 3 个小组很少作为一个单独的开发团队进行工作。当他们协作时，可以试想出他们形成了 troika！troika 来自一个俄语单词，它描述了作为一组的 3 匹马并排拉车的情景。如果获得了帮助，就可以很轻易地承担重的负载。设想会有 3 倍的工作用于建立、部署和维护一个 Web 使能应用程序(图 4-1)。第 1 章讨论了当今业务中的一些公共的问题，这些问题使得软件开发人员、QA 技术员和 IT 管理员形成了 troika。使用本书的技术、方法学以及工具，可以预计在开发和测试自动化中创新与生产的第四次潮流。

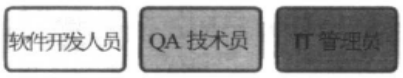

图 4-1 测试 Web 使能应用程序最高效、生产率最高的方法是将软件开发人员、QA 技术员和 IT 管理员协同起来，从而可以增加应用程序的功能、可扩展性以及可靠性。协同工作的这 3 个组构成了 troika

以我个人的角度，我们处在一个惟一的时代。这是第一次软件开发人员、QA 技术员和 IT 管理员都支持建立分布式应用程序框架和基础架构：面向对象的应用程序，在路由全局网络上运行，使用 Web 基础架构。但是，这些仅仅考虑了目前应用程序软件的复杂性而已！

在服务器端，J2EE、.NET 以及开放的 API 技术(例如 Apache Tomcat 引擎、Struts 库和 JBoss 应用服务器)为我们提供了丰富的 API：从 Web 页面、数据库访问、Web 服务互操作，到异步消息队列和电子邮件队列等。相应地，我们可以建立功能强大的应用程序，这些应用程序能够协调一个企业基础架构的许多部分。对于这些 API 和协议，我们提出了表 4-3 中的一些问题。

表 4-3　最基本的问题

角　色	问　题
软件开发人员	在新的体系结构系统中，我将如何测试新模块
QA 技术员	我将如何测试系统的可扩展性和功能
IT 管理员	我将如何测试系统的可靠性和可用性

在项目开发中的每个人都要问：我将如何测试这个应用程序呢？应该知道单独的客户端 API 不像 J2EE 在服务器上那样能够测试应用程序的所有部分。

作为一名软件开发人员，我发现单元测试可以很好地保证我所编写的服务器端软件组件可以按我的想法去执行。但是，我也发现一些组件在单元测试之前需要有正确的状态。例如，一个 Java Bean 在放置 100 个新订单、需要进入 100 个订单中进行测试的时候，发出一个通知电子邮件消息！我需要下列支持才能够自动化这样一个测试：

1. 一个可以编写测试的框架。
2. 支持我所使用的开发标准环境。
3. 查看服务器性能的多协议支持，至少包括 Web 协议(HTTP、HTTPS)，Web 服务协议(SOAP、XML-RPC)，以及电子邮件协议(SMTP、POP3 和 IMAP)。
4. 每次我更改模块时，一个进行测试自动化的工具。

设想我将这些技术、代码库以及工具软件进行简单的修补就可以完成这 4 个目标。其结果是测试自动化框架和工具能使我快速地编写测试脚本，从而自动地将 Java Bean 转化为正确的状态。另外，这些测试自动化脚本实际上扩展了我做过的像 JUnit 那样的单元测试框架。

通过使用这种方法，我发现与我一起工作的 QA 技术员用我的测试脚本对功能进行验证，在相同的测试环境中并发地运行它们以检验系统的可扩展性、并发性并恢复问题。当一名 QA 技术员在我的代码中找到问题的时候，我们一同查看测试脚本日志，这样可以定位问题的位置并且了解导致问题的原因。我所需要的是建立、运行并分析这些测试的一种方法，该方法可以将 troika 中的三分之二——软件开发人员和 QA 技术员——联合起来！

另外，IT 管理员会经常采用我的功能测试脚本并且长时间地运行它们。这些脚本通过使用与实际用户相同的内部协议驱动服务。脚本记录了系统运行的情况。而且可以为管理员和客户做出很好的服务质量报表。啊！我们拥有了 troika3 个部分的全部：软件开发人员、QA 技术员和 IT 管理员一起协同工作！

从这些经验中，我发现这种建立 troika 的方法在目前是可以实现的。我们可以从客户端为测试 J2EE、.NET 以及开放的 API 应用程序建立一个非常优秀的框架和工具。下一节将介绍一种使用 Web 使能应用程序的方法，通过该方法可以获得更多的客户并且降低业务处理的成本。

4.3 测试自动化生命周期

虽然正确的测试自动化工具、技术以及方法学将会构建出本章上一节提到的 troika，但自动化一个测试包括许多步骤。本节将为测试自动化列出可重复生命周期的这些步骤。

为了表示这个生命周期，设想在 Florida 的一所大学，这所大学打算向其 15 000 名学生、教育人员以及管理机构提供一个新的电子邮件通信系统(第 15 章将会列举这个示例)。三分之一的学生在校园内生活。其余的往返于学校或者从远程卫星课堂上学习。该电子邮件系统通过电子邮件客户软件和基于浏览器的 Web 界面对电子邮件账户进行访问，这些客户软件可以是 Microsoft Outlook 或 Mac OS X Mail。这所大学已经选择了 Web 基础架构组件为这个电子邮件通信系统提供服务，包括商用电子邮件软件包的使用，还选择了与电子邮件系统连接的 3 个 Internet 服务供应商。该大学打算验证其数据中心和带宽分配是否可以为学生、教职人员和管理机构提供服务。

图 4-2 表明了部署这所大学电子邮件通信系统的开发生命周期。通过研究这个生命周期，我希望您会看到这个生命周期可以应用到任何 Web 使能应用程序的测试之中。

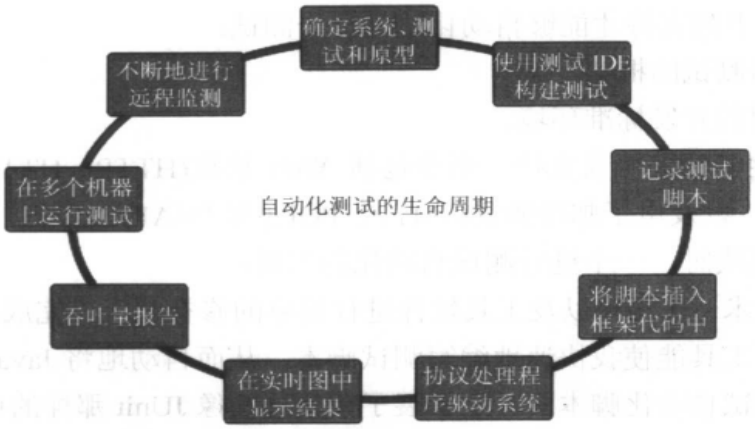

图 4-2 创建一个 Web 使能应用程序的自动测试步骤

测试的生命周期由几个连续的过程组成。每个过程建立在前一个过程的基础之上。生命周期测试电子邮件系统的可扩展性、功能以及性能。通过重复这个生命周期，我们还根据大学对系统作出的改动和升级测试了恢复性(以前解决的错误)。下面将详细地描述每个步骤。然后在第 5 章中将说明如何使用 TestMaker 免费开源测试工具完成这些过程。

1. **确定测试目标、系统组件以及用户原型**。这所大学打算验证其数据中心和带宽分配可以为学生、教职人员和管理机构提供服务。系统组件包括一个路由网络，提供 Web 界面的 3 个服务器的负载平衡组和在集群模型中运行商用电子邮件包的 4 个服务器。该系统在 3 个供应商提供的网络上运行，这些供应商有 Florida Telephone、TimeWarner 和 Verizon。大学确定了 3 个

原型用户，包括在校园生活的学生 Marilyn，在校外生活的学生 Muriel，以及学校的教师 Betsy (请注意：测试和用户原型在第 15 章中会详细地说明)。

2. **使用集成的测试环境(ITE)创建测试。** ITE 将图形界面、脚本编辑器、记录和重新操作的工具以及记录/实时图等工具集成起来，这样提供了方便使用的编程环境。虽然您可能选择一些工具用以创建测试，但我发现 ITE 可以加快生产率，很值得在 ITE 上花费时间和精力。在生命周期的这个部分中，ITE 自动创建目录，保持测试的情景以及实现原型行为的测试代理。

3. **记录测试脚本，该脚本可以通过浏览器界面驱动应用程序。** ITE 带有记录器的功能，该功能可以通过使用 Web 浏览器查看 Betsy 完成的任务。该记录器编写测试脚本，该脚本重新模拟 Besty 的行为，它通过使用 Web 界面检验 Besty 的电子邮件信箱并且发送消息。

4. **编写测试脚本，该脚本可以通过 SMTP 和 POP3 电子邮件界面驱动应用程序。** ITE 带有脚本命令以实现 Muriel 和 Marilyn 日常的任务。测试脚本重新模拟 Muriel 和 Marilyn 的行为以检验并发送电子邮件消息，就好像 Microsoft 的 Outlook 使用了 SMTP 和 POP3 电子邮件协议一样。这些脚本进行简单的功能测试。一旦运行，这些脚本将对单个的邮箱进行一次检验并且随机发送一个简单的电子邮件消息。

5. **将测试脚本插入测试框架中从而实现一个完整的测试组件。**ITE 为进行功能测试(从步骤 3 和步骤 4)并且为运行它们的可扩展性、性能、恢复和功能测试提供了一个测试框架。使用这个框架，可以确定创建虚拟用户的数目，而且可以确定不同原型行为的混合。例如，这所大学想要在 250、500 和 1000 个并发虚拟用户的情况下测试该电子邮件系统。对于每个测试，他们都希望 60%的虚拟用户具有 Muriel 的行为(在校外检验他们的电子邮箱)，30%的虚拟用户具有 Marilyn 的行为(在校内检验他们的电子邮箱)，以及 10%的虚拟用户具有 Besty 的行为(具有发送和接收大文件附件的能力)。这个框架从属性文件搜索设置，自动地运行并且进行一系列的测试。

6. **使用应用程序的内部协议运行测试。** 测试脚本通过使用内部的 HTTP、SMTP 和 POP3 协议驱动这所大学的电子邮件系统。为了测试系统，用一个真实的用户驱动该电子邮件的功能。实际上，测试脚本语言具体调用一个扩展的协议处理程序库，该库实现了很多协议而且很容易扩展。

7. **在一组实时图中表明测试的行为。** 在进行测试的时候，我发现在一组图表中查看实时测试结果的图形化显示是非常有用的。拥有测试结果矩阵的图表可以很快说明我在测试过程的基础架构上作出的改动对可扩展性和性能的提高。例如，如果吞吐量开始下降，我可以更改路由器的设置，为服务器增加更多的带宽，然后在图表中查看改动的结果。

8. **将结果记录在统计报表中。** 测试产生了大量的日志数据，这些数据可以归纳为一个或者多个报表。对于可扩展性的测试，报表会说明事务/秒、平均吞吐量、错误的指数以及可扩展性的指数。对于可接受的测试，报表会说明应用程序的测试是成功了还是失效了。对于回归测试，报表会说明以前改正的错误在应用程序中是否会再次出现。前面几章介绍了统计报表的方法，包括第 2 章的 Web Rubric 和第 3 章的可扩展性和性能指数。

9. **使用多个测试机器运行大规模的可扩展性测试。**在第 6 步运行的测试使用了一台机器运行测试框架。以我个人的经验，一个廉价的 1.4GHz 与 Intel 兼容的系统通常会在有 300~500 个并发运行测试代理的时候出现带宽和资源的溢出。为了获得 1000 个并发虚拟用户，这所大学

需要将测试安排在多台机器上。测试环境需要在多台机器上同时启动测试，在测试运行期间管理多台机器，而且从多台机器中获得并分析测试结果并生成一个报表。

10. **不断地远程监测应用程序**。这所大学从多个地点，在多个网络中运行测试框架，从而可以了解系统为用户提供了什么样的服务。测试框架运行测试代理脚本，就好像校外的 Muriel 从家中进行连接，Besty 在校园内发送消息一样。另外，这些测试说明了 ISP 连接的电子邮件服务执行得非常好，还证实了管理系统的服务质量。

测试自动化生命周期定义了在 Web 使能应用程序上执行可扩展性、功能、性能和回归测试的一系列步骤。在新创建的软件中插入基础架构和基础架构发生变动时(例如当增加新的带宽或者服务器配置发生变化时)，生命周期会再次开始。生命周期的步骤可能使用任何商用或者开源的测试工具。

4.4 本章小结

本章描述了从 20 世纪 80 年代以来的自动化的 3 次主要潮流。每次潮流都带来了一系列新的开发、测试自动化方法学以及投入市场的工具。目前人们仍然在使用它们。这给大多数软件开发人员、QA 技术员和 IT 管理员在理解、选择几个有竞争力的方法学和工具方面带来了一定难度的挑战。本章描述了软件自动化工作的 3 次潮流，如何评价一个自动化方法学，如何将一系列的软件开发和测试自动化工具组合起来以提交可靠的 Web 使能应用程序。在第 5 章中，我们将了解 TestMaker。它是我编写的开源测试工具，而且可以作为测试自动化框架和工具，并且可以满足建立并测试可扩展的、可靠的、用 Java 编写的 Web 使能应用程序的需求。

第 5 章　从方法学到设计的途径

信息系统已经达到了一定的复杂程度，这样就需要人们相互协作才能够建立、运行和监测几乎所有新的信息系统。为了回答这个问题，本书的第一部分说明了软件设计、测试和监测需要新方法学的原因。然后，介绍了在 Web 使能应用程序环境中帮助软件开发人员、QA 技术员、IT 管理员进行设计的方法学和策略(第 4 章称之为 troika)。本章将讨论与本书第二部分有关联的方法学和策略，其中列出了可用的技术、协议和工具——大多数是免费的——从而可以设计、实现并监测能够很好执行且可扩展的信息系统。

就我个人而言，我发现通过阅读基本原理、了解技术出现的错误、然后通过使用技术从而逐渐掌握它们，可以很容易地学习新技术。为了掌握它们，我使用了一系列通用的工具，包括我所创建的并在日常生活中不断维护的一个开源工具 TestMaker。本章将为您介绍一系列工具，它们可以使您快速了解一个信息系统的可扩展性和性能。

5.1　实现 troika 的工具

我从 1998 年就开始收集验证信息系统可扩展性、性能和功能的工具。目前，我是 Sun Microsystems 公司 Sun Community Sever(SCS)体系结构的主要设计师。SCS 是一个 Web 使能应用程序的集成服务器。SCS 是 Sun 公司在使用 Web 进行消息传递、电子商务、协作服务器软件上的一次率先尝试。SCS 通过 Web 页面、电子邮件、在线会议界面、聊天、智能设备提供了对公共和私有内容的集成访问。SCS 将协作数据库和目录服务器集成为单点登录功能。例如，通过电子邮件或者 Web 界面，SCS 可以方便地启动一个用户参与的讨论小组。当一个消息提交到讨论小组的时候，通过电子邮件进行参与的用户会收到一个包含提交信息的标准 Internet 电子邮件消息。关于这个电子邮件消息的回复以正确的线程次序回送到这个讨论小组。Web 的参与者会在讨论小组的浏览器中看到电子邮件回复消息的显示结果。SCS 的初始实现获得了很好的效果，像 BP、Baker Hughes 公司、甚至是 Sun 的全球销售组都使用 SCS 为雇员提供丰富、多协议协作的操作。

当 SCS 开始验证负载下的可扩展性和性能的时候，我需要这样一个测试工具，它可以检测通过 Web 界面或者其他方式提交的电子邮件消息。下面是我所需要的测试工具的一些功能列表：

1. 编写单元和功能测试的框架。同一个测试可以容易地重用可扩展性、性能和回归测试。
2. 支持开放标准的应用程序开发环境和多种平台。
3. 匹配服务器性能的多协议支持，可以引入并使用新协议。起初，我需要用于 Web 页面接口的 HTTP 和 HTTPS 协议，以及用于电子邮件功能的 SMTP、POP3 和 IMAP 协议。之后，还需要 SOAP 与 XML-RPC 协议。最后，当它们升级的时候，需要一种添加新的协议处理程序。
4. 一种使得记录、编辑和运行测试自动化非常容易的工具。
5. 在测试运行期间以及运行之后，可以快速了解测试结果的方法。

6. 在服务器组中运行功能测试的框架——即可以在实验室内部也可以在 Internet 的远端——以分布式的方式将测试结果记录到一个公共的位置。

如果上述还有些欠缺，但我的能力也只能做到这些了。如果还需要什么的话，那么只有花钱去买工具了。

我发现有十几个测试工具可以满足我的需求。我在 http://www.pushtotest.com/ptt/wiki/TestToolsInfoBook 上提供了商用和开源测试工具的更新列表。编辑这个测试工具的列表能够看出测试工具的组合对于有效地测试 Web 使能应用程序的可扩展性、性能和功能是非常必要的，我还没有发现某个单独的工具可以完成所有的目标。恰恰相反，我发现许多工具能够相互协作。我并不使用某个单独的工具，而是组装了一个工具箱。表 5-1 说明了我添加到工具箱的测试工具和库以及添加的原因。

表 5-1 Frank 工具箱的内容

所需的功能	我选取的工具/库	选取它的原因	我放弃了什么
在 HTTP 传输之上的 SOAP 请求和响应	Apache SOAP *http://ws.apache.org/soap/*	API 易学易用。支持 SOAP RPC 和文档化的编码。另外，还有一些其他的替代工具(Java JAX，位于 http://java.sun.com/web-services/；Apache Axis，位于 http://ws.apache.org/axis/，目前还不可用)	Apache SOAP 和其他 SOAP 栈相比，速度很慢。它不支持 WSDL，以及在 Java 消息服务(JMS)数据源上的 SOAP
脚本语言	Jython *http://www.jython.org*	功能完全且面向对象。可以在 Java 运行的地方执行。对 Python 对象以及任何 Java 对象进行完全访问	通过在众多语言中选择一种语言，排斥了使用其他语言的用户
将浏览器的行为记录到 Jython 脚本中	MaxQ *http://maxq.tigris.org*	稳定的代码库。经常维护。容易编写自己的测试代理脚本	设计妨碍了 MaxQ 记录使用 SSL 和 HTTPS 的浏览器操作
在 SWING 控件中的柱状和线形图表	JOpenChart *http://jopenchart.sourceforge.net*	与其他的图表库不同，jOpenChart 实现了事件模型，以随着底层数据的变化而更新 Swing 控件	我没有什么顾忌的
HTTP cookie 处理	jCookie *http://jcookie.sourceforge.net/*	处理 IETF RFC 2965 和 Netscape 风格的 cookie。具有良好、轻量、鲁棒和文档化的性质	我没有什么顾忌的
XML 数据处理	JDOM *http://www.jdom.org*	处理 XML 数据非常方便，很像 Java 的 API。避免了一般 XML 分析库的复杂性	XML 分析库变动很快，选择某一种后，很容易过时
高性能多维数字矩阵	JNumeric	对于统计分析效果很好	我没有什么顾忌的

(续表)

所需的功能	我选取的工具/库	选取它的原因	我放弃了什么
对 SMTP、IMAP 和 POP3 电子邮件的支持	Java Mail API *http://java.sun.com/products/javamail*	为 Java 开发人员提供了使用电子邮件协议的标准库	灵活，Java 邮件 API 不易使用和集成
XML-RPC 协议处理程序库	Apache XML-RPC *http://ws.apache.org/xmlrpc/*	为使用 XML-RPC 协议的主机提供了简单、方便的 API	我没有什么顾忌的

表 5-1 中的所有工具都有一些共同之处：它们都完全用 Java 编写，而且是通过免费开源许可进行发布的。这使得创建框架和工具应用程序更容易，应用程序可以集成这些库和工具。我把这些相关的工作称为 TestMaker，并且将它作为测试 Web 使能应用程序的免费开源测试工具进行发布。TestMaker 目前可以在 http://www.pushtotest.com 上免费下载。

通过在开源许可下发布 TestMaker，许多软件开发人员已经在这些年提供了其输入、指南以及体系结构上的建议。根据其概念，TestMaker 经历了 3 个主要的版本。初始的软件为 Java 开发人员提供了一个在测试逻辑中添加代码的一个简单框架。例如，TestMaker 的 1.0 版本提供了 Java 对象的一个框架，它可以向一个 Web 主机发出 HTTP 调用。Java 程序员可以像编写 Java 对象那样编写测试代理的业务逻辑。但这种方法很不稳定。当目标的 Web 主机发生很小变动的时候，测试对象的 Java 代码也要发生相应的变动。

这种方法还存在另外一个问题，用 Java 建立测试的复杂性对于许多 QA 技术员和 IT 管理员来说是非常巨大的，这是因为它需要用 Java 编写并且维护面向对象的测试代理的代码。因为这个原因，TestMaker 的 2.0 版本引入了基于 XML 的脚本语言。下面就是其中的一个示例：

```
<load>
    <script>
          <url action="new" name="myurl"
               host="http://www.pushtotest.com"/>
          <url name="myurl" action="get"
               document="index.html"/>
     </script>
</load>
```

这个脚本创建了一个 URL 测试对象，该对象与主机服务器进行通信并且请求 index.html 页面。TestMaker 2.0 版本的脚本语言包括条件、循环以及简单的表达式。用户很愿意使用脚本语言，但他们更希望这种脚本语言具有更强大的功能。

Geoff Lane 是 TestMaker 的主要项目成员(也是本书的技术编辑)，他提出在 TestMaker 2.0 版本中的 XML 脚本语言应该像流行的 Python 脚本语言那样。我已经决定不再将 TestMaker 中的 XML 语言建立且维护成为一流的语言(增加对通用表达式、全布尔逻辑的支持，脚本语言的所有其他缺点简直就如同噩梦一般)。Python 已经具备了这些语言的所有功能。

Geoff 在后来发现了 Jython，它是 Python 完全用 Java 编写的版本(关于 Jython 的详细情况请访问 http://www.jython.org)。Jython 将 Python 语言编写的脚本编译成 Java 字节代码，因此可以在 Java 虚拟机上运行。使用 Jython 编写的测试脚本可以在 Windows、Unix、Macintosh 以及其他运行 Java 的系统下执行。另外，测试脚本可以直接访问全部的 Python 对象，而且可以访问在 Java classpath 下的任何 Java 对象！使用 Jython 脚本可以调用表 5-1 中列出的任何库。在本章的后面，我会把 Jython 的全部优点介绍给需要设计、编码和测试信息系统的人们。TestMaker 的 3.0 版本替代了 XML 脚本语言和 Jython 原有的测试对象以及协议处理程序的测试面向对象库(TOOL)。TOOL 为需要协议、工具、库的开发人员提供了一个通用且可扩展的编程接口。

5.2 怎样获得 TestMaker

TestMaker 可以在 PushToTest 网站上免费下载。在浏览器中输入网址 http://www.pushtotest.com，就可以进行下载了。该网站提供了最新版本的软件和它的旧版本，而且在网站上可以咨询关于 TestMaker 的问题，另外该网站还提供了其他的技术支持服务，其中包括了对用户和开发人员的电子邮件的服务支持。

TestMaker 以 Zip 压缩文件的方式提供。将文件进行解压缩就可以安装需要运行 TestMaker 的任何部分。TestMaker 实际是一个 Java 应用程序，在 Java 1.4 或者更高的版本上都可以运行，包括 Windows、Linux、Solaris 和 Macintosh OS X。其他的基于 Java 的平台也可以运行 TestMaker，只不过还没有经过测试。

TestMaker 以编译后的二进制文件和 Java 源代码两种方式发布。二进制文件可以立即执行。从源代码中创建 TestMaker 需要流行的开源工具 Ant build/make，它可以通过 http://jakarta.apache.org/ant 获得，并且需要 Java 1.4 或者更高的编译器。

TestMaker 通常每月更新一次，并提供错误的修复和新功能。PushToTest 网站提供了免费的电子邮件公布服务，它在新版本出现的时候会发送更新的电子邮件消息。PushToTest 并没有发布或共享其电子邮件地址。

TestMaker 和开源过程

TestMaker 为我提供了开源产品开发的环境。我深刻地感到我从经验中得到了很多益处。开源项目是众多软件开发人员齐心协力的结果，他们组成开源社区，一同解决实际问题。当有功能遗漏或者发现错误的时候，用户会感到它比商用软件有更多的选择。开源项目将其源代码公开，这样任何工程师都可以解决其中的问题或者添加新的功能。根据开源许可的形式，由于项目本身的一体性，其改进应该公布到项目社区中。

TestMaker 具有开源许可，它是仿造 Apache Group 许可而来的。Apache Group 对于建立并维护 Apache Web 服务器和其他开源项目是非常成功的。这些项目提交高可用的代码，它可以用于自身或者内置于其他产品。TestMaker 许可可以在 license.html 文件中找到，并与 TestMaker 一同下载。该许可允许任何人免费下载并使用 TestMaker。该许可甚至使用 TestMaker 源代码创建其他新产品，包括商用软件。

我发现 TestMaker 社区分成了两个群体：用户和开发人员。PusthToTest 网站为用户和开发人员分别提供了 user@lists.pushtotest.com 和 dev@lists.pushtotest.com 的电子邮件列表，他们可

以提交问题、共享改进、彼此相互支持。用户列表是为想要对使用 TestMaker 测试 Web 使能应用程序有更多了解的人们而设置的。开发列表是为对 TestMaker 代码作出改进的人们而设置的。

开源项目一般需要一个主要的维护人员，他要决策项目的方向，选择适合项目的改进，组织人员解决问题并提供解决方案。在我编写本书的时候，我是 TestMaker 的主要维护人员。TestMaker 的新成员提交改进意见，包括将改进 TestMaker 的代码发送到 dev@lists.pushtotest.com 的电子邮件列表中。提交的确定要根据代码的技术熟悉程度以及与其他项目成员讨论的结果而定。在提交并接受几次改进之后，可以将这个开发人员添加到能够修改 TestMaker 代码的特权人员列表中。在本书编写完成之后，共有 5 人具有特权级别，28 名工程师参与了改进意见。

虽然 TestMaker 的测试和发行版本可以通过 PushToTest 网站获得，但最新的版本以及临时的 TestMaker 源代码却保存在 cvs.pushtotest.com 的同步版本系统(CVS)服务器上。CVS 的任何客户都可以从 CVS 服务器上下载最新的 TestMaker 源代码。CVS 服务器上的源代码可能包含错误且尚未完成的功能、或者一些测试实验；然而，CVS 中的代码需要编译。这里列出了从 CVS 下载的 TestMaker 最新的代码参数：

```
CVSROOT =  :pserver:anonymous@cvs.pushtotest.com/var/cvsroot
```

不用密码就可以登录到 CVS 上，并检查如下模块：

- 工具——接收协议处理程序的 TOOL 库和测试对象
- tm4——基于 Swing 的 TestMaker 应用程序

5.3 五分钟学会 TestMaker

TestMaker 是建立智能测试代理的易用工具。下面一节将介绍建立一个测试代理所需的步骤，包括如下几个步骤：

- 在 Windows 或者 Linux 计算机上安装 TestMaker
- 了解 TestMaker 图形环境
- 记录一个测试代理
- 理解 Jython 脚本语言和 TOOL 对象

在 Windows 或者 Linux 下安装 TestMaker

TestMaker 将一组文件压缩成为 ZIP 文件的形式。为了安装 TestMaker，需要将该文件解压缩到一个目录下。TestMaker 文档将该目录指定为 testmaker_home，并将 TestMaker 文件解压缩到该目录下。

当从一个以前的版本进行升级的时候，我建议对以前的 testmaker_home 目录作一个备份。testmaker_home/settings 目录保存了关于打开窗口大小、菜单选择以及其他选项的设置。testmaker_home/agents 目录在默认情况下保存了使用 TestMaker 时创建的代理。这两个目录下的内容要从以前的版本中复制到升级的 TestMaker 版本中。

testmaker_home 目录包含了几个文件和目录，表 5-2 对其进行了描述。

表 5-2 TestMaker 标准文件和目录

文 件 名	目 的
agents	代理示例，用于演示 TestMaker 如何利用实际示例
doc	关于 TestMaker 的文档和额外信息，包括 TestMaker 脚本语言的参考指南以及 Java 开发人员希望改进 TestMaker 的 Java 说明文档
source	建立 TestMaker 的源代码
lib	Java 存档资源(Jar)包中为 TestMaker 和所有支持库所编译的 Java 类
util	在 Windows、Unix、Linux、Mac OS X 命令行下运行 TestMaker 测试代理所需的工具
license.html	为 TestMaker 和与 TestMaker 一起分发的所有捆绑开源项目所提供的完整许可，包括 Apache SOAP、Apache Xerces、Jython、Java Mail API，以及 Java Activation Framework
testmaker.bat	Windows 操作系统的启动脚本
testmaker.sh	Linux 操作系统的启动脚本
readme.html	关于 TestMaker 的最新改动和增添功能的文档

5.4 运行 TestMaker

TestMaker 在 Windows、Linux 和 Mac OS X 系统上运行简单的启动脚本：

- Windows 下的用户运行 testmaker_home/testmaker.bat 文件。该文件启动 Java 并执行 TestMaker 应用程序。
- Unix、Linux 和 Mac OS X 下的用户运行 testmaker_home/testmaker.sh 文件。该文件执行启动 Java 并执行 TestMaker 应用程序的一个简单 shell 脚本。

当 TestMaker 启动时，可以看到描述系统环境的安装信息、classpath 信息、版本号以及其他诊断信息。这些信息在使用 TestMaker 的时候对于确定问题是非常有帮助的。根据计算机系统的速度，TestMaker 环境应该在 30 秒钟之内完成启动。

在安装或者启动 TestMaker 出现问题时，可以查看支持文件或者在 PushToTest 网站 http://www.pushtotest.com 进行咨询以获得帮助。

5.4.1 熟悉 TestMaker 图形环境

TestMaker 的版本 4 是基于 Swing 的 Java 应用程序，它可以为希望使用集成开发环境的软件开发人员、QA 技术员和 IT 管理员提供许多功能。TestMaker 具有一个功能强大的编辑器、输出面板、测试代理执行控制、可视化文件系统导航器、在线帮助，以及一个新代理向导。

图 5-1 说明了 TestMaker 的图形环境和用于创建并运行测试代理的许多元素，其中包括如下元素：

- **菜单条**——下拉菜单为创建新代理、打印代理脚本、查找和替换、设置图形环境的选项提供了命令功能。
- **资源管理器**——资源管理器是查看本地计算机文件系统的浏览器。通过小而旋转箭头的图标可以查看目录中的文件。双击一个文件可以在脚本编辑器中打开该文件。资源浏览器在下次运行 TestMaker 的时候会记住该目录视图。

- **脚本编辑器**——TestMaker 有一个用于编写测试代理脚本的功能强大的编辑器，包括剪切板的剪切、复制和粘贴测试代理脚本的操作，取消和重做变动，语法高亮显示。当一个代理脚本出现错误的时候，编辑器将突出显示包含语法或者其他编程错误的那行代码，或者突出显示发现错误的那行代码。
- **输出**——每个正在运行的代理在输出窗口中会显示进程和统计信息。当执行多个代理的时候，输出窗口使用制表图标系统在代理的输出显示之间进行切换。
- **执行控制**——在资源管理器下面是 Execution 控制面板，它用于运行、暂停和停止测试代理。正在运行的测试代理出现在 Execution 控制面板下面的列表中。最左边的 Run 按钮将启动在脚本编辑窗口中选取的当前测试代理脚本。Pause 和 Stop 按钮将在当前正在运行的测试代理列表中暂停和停止所选的测试代理。

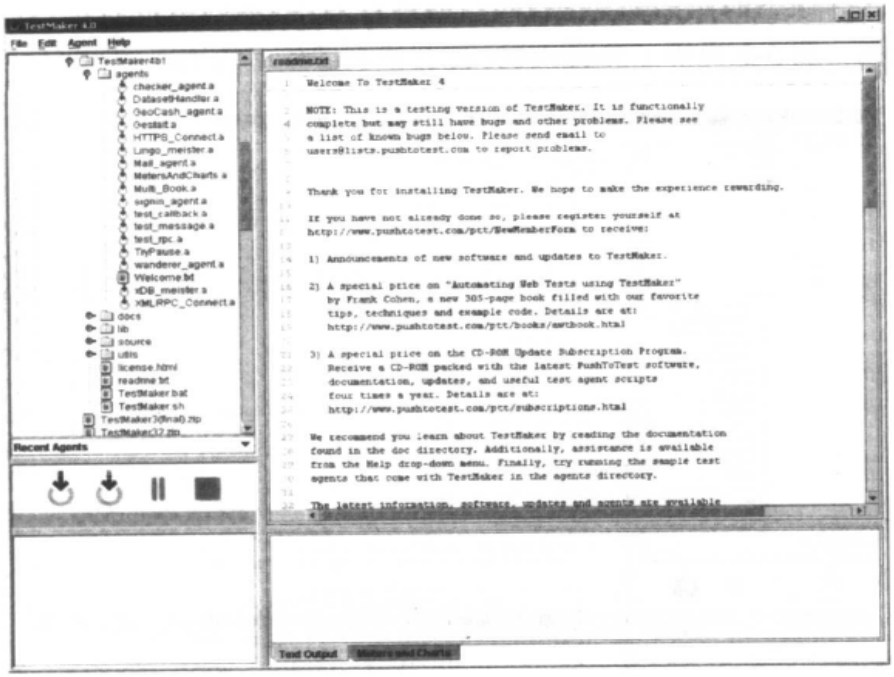

图 5-1　TestMaker 包含了一个集成的编辑器、文件导航资源管理器、输出窗口，以及工作区管理的功能

5.4.2　打开并运行测试代理

在 TestMaker 中建立并运行测试代理很直接且很容易。在本节中，我们将运行一个示例测试代理，然后说明如何使用记录器创建一个新代理。本节中的插图取自于 TestMaker 版本 4 中的窗口。

这个示例用到了在 testmaker_home/agents 目录中的一个测试代理示例。该代理功能完备并与 PushToTest 主机上的 examples.pushtotest.com 网站相连接。

在开始之前，通过运行 testmaker_home 目录下的 TestMaker.bat 或者 TestMaker.sh 脚本启动 TestMaker。TestMaker 图形环境包括编辑代理脚本、浏览代理文件系统、查看输出、控制代理的执行等功能。使用 TestMaker 并运行代理，可以立刻使用测试代理示例中的任何示例。

TestMaker 使用资源管理器打开并且操作代理脚本。图 5-2 说明了 TestMaker 首次运行时的情况，资源管理器表明了最高级的目录。在 Windows 系统中，资源管理器列出了所有的驱动器。在 Linux 系统中，资源管理器将显示根目录。

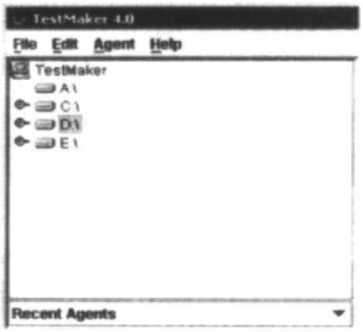

图 5-2 资源管理器面板为查找并打开测试代理脚本提供了方便的文件导航器

通过使用资源管理器导航到 testmaker_home/agents 目录下，打开一个测试代理的示例。单击旋转图标显示该目录下的文件以及子目录。通过双击 HTTP_Connect.a，代理目录可以打开一个测试代理示例。图 5-3 说明了 TestMaker 打开该测试代理的情况。

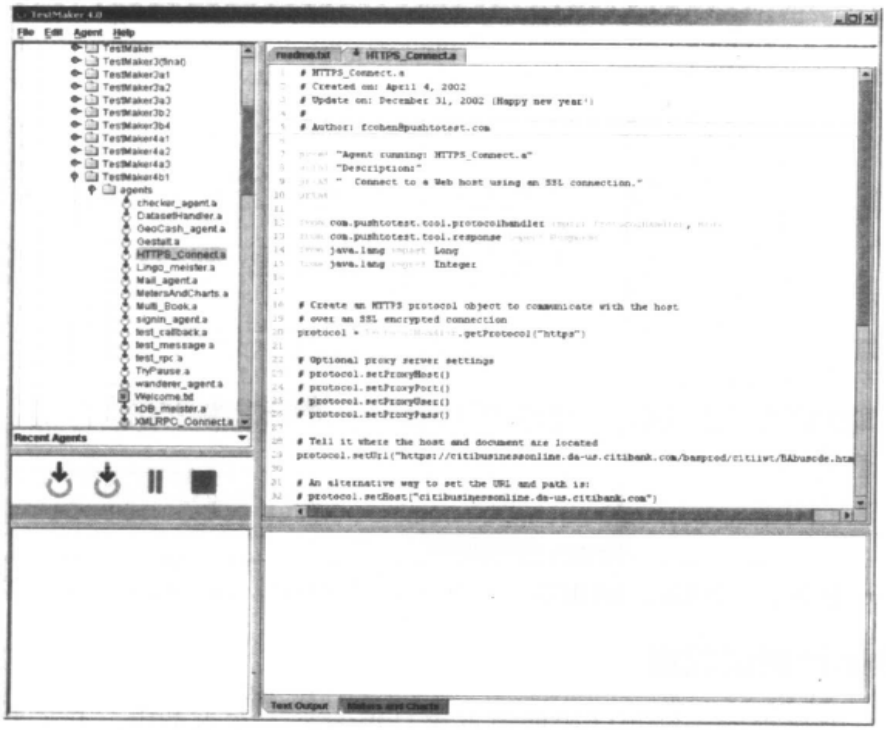

图 5-3 在 TestMaker 中打开了 HTTP_Connect.a 测试代理

HTTP_Connect 是对银行网站安全地发出一个 HTTPS 请求的简单脚本。我们将在稍后说明完成脚本所需的命令。现在，应该认可这些代理的作用以及 TestMaker 图形环境对查看和执行代理提供的功能。

为了运行这个代理，在 TestMaker 窗口左下方的 Execution 面板中单击 Run 按钮。也可以从 Agent 下拉菜单中选择 Run，或者按下 Control-R。如图 5-4 所示。

图 5-4　Execution 面板在 Explorer 面板下面，它控制在本地机器上运行测试代理，在分布的 TestNetwork 节点上运行测试代理，暂停正在运行的测试代理，并可以停止测试代理

TestMaker 通过为代理创建一个新线程从而执行代理。运行的代理会出现在按钮下面的正在运行的代理列表中。代理的输出会出现在脚本编辑器窗口下面的输出面板中。HTTP_Connect 代理的输出应该如下所示：

```
Agent running: HTTPS_Connect.a

Description:
   Connect to a Web host using an SSL connection.

Received this response from the secure host:
<html><head><meta name="Author" content="gm"><meta name="Pro-
gram" content="BAbuscde.html"><meta name="Description" con-
tent="starting frameset  (normal)  "><meta name="Version"
content="1.0"><meta name="Last Revision" content="July 12,
2002"><!--  (gm)  7/12/02 -- added utillS.hs --><title>CitiBusi-
ness Online</title><script language="JavaScript"  src="js/
utillS.js"></script><script language="JavaScript" src="js/
UtilScreen.js"></script><script language="JavaScript" src="js/
referrer.js"></script><script language="JavaScript" src="js/
BAbuscde.js"></script></head><frameset rows="150,*,130" bor-
der="0" frameborder="no"> <frame name="ltframe" src="BAbus-
cdT.htm" noresize scrolling="no" marginheight="2">      <frame
name="lmframe" src="BAbuscdMI.html"><!-- <FRAME NAME="lmframe"
src="BAbuscdMI.html"> --> <!-- PORTAL version --><!-- <FRAME
NAME="lmframe" src="BAbuscdM.htm"> --> <!-- non-PORTAL version
-->   <frame name="lbframe" src="BAbuscdB.htm" noresize scroll-
ing="no"></frameset></html>

Timing values for this request:

Total:  12568
Data :  6840
Setup:  5528

Agent complete.
```

HTTP_Connect 代理接收在使用 SSL 加密的安全 HTTPS 连接上的 HTML 页面——更多的安全信息请参阅第 10 章。输出面板给出了完成请求的全部时间。其值是用毫秒表示的，其中 1000 毫秒等于 1 秒。数据时间是指请求在 Web 服务开始响应数据之前所用的时间。

当然，TestMaker 也可以处理对使用 SOAP 和 XML-RPC 协议的 Web 服务更为复杂的调用。接下来，我们将打开第二个代理，以查看 TestMaker 图形环境同时处理多个代理的情况。

5.4.3 通过新代理向导建立代理

下面，我们将介绍如何通过使用新代理向导创建一个新测试代理的过程。该向导提供了建立测试代理脚本的 4 个方法。每个方法都可以在 File | New Agent 的下拉菜单中找到(图 5-5 所示)。

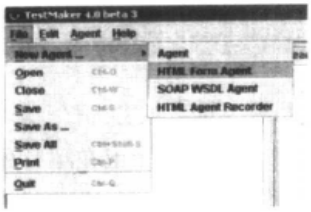

图 5-5 新代理向导为建立测试代理提供了几个有用的工具

新向导可以建立测试 HTML 表单和基于 SOAP 的 Web 服务的代理框架。另外，HTML 代理记录器记录了通过 Web 浏览器对 Web 使能应用程序的操作，并且自动地创建了一个完整的测试代理脚本。首先，看看 HTML 表单的代理命令是如何运行的。然后，使用 HTML 代理记录器记录一个 Web 使能应用程序。

HTML 表单代理命令编写代理框架，以执行一个 HTTP Post 命令，其中在 HTML 表单中发送了一些参数。可以为新 HTML 表单代理提供一个执行 HTML 网页的 URL。新向导检索 HTML 页面并发现任何<form>元素，然后建立 TestMaker 脚本，它将在表单引用的 servlet 中执行 HTTP Post 命令。为了使用 HTML 表单代理，选择 File | New Agent | HTML Form Agent。这样会出现图 5-6 所示的对话框。

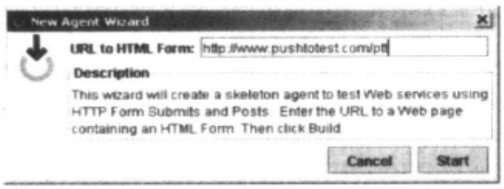

图 5-6 将 URL 输入到带有 HTML 表单的页面中

将 http://examples.pushtotest.com/responder/htmlresponder?file=file2.html 输入到 URL 入口区域中。这是一个特定领域的 PushToTest 主机，它可以提供几个支持 TestMaker 的示例文件和服务，以及一些 TestMaker 的测试代理示例。

接下来，单击 Start 按钮。Editor 窗口在几秒钟内会出现可以在网页中操作 HTML 表单的测试代理脚本。由于可能是第一次接触 Jython 脚本语言，我将在此详细地介绍该脚本语言。首先，给出这个完整的脚本，然后，将解释脚本中每个独立的功能。

```
# Agent name:  Try Examples Form
# Created on: July 4,  2003

from com.pushtotest.tool.protocolhandler import \
ProtocolHandler, Header,  Body, HTTPProtocol,  \
```

```
HTTPBody, HTTPHeader

from com.pushtotest.tool.response import Response,  \
ResponseLinkConfig,  SimpleSearchLink

from java.lang import Exception

# The New Agent Wizard searched the page at
# the given URL for HTML forms.

# First, here is the TestMaker script to
# request the page at the given URL.

http = ProtocolHandler.getProtocol(  "http"  )
http.setType( HTTPProtocol.POST )
http.setUrl(  "http://examples.pushtotest.com/responder/ \
       htmlresponder?file=file2.html"  )
response = http.connect()
print
print response
print

# Form 1

http = ProtocolHandler.getProtocol("http")
http.setUrl( \
  "http://examples.pushtotest.com/responder/htmlresponder")
http.setType(HTTPProtocol. POST)
body = ProtocolHandler.getBody("http")
body.addParameter ("firstname", " "  )
body.addParameter ("lastname",  " "  )
body.addParameter ("phone",  " "  )
body.addParameter ("account",  " "  )
body.addParameter ("amount",  " "  )
body.addParameter ("firstname",  " "  )
http.setBody(body)
response = http.connect()
print
print response
print

# Form 2

http = ProtocolHandler.getProtocol ("http")
http.setUrl( \
  "http://examples.pushtotest.com/responder/htmlresponder")
http.setType(HTTPProtocol.POST)
body = ProtocolHandler.getBody("http")
body.addParameter("firstname",  " "  )
body.addParameter("lastname",  " "  )
body.addParameter("firstname",  " "  )
body.addParameter("Submit",  "Update")
body.addParameter("Submit",  "Delete")
body.addParameter("firstname",  " "  )
body.addParameter("radio",  "value495749")
```

```
body.addParameter("comment",  " "  )
body.addParameter("select",  "null")
http.setBody(body)
response = http.connect()
print
print response
print
```

注意：

本书约定使用\将很长的代码行分成几行。当在 TestMaker 中输入示例的代码时，必须忽略这个字符，而只在一行中输入代码。

在详细查看 Jython 脚本之前，请看看浏览器所指向的网页：

```
http://examples.pushtotest.com/responder/htmlresponder?file
=file2.html
```

在网页上会发现 PushToTest 的标志、一些介绍性的文本以及两个表单。表单 1 显示了名和姓、电话号码、账号以及金额的输入字段。输入字段是简单的文本框。单击 Transfer Funds 按钮可以将 HTTP Post 命令发送到 examples.pushtotest.com 主机上。响应者 servlet 通过显示包括 Post 表单提交数值的页面来回应 Post 命令。

这个页面还包含一个表单。表单 2 显示了名和姓的输入字段、一个用于注释的文本区域、一个用于继续(follow-up)的单选按钮组和一个用于选择颜色的多选框列表。单击 New、Update 和 Delete 按钮将 HTTP Post 命令提交到 examples.pushtotest.com 主机上。响应者 servlet 通过显示包括 Post 表单提交数值的页面来回应 Post 命令。

现在，我们来详细地看看测试脚本，理解 Jython 和 TestMaker TOOL 是如何协同工作以实现一个简单测试的。

```
from com.pushtotest.tool.protocolhandler import  \
   ProtocolHandler, Header, Body, HTTPProtocol,  \
   HTTPBody, HTTPHeader
```

这条命令以及其后的两条命令确定了 Jython 的 Java 对象，该脚本将使用这些对象与 examples.pushtotest.com 主机进行通信。由于该脚本使用 HTTP 协议与主机进行通信，因此脚本导入了 HTTPProtocol 对象和其他几个对象。HTTPProtocol 对象为向主机发出 GET 和 POST 请求提供了一些必要的方法。

为了方便起见，新代理向导创建获得页面的脚本命令。

```
http = ProtocolHandler.getProtocol(  "http"  )
```

这条命令使用了 ProtocolHandler 对象，它返回一个新的 HTTPProtocol 对象，脚本稍后将使用 http 变量引用这个对象。

```
http.setUrl( \
  "http://examples.pushtotest.com/responder/htmlresponder \
  ?file=file2.html" )
```

http 变量的第一次使用是用于访问 HTTPProtocol 对象的 setUrl 方法。这表明在连接主机的时候，HTTPProtocol 对象所使用的 URL。

```
response = http.connect()
```

connect 方法指明了 HTTPProtocol 对象将一个 GET 请求提交给主机。它将返回一个新的 HTTPResponse 对象。response 变量现在指向了 HTTPResponse 对象，而且可以访问许多处理 response 数据的方法。例如，response.getTotalTime()返回连接主机所用的全部时间，发送请求并且得到返回的所有数据。参见 testmaker_home/doces 目录或者访问 http://docs.pushtotest.com 网站可以获得 TOOL 对象和方法的完整列表。

```
print
print response
print
```

这些命令将 HTTPResponse 对象的内容打印到输出窗口中。
新代理向导在给定的 HTML 表单的 URL 中查找该页面。下面的代码处理第一个表单。

```
http = ProtocolHandler.getProtocol("http")
http.setUrl( \
  "http://examples.pushtotest.com/responder/htmlresponder")
```

在脚本中，已经看到了用于 GET 请求的两个命令。因此，我就不再解释它们了。

```
http.setType(HTTPProtocol.POST)
```

在默认情况下，HTTPProtocol 对象将 HTTP Get 命令提交给主机。因为这是一个表单，我们使用 setType 方法对它进行设置，以向主机发出 HTTP Post 请求。

```
body = ProtocolHandler.getBody("http")
```

HTTP Post 命令通知主机立即查找在 HTTP 头部之后的数据内容(payload)。TOOL 使用 HTTPBody 对象设置 Post 命令的数据内容。这条命令创建一个新的 HTTPBody 对象，用 body 变量指向该对象。

```
body.addParameter("firstname",  " "  )
body.addParameter("lastname"-,  " "  )
body.addParameter("phone",  " "  )
body.addParameter("account",  " "  )
body.addParameter("amount",  " "  )
body.addParameter("firstname",  " "  )
```

这些命令将在 HTTP 请求期间发送给主机的参数告知 HTTPBody 对象。在给定的 URL 的 HTML 页面的表单中，每个名/值对都有一条命令。

```
http.setBody(body)
```

setBody 方法确定了 HTTPBody 对象是 HTTPProtocol 对象的一部分。当 HTTPProtocol 对象与主机相连的时候，它将使用下面定义的 HTTPBody 对象。

```
response = http.connect()
```

connect 方法通知 HTTPProtocol 对象创建请求，包括 HTTPBody 参数并且向主机发送一个 Post 请求。Connect 方法返回一个新的 HTTPResponse 对象，该对象将主机的响应封装在里面。

```
print
print response
print
```

这些命令将 HTTPResponse 对象的内容打印到输出窗口中。

剩余的脚本重复了 Jython 和 TOOL 命令，向第二个表单发送请求，因此我就不在这里解释了。第 6 章为 Web 主机提供了更多使用 HTTPProtocol 对象的示例。

在第一个示例中，介绍了 TestMaker 的新 HTML 代理向导是如何在网页中发现这两个 HTML 表单的，以及如何使用 TOOL 命令自动编写 Jython 脚本以驱动使用内部 HTTP 协议的 Web 主机。为了运行这个脚本，只需在 Execution 面板中单击 Run 按钮，或者在 Agent 的下拉菜单中选择 Run。其结果就会出现在输出面板中。

上面的这些内容可能使您产生了一些兴趣，下面将介绍 TestMaker、Jython 和 TOOL 的实际功能。本章的下面一节将说明如何使用记录器创建一个测试代理，该代理会在下一个示例中给出与之相连的 Responder 主机的吞吐量。

记录测试

代理记录器通过浏览器查看对 Web 使能应用程序的操作并且编写测试代理脚本。产生的测试脚本功能完善而且进行事务/秒的测试。该脚本也可以插入到 XSTest 框架中——将在第 6 章讨论——可以完成可扩展性、性能和监测测试。

记录器是根据智能代理服务建立的，它监测浏览器与服务器之间的 HTTP 流量。代理对来自浏览器的 HTTP Get 和 Post 命令以及来自服务器的响应解码。然后，代理编写重新操作 Web 使能应用程序所需要的 Jython 和 TOOL 的脚本命令。这种设计支持 HTTP 1.0 和 1.1 兼容的浏览器，包括 JavaScript、plug-in、ActiveX 对象和 Java applet 的使用。

代理记录器并不支持 HTTPS 连接。在 TestMaker 代理接收来自浏览器的请求时，请求是加密的，TestMaker 不能解码 HTTP 协议的主体部分。我目前知道的惟一工作办法是在记录测试的时候临时关闭 SSL 加密应用程序。

在使用记录器之前，需要将浏览器配置为通过记录器代理进行通信。在默认情况下，TestMaker 将代理端口设置为 8090。浏览器选项对话框通常是设置代理端口的地方。每个浏览器在控制代理服务器上都有所不同。例如，在 Windows 2000 的 Microsoft IE 6.0 中，在 Tool 下拉菜单的 InternetOption 中可以找到代理服务器的设置，如图 5-7 所示。选择 Connections 选项卡，然后单击 LAN Settings...按钮。所出现的对话框的下面部分用于控制代理服务器的设置。

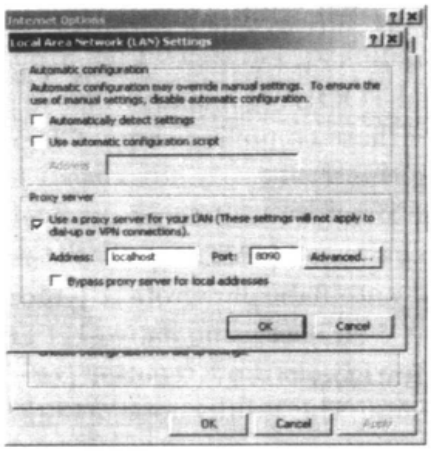

图 5-7　在 Windows 2000 的 Microsoft IE 6.0 中配置记录器代理服务器的设置

如果系统中用到了 8090 端口，那么要为 TestMaker 代理服务器选择另一个端口号。在 Help 下拉菜单中选择 Preferences 命令，然后选择 ReCorder 选项卡。

通过对代理的配置，浏览器中的每个请求都会流经 TestMaker 代理服务器。这带来了一个副作用，TestMaker 需要在浏览器中运行。

接下来，我们将在 examples.pushtotest.com 主机上对 Web 使能应用程序的测试进行记录。这个应用程序是一个简单的响应者 servlet，它提供随机内容的页面，对 HTML 表单中的 HTTP Post 命令作出响应并且提供测试所需的带有 HTML 链接的网站。

首先打开 TestMaker，配置浏览器，然后选择 File | New | HTML Agent Recorder 菜单选项。TestMaker 会显示一个询问新测试代理名称的对话框，如图 5-8 所示。输入 MyFirstTest.a 作为测试代理的名称。

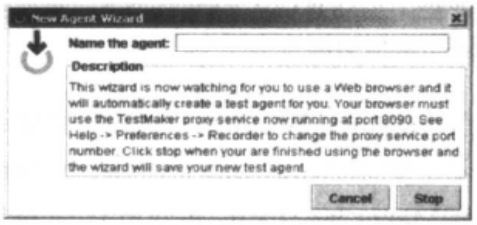

图 5-8　新代理向导要求用户输入测试代理的名称。然后，记录器查看所用的浏览器并且自动编写测试代理脚本

在 examples.pushtotest.com 网站上，我们想要记录的测试使用 3 个页面，如图 5-9 所示。首先打开浏览器，进入这个 URL：http://examples.pushtotest.com/responder/htmlresponder。

通过单击 file2.html 链接继续记录该测试，这个链接是网页的一部分。浏览器将显示一个包含两个表单的新网页。在第一个表单中，输入姓名以及电话号码。然后，输入 1081 作为账号，在金额字段中填入 75.36。最后，单击 Transfer Funds 按钮。浏览器将显示响应在前面网页中已提交的 HTML 表单信息的一个网页。

回到 TestMaker 并单击 Stop 按钮。TestMaker 将显示一个标准的文件选择对话框。选择想要保存这个新代理文件的目录，然后为该代理输入文件名，单击 Save 按钮。新测试代理脚本将会在 TestMaker 的 Editor 面板中出现。为了运行这个新脚本，在 Execution 面板中单击 Run 按钮。另外，也可以在 Agent 下拉菜单中选择 Run 命令。当使用浏览器驱动在 examples.pushtotest.com

网站上的 Web 使能应用程序时，这个代理将重复以前的操作。然而，它并不是仅仅重复一次以前的操作，而是重复 50 次。然后，这个测试代理将报告该 Web 使能应用程序的平均事务/秒的比率。

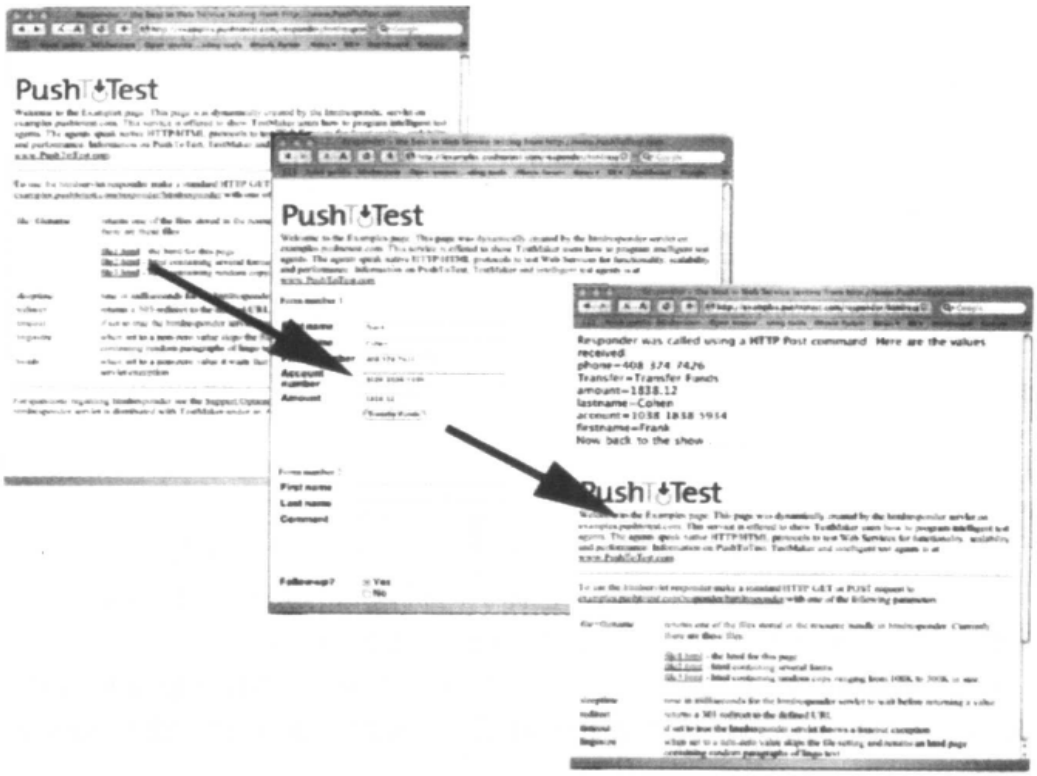

图 5-9 Web 使能应用程序显示了带有 HTML 链接的网页，然后是一个 HTML 表单，最后动态创建我们提交表单数值的网页

记录器使得 TestMaker 嵌入的 Jython 脚本语言更加有效地重用已记录的对可扩展性、性能和监测测试情景进行的功能测试。因为 TestMaker 嵌入的 Jython 脚本语言是完全面向对象的语言，所以它可以通过 Java 和 TestMaker 的 TOOL 库建立任何对象，并且还能为测试代理中的协议处理提供一个统一且可扩展的 API。我真的非常喜欢 Jython！

5.5 喜欢 Jython 的原因

几年来，Python 脚本语言成为了软件开发人员和 QA 技术员使用的流行语言，它可以为执行测试和指导维护快速地编写一次性脚本。Jython 是完全用 Java 实现的 Python 语言。虽然对于一些计算机科学的学生来说，这可能是一个很好的理论练习作业，但 Jython 代表了向 Java 平台的一次跨越。Jython 不仅实现了 Python，还能让 Python 脚本使用任何 Java 对象。例如，如果您对 Python 中的随机数产生器对象并不是很满意， Jython 可以让您使用 Java 的随机数产生器。这种灵活性是 TestMaker 的基础，因而可以建立实现用户原型行为和通过应用程序的内部协议驱动 Web 使能应用程序的测试代理。

我甚至连 Jython 测试设置之外的东西都非常喜欢。下面列出了一些我喜欢 Jython 的原因。

5.5.1 Jython 十分快速

通过几行 Jython 代码，我编写了一个完整的程序。例如，如下是我的第一个“Hello World”应用程序。

```
import javax.swing as swing
hello =swing. JFrame("Welcome to Jython")
hello.size =  (200,  200)
hello.show()
```

我把该代码段输入到 TestMaker 的 Editor 面板中，然后单击 Run 按钮。Jython 创建了一个 java.swing.JFrame 对象，设置字符串的数值并在屏幕上显示出来。使用 hello 变量，我可以访问任何 JFrame 方法。如果想要查看这些方法的列表，也可以使用 Jython：

```
print dir( JFrame )
```

运行这行脚本就可以将 JFrame 对象的方法打印到 TestMaker 的输出窗口中。Jython 并不要求编译代码、JAR 包、classpath 配置或者其他典型 Java 操作的设置。当我编写原型和调试测试脚本的时候，我发现 Jython 的快速性是非常有用的。

5.5.2 可以免费获得数据结构和列表迭代

Jython 内置了功能强大的数据结构。另外，可以在 Jython 脚本内直接使用 Java 的集合类。例如，假定我有一个需要保存并且以后需要检索的 URL 列表。

```
urls = {  "pageone"  :  "http://images.pushtotest.com",  \
"pagetwo"  :  "http://examples.pushtotest.com",  \
"pagethree"  :  "http://www.jython.org"  }
```

上面定义了保存 3 个名/URL 对的 Jython 目录对象。如果需要打印这个目录的列表，使用如下命令：

```
for i in urls.keys()
  print "The key of", i, "found the value of",urls[i]
```

Jython 的 for 命令将在 URL 目录里迭代所发现的所有关键字，能够使用 Java 的 HashMap 对象完成相同的工作，但是现在我有一个选择：使用 Jython 的目录并列出对象，或者使用 Java 的集合类。我结合使用这两种方法(随便说一下，Sun 正在为 Java 的下一个版本作出改进，其中的循环合并了简单的迭代创建)。

5.5.3 自动选择类型的动态变量

在 Java 中，在使用变量之前，需要创建所有的变量以及它们的类型。在 Jython 中，变量和它们的类型可以在任何时刻动态地创建。

```
def myfunction(x,y);
    z = x*y
    return z

a = 5
b = 7
print "Result: ",myfunction(a,b)
```

在 TestMaker 中运行这个脚本，在输出窗口中将看到：

```
Result:35
```

Jython 动态地创建了局部变量(z，x，y)以及全局变量(a，b)。Jython 可以分配变量的类型。这并不是说 Jython 是一种无类型的语言。Jython 实际上是强类型的。例如：

```
a = 5 + "1"
```

这条命令会抛出一个异常，因为 Jython 将其解释为“整数 5 加上字符串值‘1’”，这在 Jython 语法中是错误的。

5.5.4 函数是一流的

Jython 的函数、方法、模块和类都是一流的对象。它们能够传递和从函数调用中返回。虽然 Java 可以使用匿名的内部类或者反射(reflection)完成相同的工作，但在 Jython 中引用其他对象的函数调用是更加方便的方法。

```
def scramable( a ):
    return a = a + "ality"

def handleit( mystring, myFunction):
    print

in = "function"
printf "Result: ", handleit ( in )
```

在 TestMaker 中运行该脚本，在输出窗口中会看到如下内容：

```
Result: functionality
```

Jython 的一流函数、方法和类使得编写简洁的面向对象代码变得更加容易。

5.5.5 Java 集成

我所喜欢 Jython 的上面 4 点是标准 Python 语言也具备的功能。Jython 进一步对现有的 Java 代码提供了方便的访问方法。Jython 可以像使用其模块一样导入 Java 包。在使用 Jython 创建对象时，解释器会自动地创建一流的 Java 对象。这些对象可用于 Jython 脚本，也可以直接通过 Java 对象访问。

Jython 类甚至可以继承 Java 类。典型的 Java 继承规则是 Java 开发人员所希望的。Jython 对象可以覆盖父类的方法，也可以重载父类的方法。无论哪种情况，Jython 会根据在运行时调用的参数来选取正确的方法。

5.5.6 Bean 属性内省

Jython 完全支持 Java 对象的内省(introspection)。Jython 自动地从 Java Bean 中调用 Java 对象的 getAttribute()方法对方法进行内省。对于需要测试企业 Bean 和容器管理持久性的 J2EE 开发人员来说，这是非常有用的。Jython 进行内省从而查找企业 Bean 中的 getter 和 setter 方法，这样可以确定一个智能测试代理需要采取哪些步骤来测试企业 Bean 的功能。嵌入到 XSTest 中的相同的脚本表明了企业 Bean 是如何处理多个并发请求负载的。

5.5.7 Sun 公司正在采用 Java 中的脚本

在 Java 标准化组织(JCP)的当前工作中，将要为脚本语言添加在 Java 平台上的标准支持。JCP 是看似开放标准的主体，它在成为 Java 自身的新功能上运行。JCP 在 Java 说明书请求(JSR)中确定 Java 所需的功能。JSR 223 描述了如何编写可移植的 Java 类，这些类可以从用脚本语言编写的网页中进行调用，包括安全性、资源和类加载器上下文中的细节。详细资料请访问 http://www.jcp.org/en/jsr/detail?id=223。以我个人的观点，JSR 223 为 Java 添加脚本语言的支持并不是一种好方法。可以在 http://www.pushtotest.com/ptt/wiki/JavaScripting 上阅读我的观点。然而，这些意味着 Jython 的使用周期更长，更适合大的 Java 组织。这使得 Jython 更加优秀。

我希望上述 7 个喜欢 Jython 的原因可以鼓励您使用 Jython，并用它来开发 TestMaker 之外的项目。Jython 可以通过 Jython.com 进行下载。

5.6 使用 Jython 并入 Junit

JUnit 是建立 Java 类单元测试的一个框架。由于它的简单设计和强大的对象使用能力，使得它在软件开发人员和 QA 技术员之间广为流行。另外，它非常适合敏捷(也称可扩展和 XP)编程技术中“测试第一位”的方法学。因为 JUnit 是用 Java 编写的，所以它可以作为 Jython 集成测试工具和框架的一个很好示例。在本节中，我将介绍 JUnit 并且说明如何在 TestMaker 框架中使用 JUnit 和其他 Java 测试工具。

5.6.1 JUnit 用于可重复测试

敏捷编程的规则之一是“测试第一位”。按照字面上的意思，代码的一个片断只有在验证其功能、界面以及实现与设想情况一致的时候，才能够加入源代码树中。因为复杂的互操作信息系统是由大量的软件组件和模块组成的，因此这个规则是非常重要的。如果没有测试，某个模块的改动可能会影响到另一个模块。JUnit 为建立测试实现了一个框架，它会在系统每次发生变化时自动地运行这些代码。

在 JUnit 之前，有几个工具试图为测试软件模块开发出可理解的且广泛使用的框架。Kent Beck 在 Smalltalk 中建立单元测试框架的早期工作引导了使用 Java(http://www.junit.org)、Python (http://pyunit.sourceforge.net/pyunit.html)和.NET(ftp://sourceforge.net/projects/nunit/)建立单元测试框架的先例。有关 Kent 的初始目的的详细情况请访问 http://www.xprogramming.com/testfram.htm。

单元测试框架有着一系列明确的目标：

1. **使用更加方便**。软件开发人员除了他们其他的职责外还要编写单元测试。因此，单元测试框架需要提供熟悉的工具，这要求用最少的工作编写单元测试。

2. **使测试持续更久的时间**。对一个新开发的软件进行测试可以说明软件在这个时刻的运行情况。但是，它无法保证同一个模块明天、下个星期、或者以后五年还这样运行。单元测试需要不断地持续进行。除了起初的开发人员，其他人更有可能在以后使用这个单元测试。因此，单元测试框架必须保证单元测试在以后也有效。

3. **容易地使用现有的测试**。编写新的测试会消耗时间和精力。单元测试框架必须容易地在新测试中重用已有的单元测试。

JUnit 可以在软件开发生命周期的任何时刻运行，但很多时候通过 Java 编译器运行 JUnit 对一个新编译的应用程序进行单元测试。当与流行的 Ant(http://ant.apache.org/)编译工具相结合时，以 JUnit 框架建立的单元测试在编译之后会自动地被 Ant 编译脚本调用。在这种情况下，单元测试是自动的、透明的。在软件开发人员的心目中，单元测试仅仅是编译器操作的一部分。编译器检测源代码的错误，单元测试然后运行查找错误和纠正操作的编译代码。

5.6.2 一个 JUnit 示例

JUnit 通过实现 TestCase 这个基本概念从而完成上述这些目标。它在建立测试的时候匹配大多数软件开发人员日常所做的工作。TestCase 对象实现了 3 个方法：setUp 方法用于测试之前对系统进行配置，runTest 方法用于运行测试，tearDown 方法执行所需要的任何清除任务。TestCase 实现这些方法从而可以运行软件模块并且检查对有效数据的响应。最后，TestCase 实现了处理异常的方法，异常是正被测试的对象抛出的，TestCase 还实现了对测试结果的记录，以便用于后面的分析。

为了结合上下文，考虑一个带有两个参数的对象，把这两个数相乘并且返回结果。

```
public class Utilities
{
      public int multi(  int valueone,  int valuetwo )
      {
            return valueone * valuetwo;
      }
}
```

检验 Utilities 类中的 multi 方法的 JUnit 测试如下：

```
public class TestUtilites extends TestCase
{
      public void testMulti()
      {
            int valueone = 24;
            int valuetwo = 2;
            int expectedresult = 48;

            Utilities util = new Utilities();

            int result = util.multi( valueone, valuetwo );

            assertTrue( expectedresult == result );

      }
}
```

一旦定义测试，就可以通过创建实例和调用 testMulti 方法来运行它。

```
TestCase test = new TestUtilities  ("testMulti");
test. run();
```

如果测试运行的所有目标都能够达到，testMulti 方法就会返回正确的结果。如果没有返回预期的结果，Junit 就会抛出一个异常并且表明测试失败了。

JUnit 框架提供了一组测试运行器(runner)，它们可以运行一组测试并且收集结果。下面是正在运行两个测试的示例：

```
Public static Test suite()
{
      suite.addTest(new TestUtilities("testMulti"));
      suite.addTest(new TestUtilities("testAdd"));
      return suite;
}
```

5.6.3 JUnit 和 TestMaker

TestMaker 中的 Jython 脚本语言可以运行任何 Java 对象，包括 JUnit。在 TestMaker 中运行 JUnit 意味着可以用 JUnit 框架测试 Web 使能应用程序、EJB 以及 Web 服务。为了将 Junit 添加到 TestMaker 中，需要执行下列步骤：

(1) 将 JUnit 的 Java 归档资源(JAR)文件添加到 TestMaker 的 classpath 中。在 Unix 和 Mac OSX 系统中可以通过编辑 TestMaker_home/TestMaker.sh 文件，在 Windows 系统中可以通过编辑 TestMaker_home/TestMaker.bat 文件来完成。例如，对于 Windows 操作系统，添加如下命令：

```
set LOCALCP=%LOCALCP%;%tmlib%\Junit.jar
```

(2) 将一个 import 语句添加到 TestMaker 测试脚本中，以便识别 Junit 类。例如：

```
from junit.textui import TestRunner
```

(3) 在测试脚本中使用 JUnit 类和方法。

作为一个在 TestMaker 测试代理脚本中使用 JUnit 的实例，可以设想一个将金额从一个账户转移到另外一个账户的 Web 使能应用程序的示例。这个示例使用了 TestMaker 的 HTTP 协议处理程序 POST 转移细节，使用了 JUnit 验证 HTTP 的响应值。下面是测试脚本的完整代码，而且给出了详细的注释。这个示例的代码以及本书其余的代码可以通过 http://www.pushtotest.com 立即下载。

```
from junit.framework import TestCase
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  Body, HTTPProtocol, HTTPBody
from com.pushtotest.tool.response import Response
class TestTransfer( TestCase ):
      def runTest(self):
            self.http = ProtocolHandler.getProtocol("http")
            self.http.setUrl( \
```

```
        '''http://examples.pushtotest.com/ \
        responder/htmlresponder'''  )
    self.http.setType( HTTPProtocol. POST)

    self.body = ProtocolHandler.getBody(  'http'  )
    self.http.setBody( self.body )
    self.body.addParameter('firstname',  'Frank')
    self.body.addParameter('lastname',  'Cohen')
    self.body.addParameter('phone',  '374 7426')
    self.body.addParameter('account',  '10382')
    self.body.addParameter('amount',  '300.00')
    self.body.addParameter('Transfer', 'Transfer Funds')
    self.response = self.http.connect( 0 )
    self.code = self.response.getResponseCode()

    self.assertEquals(220,  \
      self.response.getResponseCode()  )

mytest = TestTransfer(  "test_transfer"  )
mytest.runTest()
print "done"
```

脚本的关键之处在于它一直使用了 Jython 环境。在这个测试脚本中，我们使用了 Jython 对象、TestMaker 的 TOOL 协议处理程序对象以及 JUnit 测试对象。下面是测试脚本在深度方面的一个解释：

```
from junit.framework import TestCase
```

Import 命令确定了测试代理脚本要用到的 JUnit 对象。回忆一下，我们在 TestMaker_home/TestMaker.bat 脚本中为了包含 junit.jar 包修改了 TestMaker 的 classpath。

```
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,Body,HTTPProtocol,HTTPBody

from com.pushtotest.tool.response import Response
```

这些 Import 命令确定了测试代理脚本会用于与主机通信并接收响应的 TOOL 协议处理程序对象。

```
Class TestTransfer( TestCase ):
```

这个测试确定了一个标题为 TestTransfer 的新对象，该对象从 JUnit 的 TestCase 类中继承了其方法。

```
  def runTest(self):
```

TestTransfer 对象只有一个 runTest 方法，该方法执行实际的测试。它向主机发出一个 HTTP Post 请求并且接收响应。

```
    self.http = ProtocolHandler.getProtocol("http")
```

这个脚本创建了一个新的与主机通信的 HTTPProtocol 处理程序对象。http 变量将是对该协议处理程序的脚本的引用。

```
self.http.setUrl(\
'''http://examples.pushtotest.com/ \
responder/htmlresponder''')
self.http.setType( HTTPProtocol.POST )
```

这段脚本确定了主机的 URL 和协议处理程序的 POST 请求。现在，这个协议处理程序需要转移函数的参数，然后就可以向主机发送请求了。

```
self.body = ProtocolHandler.getBody( 'http' )
self.http.setBody( self.body )
self.body.addParameter( 'firstname', 'Frank')
self.body.addParameter( 'lastname' , 'Cohen')
self.body.addParameter( 'phone', '374 7426')
self.body.addParameter( 'account', '10382')
self.body.addParameter( 'amount', '300.00')
self.body.addParameter( 'Transfer', 'Transfer Funds')
```

这段脚本创建了一个新的 HTTPBody 对象，而且添加了执行资金转移函数所需的参数。在多数情况下，您可能更喜欢用 TestMaker 的记录器来创建这个脚本。记录器通过查看浏览器自动地确定参数。详见本章的记录器一节。

```
self.response = self.http.connect( 0 )
```

这个协议处理程序可以向主机发送请求了。connect 方法让这个协议处理程序发出请求并且返回一个新的 response 对象。response 对象返回响应码，而且为响应数据的处理提供了几个方便的方法。

```
self.code = self.response.getResponseCode()

self.assertEquals(200, \
  self.response.getResponseCode() )
```

通过响应码，这个脚本使用 JUnit 的 assertEquals 反复检查 HTTP 响应码是否为 200。如果响应码为 200，那么该脚本将继续执行，如果响应码为其他值，Junit 就会抛出下面的异常：

```
junit.framework.AssertionFailedError:
expected:<200> but was:<500>
```

在这个示例中，说明了 Jython 很容易与其他基于 Java 的测试框架相互协作，这里也包括了 JUnit。另外，JUnit 和 TOOL 的协议处理程序的结合也让 JUnit 可以与 Web 使能应用程序交互操作，JUnit 本身并没有提供特殊的功能。

5.7 本章小结

本章列出了用于建立智能测试代理的测试工具和库的工具箱。在第 6.2 节的示例中包括了 TestMaker 代理以实现真正的测试，从而可以了解现有的技术。虽然本章说明了如何安装并使用 TestMaker，包括 TestMaker 脚本语言的指南以及一些测试代理的示例，但是对于 TestMaker 的详细说明将在下面几章介绍，关于额外的信息和帮助请到 PushToTest 网站 http://www.pushtotest.com 上查阅。

第 6 章　HTTP/HTML 环境中的设计与测试

在 20 世纪 90 年代末，术语“Web”进入了全球公共的词典，它就像可乐、出租车、信用卡一样得到了全球化。目前，全世界的人一旦知道有什么资源“在 Web 中”，他们就可以坐在个人电脑前通过浏览器得到它。在浏览器的背后是大量的软件协议、路由网络连接和信息系统。那些在网上建立、测试以及维护资源的人员需要了解 Web 技术，以及测试资源是否真的位于 Web 中的策略和工具集。

将东西放在 Web 中，需要组合一些技术。虽然 HTTP 是 Web 的协议，但超文本链接标记语言(HTML)和 XML 是具有一定格式的语言。像 JSP、Servlet、Microsoft ActiveX、VB、Pearl、Python、Ruby、Macromedia Flash、GIF 图像和 JavaScript 这样的技术都为软件开发人员提供用户个性化和喜好的 Web 资源起到了一定的作用。这些系统在服务器创建网页启动资源的时候提供了类似桌面应用程序的功能。本章将介绍如何在 HTTP/HTML 环境中高效地测试 Web 使能应用程序。

本章我们所测试的应用程序假定了这样的情况，浏览器要很“小”而服务器要很“大”——也就是说，浏览器仅仅显示服务器发送的内容，而服务器提供业务逻辑、计算能力、与其他系统的互操作性以及存储能力。后面的章节将列出新的体系结构和测试方法，这些技术会在浏览器和服务器之间作出权衡。

首先，本章说明了基础内容。我们开发 HTTP/HTML 环境，包括像 XML 和 XHTML 的额外内容语言。然后，解释了如何处理 HTML 表单和 cookie。之后，列出了几个用于验证 Web 使能应用程序响应的策略，其中包括了检验一个使用 XML 编码的响应。

6.1　HTTP/HTML 环境

我们来开发一个银行服务，用户可以使用这个服务将资金从一个账户转移到另外一个账户。目前，多数银行都提供 Web 接口。银行通常只通过 Web 浏览器界面为用户提供很小一部分功能的子集。在这种情况下，浏览器通常通过与大型机的 HTTP 连接来接收 HTML 代码。大型机提供用于识别(或者登录)用户的业务逻辑，并提供将资金从一个账户转移到另一个账户的功能。大型机上的代码可能是用 COBOL 或者其他过时的编程语言编写的。因此，毫无疑问，这种体系结构很像老式的客户/服务器系统——服务器发出命令，浏览器接收命令。

这并不是说银行的 Web 开发人员无法设计并建立良好的 Web 应用程序。他们可以而且可以经常进行优秀的设计；然而，底层的处理是一成不变的。在基本的 HTTP/HTML 体系结构中，浏览器执行下面 4 项任务：

1. 将 HTML 代码显示在屏幕上。

2. 向服务器提交 GET 请求。
3. 向服务器提交表单 POST 请求。
4. 在浏览器与服务器之间对通信进行加密和解密。

图 6-1 说明了一个客户将资金从一个银行账户转移到另一个账户的过程。首先，客户通过浏览器页面连接到银行的登录页面。登录页面有一个简单的表单，客户需要填写客户的身份和密码。客户单击登录按钮，浏览器会显示转移资金的页面。转移资金的页面显示了一个简单的表单，其中有源账户号码、转移的金额、目标账户的号码。客户输入这些值，然后单击转移按钮。这样客户会在浏览器中看到确认页面。

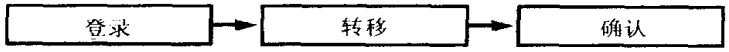

图 6-1　客户在转移资金时，会在浏览器中看到 3 个页面。然而，在这种情况下，却发生了 6 个 HTTP 事务

虽然客户在事务处理的过程中只看到了 3 个页面，但却发生了 6 个 HTTP 事务。在这种情况下，浏览器和服务器都使用 HTTP 协议发送指令和/或者 HTML 代码。浏览器不但获得了显示 HTML 代码的 HTTP 命令，还接收了向后续服务器发出调用的重定向命令。在所有这些情况中，浏览器发出请求并且从服务器接收响应。

在图 6-2 中，观察在 HTTP 协议级别下浏览器和服务器之间的通信，这样可以更好地理解资金转移事务。浏览器发出请求，服务器进行响应。无论看起来网页有多么复杂，即使在网页具有像 GIF 图像、Flash 动画、JavaScript 滚动效果的特殊表示形式的时候，浏览器与服务器之间的每次通信也都是一样的。

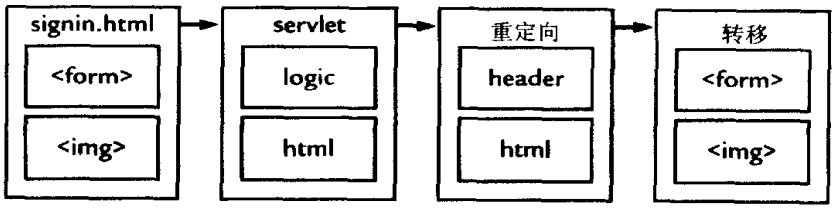

图 6-2　在转移资金事务的步骤中，浏览器发送一系列的 GET 和 POST 命令，服务器每次用一个新网页对其作出响应

图 6-2 说明了每个发送给服务器的 GET 和 POST 命令。每个命令接收一个响应，包括响应代码。例如，标准的响应代码是 200，这表明成功地处理了命令而且没有错误发生。表 6-1 描述了转移资金事务的步骤。

表 6-1　处理银行资金转移的 GET 和 POST 命令序列

步骤	浏览器向服务器发送命令	发送给服务器的参数	服务器对浏览器作出响应的 HTTP 头部	服务器对浏览器作出响应的 HTML
1	GET	signin.html	200，正常	包含登录表单的 HTML
2	POST	handle_signin servlet	302，重定向到 transfer.html 网页，同时发送确定用户和会话的 cookie	包含链接 transfer.html 网页的 HTML
3	GET	transfer.html	200，正常	包含转移资金表单的 HTML

(续表)

步骤	浏览器向服务器发送命令	发送给服务器的参数	服务器对浏览器作出响应的HTTP 头部	服务器对浏览器作出响应的HTML
4	POST	transfer_funds servlet	302，重定向到confirmation.html 网页	包含链接 confirmation.html 网页的 HTML
5	GET	confirmation.html	200，正常	包含确认页面的 HTML

浏览器向服务器发出 GET 和 POST 命令并且接收响应。每个 GET 至少要说明目的服务器和所需的网页、文档或者 servlet。通常情况下，GET 也发送可选的 HTTP 参数和有助于服务器更好地理解请求的 HTTP 头部。例如，一个 GET 命令可能也包含一个头部参数，它在称为 cookie 的参数中传递当前服务器的会话信息。GET 请求中其他常见的内容包括名为 Accept 的 HTTP 参数，它为服务器提供一个浏览器可以显示的媒体类型的列表。例如，Accept:text/html，flash/image 告诉服务器，浏览器可以显示包含 Flash 动画的 HTML 网页。

除了转移数据，浏览器的 POST 命令看起来很像 GET 命令。GET 请求在 HTTP 头部向服务器发送额外的 HTTP 参数列表。在上面的第 2 步，浏览器给 handle_signin servlet 发送一个 POST 请求并且传送包含用户身份和密码的表单数据。POST 请求看起来像是 GET 请求，然而，用户的身份和密码被发送到 HTTP 体中，该 HTTP 体是在请求头部之后的一整块数据。这可以使 POST 请求包含无限量的数据。

6.2 HTTP/HTML 环境中的常见错误

HTTP/HTML 体系结构是如此的简单！为什么还需要对它们进行测试呢？是什么经常失败或者出错？证明的结论是很多东西能够并且经常会出现错误。下面是 HTTP/HTML 环境中最常见的错误：

- 浏览器与标准不兼容
- 无效的数据
- 会话问题

下面从软件开发人员、QA 分析员和 IT 管理员在 HTTP/HTML 环境中遇到的最严重的问题开始讲起。

6.2.1 兼容性和浏览器缓存

Web 流行的一种重要原因是不存在一种控制 HTTP 和 HTML 的标准体。像万维网协会(http://www.w3c.org)和 Internet 工程任务组(http://www.ietf.org)这样的标准化组织都试图规定标准，并在许多组中协调工作。然而，这些组织都无法阻止革新，甚至是一个不良的实现。为了更好地理解这个问题，下面介绍 WebMonkey 对浏览器兼容的比较图，详细信息请访问 http://hotwired.lycos.com/webmonkey/reference/browser_char/index.html。HTTP 和 HTML 自发更新的性质使得热忱的工程师可以自由地建立和发布新的 Web 技术。虽然这种情况对于大量的 Web 用户来说是有益的，但在测试 Web 使能应用程序的时候却出现了问题。

例如，对于早期的 Web，Internet 连接缓慢，而且经常失败。软件工程师编写的浏览器软件尽量通过网页的缓存帮助用户。在浏览器中单击 Back 按钮就可以马上跳转到以前缓存的网页，而不必在低速的 Internet 中向服务器发送耗时的请求。当在大量静态显示网页对所有用户都一致的情况下，这种方法有很好的效果。

随着 Web 的迅速普及，缓存方法对于将文档、电影、图形和声音从服务器传送到浏览器上变得非常重要。通过将缓存服务器设置在 Internet 的边界上，可以提高内容交付的速度，而且带宽的需求也随之降低了。虽然 Internet 浏览器包含缓存，但缓存依然是目前 Web 使能应用程序开发人员所面临的最大问题。即使 HTTP 1.1 规范中包括缓存方法，大多数浏览器也没有完全遵从该标准。

让我们在更深层次上检测缓存问题，从而可以看到不兼容性是如何影响 Web 使能应用程序的。大多数 Web 使能应用程序动态地创建每个网页，其中包括每个用户的个性化的内容。虽然 HTTP 1.1 规定了浏览器不能对页面进行缓存，但这些技术很少被采纳。Microsoft 的 Internet Explorer(MSIE)浏览器尤为严重。在 MSIE 中单击 Refresh/Reload 按钮无法保证浏览器与服务器相连接，并真正地请求页面。

设想在前面的银行 Web 使能应用程序的客户使用了登录页面，但是浏览器却显示转移网页的缓存版本。用户可能没有注意到缓存的内容中含有已经输入到表单的账号。服务器可能将信息保存到网页的缓存中，这使得第二次使用变得毫无意义。在许多情况下，浏览器缓存会导致用户不能达到他们的目标。

测试并且限制浏览器缓存的有效方法是取消浏览器缓存机制。例如，应该测试每个服务器的响应，以查看 HTTP 头部值中是否包含：Pragma：NoCache(其他 HTTP 头部包含 Expire 和 Last-Modified 信息，可以帮助服务器了解何时不需要缓存)。尽管许多浏览器忽略了没有缓存的头部，但利用这个信息经常会有助于减少用户遇到的缓存页面。另外，网页 HTML 内容应该在<head>中包括这个缓存标记：

```
<html>
<head>
   <meta http-equiv="Pragma" content="no-cache">
</head>
<body>
...
</body>
</html>
```

如果浏览器忽略了缓存标记，那么最好的办法是创建动态的网页内容，用户可以使用这些内容表明他们正在查看已经缓存的网页。例如，如果每个网页都包含一个自增的整数，那么刷新该网页应该会增加这个数值。相同数值的网页表明用户查看的是缓存的网页。另外，测试可以在 HTTP 响应头部中检测日期/时间值。

6.2.2 无效数据

浏览器通过使用 HTTP 协议向服务器发出 GET 和 POST 请求。GET 请求包括一个 URL、HTTP 头部信息和一系列名/值对。例如，设想一个网页提供很多电影，每个电影名字以超链接的形式使得用户通过单击即可进入。当用户单击一个链接的时候，浏览器向服务器发送一个 GET 请求：

```
GET /signin_handler?name=frank&movie=Star%20Wars HTTP/1.0
User-Agent: Mozilla 5.28
Host: examples.pushtotest.com
Accept: text/html, image/gif, image/jpeg, *;
Connection: keep-alive
```

虽然 HTTP 的 GET 命令是轻量级的而且非常通用，但它并没有告诉服务器任何有关数据标识的信息。服务器是如何知道 name 和 movie 的值的呢？它是如何从无效的 movie 值中知道正确的值呢？或者怎么知道 movie 值是用 URL 编码的呢？浏览器可以创建自认为是很有效的 GET 请求，但服务器可能无法接受。有效数据的软件测试方法对于部署高质量的 HTTP/HTML Web 使能应用程序来说是非常重要的。

为了抓住这些问题的所在，应该在每次测试 Web 使能应用程序的时候查找下面的每一种类型的无效数据：

- **太少或者太多的参数**——HTTP/HTML 环境没有定义要发送或者接收的参数说明。这需要开发人员和 HTML 设计人员在建立应用程序之前进行协调。通过发送比预计的数量要少的参数对 Web 使能应用程序进行测试通常会发现服务器的不良业务逻辑和安全漏洞。
- **顺序错误的数据**——应该安排正确触发的数据顺序。例如，一个已排序的测试会向服务器发送一个银行账号资金转移的命令，而不是在服务器会话中首先发送确定后面的账号的 GET 命令。测试应知道服务器如何处理无序的情况。
- **边界数据错误**——确定测试中有效数值的范围。如果一个名字不能超过 15 个字符，那么测试应该确定服务器是否可以处理 17 个字符长的名字。
- **格式错误的数据**——在 HTTP/HTML 环境中没有规定数据内容的形式。数据的每个部分都是一组字符。而且也没有对字符进行定义。在调用中的 HTTP 头部值可以选择性地包含编码类型的定义，例如 UTF-8。

下面介绍一个更深层次格式错误数据的示例。HTTP/HTML Web 使能应用程序由于 HTML 的性质，它特别容易受到无效数据问题的攻击。HTML 混合了在网页中布局内容的指令。即使目前流行的 HTML 编辑工具也可以方便地创建无效的 HTML 代码。必须创建具体的测试，查看服务器在收到无效的 HTML 格式时是如何进行响应的。例如，下面的 HTML 在第一个 input 标记中遗漏了一个双引号：

```
<html>
<body>
<form action="signin_handler">
<input name="signin_name value="Default user">
<input name="password" type="password" value="pass">
</form>
</body>
</html>
```

服务器接收如下形式的 POST 命令：

```
POST /signin_handler HTTP/1.1
Referrer: http://examples.pushtotest.com/...
Content-length: 178

signin_name%2Fvalue=&password=pass
```

注意 signin_name%2Fvalue=参数，这是由于遗漏了双引号而产生的。查看服务器是如何响应这种无效数据的，对于一个成功的测试方法十分关键，尤其是在 HTTP/HTML Web 使能应用程序中。

6.2.3　会话问题

HTTP/HTML 环境中的初始设计是无状态的。每个请求和响应都是独立的。动态、个性化的 Web 使能应用程序通过 Cookie、Applet、ActiveX 控件以及编码的 URL 实现了状态的变化。一旦引入了状态信息，服务器就需要在会话中记录状态数据。智能测试代理非常适合测试 Web 使能应用程序的会话问题。

智能测试代理方便地实现了以下会话测试：

- **无效会话标识**——每个 Web 使能应用程序根据各自的形式确定会话标识的格式。例如，PushToTest 网站的 Cookie 值的形式是：38849198981。每个有惟一 IP 地址的新用户会使得数值增加 1。测试代理应该尝试有效数据，例如从服务器接收到的这些数据。但是也应该引入会话标识从而可以查看服务器是如何处理无效数据的。
- **长会话**——每个会话都要求服务器使用资源保存会话数据。Web 使能应用程序在会话结束后将重新使用这些资源。通过长时间地使用相同的会话信息，测试代理可以很容易地将服务器资源的利用提高到最大值。

正如我们所看到的，在 HTTP/HTML Web 使能应用程序中可能会出现很多错误。建立、运行 HTTP 测试代理是发现并解决这些问题的一种很好的技术。

6.2.4　创建 HTTP 测试代理

在本节中，介绍了如何建立 HTTP 测试代理的脚本。为了熟悉该方法，我将给出一个完整的测试脚本，您可以在 TestMaker 中运行它。在第 5 章中已经介绍了 TestMaker。首先概述了智能测试代理，并且演示了代理脚本是如何向服务器发送请求、有效记录、会话以及重定向的，以及如何保证服务器响应的有效性。

智能测试代理技术的重要目的是在向用户暴露错误、失败和可扩展性问题之前了解系统的可扩展性、性能和功能特征。智能测试代理模拟用户原型，正如第 7 章中描述的缓慢、烦乱的 Wanderer 代理。图 6-3 说明了 Wanderer 是一个典型的智能测试代理，它与其他代理同时运行，模拟一台服务器同时处理多个用户的真实情况。其他并发运行的代理模拟各自的用户原型：Validator 随机地读取网页并检查网页的内容，Sign-in 代理用不同的用户名和密码尝试登录 Web 使能应用程序。

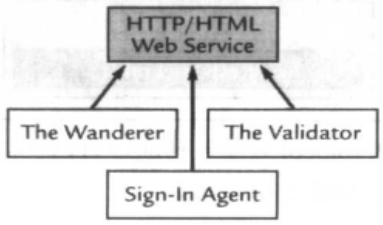

图 6-3　多个并发的正在运行的智能测试代理测试一个 HTTP/HTML Web 使能应用程序

Wanderer 是一个智能测试代理，它可以随机地读取 PushToTest 测试服务器上的网页，PushToTest 是 TestMaker 的主要维护者。Wanderer 开始使用 HTTPProtocol 对象获得网页。然后，

在网页中找到超链接并且随机选择一个。Wanderer 也记录接收每个网页的时间。Wanderer 每次在加载 10 个网页之后就会暂停，给加载时间最久的网页加以奖励。

TestMaker 提供了创建和运行 Wanderer、Sign-In 和 Validator 智能测试代理所需的一切。虽然 TestMaker 新代理向导可以通过方便的图形用户界面自动地创建智能测试代理，但了解 TestMaker 组件对于成功地编写和运行其智能测试代理还是非常重要的(见图 6-4)。

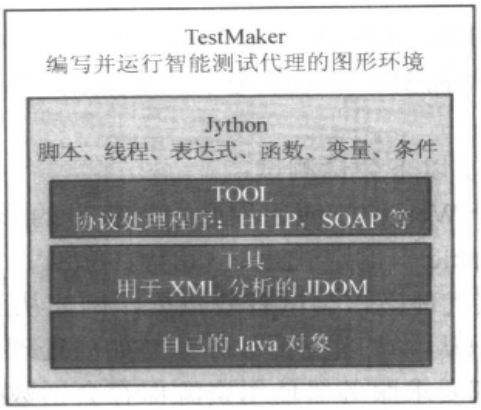

图 6-4 TestMaker 环境的体系结构视图，该视图展示了用于建立智能测试代理的全部组件

虽然第 5 章介绍了 TestMaker，但说明 TestMaker 组件彼此之间如何适应仍然是很重要的。TestMaker 定义了 TOOL，为使用 HTTP、HTTPS、SOAP、XML-RPC 协议与服务器进行通信的可扩展系列的协议处理程序提供了一个通用的接口。TestMaker 提供了 JDOM，它是处理 XML 数据的工具，在本章的稍后部分，Validator 代理将用到该工具中。

Jython 脚本语言是您的测试代理和 TOOL 对象之间的粘合剂。作为辅助工具，TestMaker 提供了查看 HTML 和编写测试 HTTP/HTML Web 使能应用程序的 Jython 脚本的一个记录器。

TOOL 实现了 HTTPProtocol 对象，可以将它用到 HTTP 和 HTTPS 协议中，这样就可以发出 GET 和 POST 请求，处理 HTTP 头部参数(包括 Cookie)，以及查找服务器响应。图 6-5 说明了 HTTPProtocol 对象的总体结构。

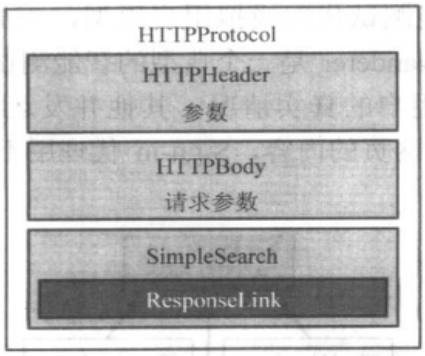

图 6-5 TOOL 的 HTTPProtocol 对象包括通过 HTTP 和 HTTPS 协议连接到主机、传递参数并搜索结果的对象

下一节将描述 Wanderer 代理如何使用 Jython 脚本语言建立 HTTPProtocol 对象，与服务器相连并返回响应。虽然脚本语言是没有测试代理特定限制的完全面向对象的语言，但通常会将一个智能测试代理分成以下几个部分：

- 引言和作者信息。这也解释了代理的目的
- 用于定位和使用 TOOL、Java 和 Python 对象的导入语句
- 变量的定义
- 函数的定义
- 主要代码
- 完成之后的分析和报表
- 清除和结束

6.3 现有的 HTTP 通信

图 6-3 描述了发送 HTTP/HTML Web 使能应用程序请求的 3 个并发智能测试代理。Wanderer 的角色是通过发出请求在 Web 使能应用程序上创建负载，这使得 Web 使能应用程序会对相对较大的数据块作出响应。Sign-in 和 Validator 代理的角色是测试和验证 Web 使能应用程序的核心功能，其方法是通过请求需要高级的业务逻辑的功能，例如客户的登录。

Wanderer 在 TOOL 中使用脚本语言创建并管理 HTTP/HTML 对象。在本节，我们介绍 Wanderer 代理，以查看 Python 和 TOOL 是如何协同工作的。下面是完整的 Wanderer 代理，之后是 Wanderer 组件的详细注释。本书列出的全部代码也可以通过 http://www.pushtotest.com/docs 处下载。

```
# Agent name: wanderer_agent.a
# Created on: May 15,  2002
# Author:  fcohen@pushtotest.com

print "Agent running: wanderer_agent.a"
print "Description:"
print "  This agent wanders the examples.pushtotest.com/
responder"

print "   Web site"
print "   Web site finding hyperlinks and following them."
Print "   Wanderer also keeps track of the time it takes"
Print "   to receive pages."
Print "   Every 10 pages wanderer awards the slowest page."
print

# Import tells TestMaker where to find Tool objects

from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  Header,  Body, HTTPProtocol,  \
HTTPBody, HTTPHeader

from com.pushtotest.tool.response import Response,  \
ResponseLinkConfig,  SimpleSearchLink

# Import useful Python and Java libraries
from urlparse import urlparse
from java.util import Random

# Global variable definitions

next_url = "http://examples.pushtotest.com/responder"
```

```
host = ""          # Holds the decoded host name from a URL
doc = ""           # and the document name from the URL
params = ""        # and the parameters of the call

f1 =  '<a href="http://'  # Used to search for hyperlinks
f2 =  '">'

worsttime = 0     # Tracks the page that took the longest
worstcount = 0
worstname =  ""

r = Random()      # A basic random number generator

# hostdoc_decoder: Decodes a URL into the host name

def hostdoc_decoder( theurl ):

        global host,  doc, params, next_url,  last_good url

         # urlparse is a handy library function that
         # returns a tupple containing
         # the various parts of a URL,  including host,
         # document, parameters,  etc.

        parsed_tup = urlparse( next_url )

        # Validate the parsed URL,  if it is invalid
        # return with host = null
        # which will signal that another URL is needed

        if  ( len( parsed_tup[1]  )  == 0 )  :
              host=""
              return

        host = parsed_tup[1]
        doc = parsed_tup[2]
        params = parsed_tup[4]

 #     print "host=",host," doc=",doc," params=",params

 # Main body of agent

print "Setting-up to make first request."

 # Create the needed objects to communicate with the host

httphandler = HTTPProtocol()

# Define a ResponseLink object to search for an <a href> tag

responselink = ResponseLinkConfig()
responselink.setParameter(  'beginsearch',  f1 )
responselink.setParameter(  'endsearch',  f2 )

# In the TOOL object hierarchy the search parameter
# definition is in a separate object so that a
# single response may have multiple search patterns

search = SimpleSearchLink()
```

```
search.init( responselink )

# Find n documents

print "Requesting document:  ",  doc

while 1:
      hostdoc_decoder( next_url )

      if host==""

          # The host we picked isn't valid so
          # raise an exception and end

          raise Spider_Error(  "Giving up!"  )

      httphandler.setHost( host )
      if params  ==  "":
          httphandler.setPath( doc )
      else:
          httphandler.setPath( doc + "?" + params )

      # Request the document from the host
      response = httphandler.connect()

      # Find the next document URL in the body of the response
       found = search.handle( response )

       # How many found items in the list
       foundcount = found.getParameterValue \
       ("simplesearch.foundcount")

       if  (  foundcount == 0 ):
           raise Spider Error(  "No document URLs found."  )

       # Pick a URL to load the next document
       foundlist = found.getParameterValues \
       ("simplesearch.founditems")
       doc = foundlist.get( r.nextInt( foundcount )  )

       # Remember the previous host just in case we need to
       # do some backtracking
       last_good_url = next_url

       # Next trim the <a href= and > tags to find the hyperlink

       next_url =  "http://"  + doc[ len(fl)  :  (len(doc)  \
       - len(f2)  )  ]

       print "links:  ",foundcount.toString()," \

       choosing:",next_url

      print "doc =",doc
      print

      # Time for an award to the page that had the worst time?
      if response.getTotalTime()  > worsttime:
            worsttime = response.getTotalTime()
```

```
            worstname = last_good_url

        worstcount = worstcount + 1

        if worstcount > 10:
            print "=================Award time==================="
            print "The award goes to:  ", worstname
            print "which took ",worsttime," in milliseconds \
            to complete."
            print
            worstcount=0

print "Agent finished."
```

Wanderer 直接向 examples.pushtotest.com 服务器发出请求。PushToTest 安装在这个服务器上面，它是 TestMaker 的主要维护者。下面，我们将介绍组成 Wanderer 的每个独立的部分。

TestMaker 绑定了 Jython，它是 Python 语言的 Java 实现。虽然在使用 TestMaker 中不必学习 Python，但对该语言的基本了解还是有益处的。TestMaker 包括了编写并维护测试代理的新建代理向导(New Agent Wizard)，它可以有助于使用 Python 语言。如果要了解 Python、Jython 和 TestMaker，请访问 http://docs.pushtotest.com，可以获得一些相关书籍和 Web 资源的详细信息。

在 Jython 中，每个 Python 对象都可以像 Java 对象一样实例化、维护、调用和销毁。Jython 还有从脚本语言中直接添加 Java 对象的优势。Import 命令会告知 Jython 在哪里查找用于代理脚本的 Python 和 Java 类。在 Jython 中使用 Java 对象的格式如下：

```
from package import object
```

import 语句使得我们可以在 Jython 脚本中访问 ProtocolHandler、HTTPProtocol、HTTPBody 和 HTTPHeader 对象。

```
# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import ProtocolHan-
dler,  Header,  Body,  HTTPProtocol,  HTTPBody,  HTTPHeader
from com.pushtotest.tool.response import Response, Response-
LinkConfig,  SimpleSearchLink

# Import useful Python and Java libraries
from urlparse import urlparse
from java.util import Random
```

这些 import 语句告知 Jython 在何处查找 TOOL 和 Java 对象的协议处理对象，例如 urlparse 和 Random 对象。urlparse 对象是获得 URL 并且将其拆成主机、端口号和文档参数的一个工具对象。Random 是 Java 的一个简单随机数发生器。下面，将创建代理以后需要使用的变量。

```
next_url = "http://examples.pushtotest.com/responder"

host = ""         # Holds the decoded host name from a URL
doc = ""          # and the document name from the URL
params = ""       # and the parameters of the call

f1 =  '<a href="http://'  # Used to search for hyperlinks
f2  =  '">'
```

```
worsttime = 0    # Keeps track of the page that took the longest
worstcount = 0
worstname  =  ""
```

这些命令创建了代理脚本中用到的变量。Java 开发人员使用的 Python 是一种动态输入的语言——在 Java 中每个变量需要在使用之前定义其类型，而 Python 利用所赋数据的类型来创建变量的类型。在 Python 中，不必将 next_url 定义为 String 对象。Python 将“http://examples.pushtotest.com/responder”作为一个字符串并且自动地将 next_url 定义为一个 String 对象变量。

```
r = Random()      # A basic random number generator
```

在前面列出的代码中，我们可以了解 String 对象是如何创建的。这里给出了所创建的第一个复杂对象。变量 r 现在指向了 Java 的 Random 类的一个新实例。

```
def hostdoc_decoder( theurl ):

    global host,  doc, params, next_url,  last_good_url
    parsed_tup = urlparse( next_url )

   # Validate the parsed URL,  if it is invalid return
   # with host = null
   # which will signal that another URL is needed

   if ( len( parsed_tup[1]  ) == 0 )  :
         host= ""
         return

  host = parsed_tup[1]
  doc = parsed_tup[2]
  params = parsed_tup[4]
```

函数和命令组在 Python 中用空格指示。Java 和 C 语言使用花括号{}、逗号和分号的组合来指示命令组。在 Python 中，在命令之前的空格数目定义了一组命令。例如，上面的 hostdoc_decoder()函数通过 def 命令进行定义，该函数的命令通过缩进带空格的命令进行编组。这种系统开始可能看起来不太正常，但它的简便却带来了很多益处。

hostdoc_decoder()是带有 URL 并将其解码为主机名、文档名和参数的函数。例如，hostdoc_decoder 将对下面的 URL 进行解码：

```
http://examples.pushtotest.com/htmlresponder/
responder?file=file1.html
```

解码后分成下面几部分：

- host: http://examples.pushtotest.com
- doc:/htmlresponder/responder
- params: file=file1.html

```
print

# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler, Header, Body, HTTPProtocol, \
HTTPBody, HTTPHeader

from com.pushtotest.tool.response import Response, \
ResponseLinkConfig, SimpleSearchLink

# Import useful Python and Java libraries
import sys
import java
from urlparse import urlparse

# hostdoc_decoder: Decodes a URL into the host name
# and document name

host = ""          # Holds the decoded host name from a URL
doc = ""           # and the document name from the URL
params = ""        # and the parameters of the call

def hostdoc_decoder( theurl):

    global host, doc, params  http_ph

    # urlparse is a handy library function that returns
    # a tupple containing
    # the various parts of a URL, including host, document,
    # parameters, etc.

    parsed_tup = urlparse( theurl )

    # Validate the parsed URL, if it is invalid return
    # with host = null
    # which will signal that another URL is needed

    if ( len( parsed_tup[1] ) == 0 ) :
        host=""
        return

    host = parsed_tup[1]
    doc = parsed_tup[2]
    params = parsed_tup[4]

    http_ph.setHost( host )
    if params == ""
       http_ph.setPath( doc )
    else:
       http_ph.setPath( doc + "?" + params )

#    print "host=",host," doc=",doc," params=",params

# Main body of agent

print
"============================================================"
```

```
print "Step 1: Get the sign-in form"
print

# This URL returns an HTML page that contains a
# sign-in form

urlone = "http://examples.pushtotest.com/responder \
/htmlresponder?file=file2.html"

http_ph = ProtocolHandler.getProtocol("http")

hostdoc_decoder( urlone )

print "Sending request."

response = http_ph.connect()

# get an Iterator of all the keys

k = response.getParameterKeys();

for i in k:
   kv = k.next()
   print kv,":",response.getParameterValue( kv )
   print

print
"============================================================"
print  "Step 2:  Do a form submit to sign-in"
print

urltwo = "http://examples.pushtotest.com/responder \
/htmlresponder"
urlthree = "http://examples.pushtotest.com/responder \
/htmlresponder?file=file3.html"

hostdoc_decoder( urltwo )

http_ph.setType( HTTPProtocol. POST)

body = ProtocolHandler.getBody(  "http"  )

# send the sign-in values
body.addParameter(  "login",  "myname"  )
body.addParameter(  "passwd",  "secret" )

# Next we use some of the responder servlets custom
# commands. Details of these commands are found
# on http://examples.pushtotest.com/responder/htmlresponder
# Tell the responder to return a redirect response
# code to urlthree
body.addParameter(  "redirect", urlthree )

# Tell the responder servlet to set a cookie
body.addParameter(  "cookie",  "sessionid"  )
body.addParameter(  "cookievalue",  "388281v90981"  )

# Tell the responder to return the contents of file2.html
```

```
body.addParameter( "file", "file2.html" )

http_ph.setBody( body )

# We use a special form of connect() which tells the
# http protocol handler not to follow any redirect
# commands received from the host

print "Sending request."

response = http_ph.connect( 0 )

print
print "Here is what we got back from the host:"
print "response code =",response.getResponseCode()
print
print "Response parameters from the host:"

# get an Iterator of all the parameter keys

meb = response.getParameterKeys()

for i in meb:
    print i, "=" response.getParameterValue( i )

# Display the cookie name/value pairs
# received from the host

# Display the cookie name/value pairs received from the host
print
print "jCookie values:"
for i in http_ph.getCookies():
    print "bop=",i

print
print
print "=============================================="
print "Step 3: Manually handle a redirect command"
print "from the server"
print

urlfour = response.getParameterValue( "Location" )

print "The host told us to redirect to:"
print urlfour
print

hostdoc_decoder( urlfour )

# In Step 2 we added a parameter that told the
# examples/responder servlet to send back a redirect
# command. Since we're using the same http_ph object
# that we used in Step 2 we need to tell the
# object to not send us the redirect this time.
# This situation is unlikely in a normal test but
# the following command is needed for this example agent.
body.removeParameter( "redirect" )
```

```
# Since we are reusing the http_ph object we used in Step 2
# this object already has the received cookie values from
# last host response and will play them back to the host

print "Sending request."

response = http_ph.connect()

print
print "Here is what we got back from the host:"
print "response code =",response.getResponseCode()
print
print "Response parameters from the host:"

# get an Iterator of all the parameter keys

meb = response.getParameterKeys()

for i in meb:
    print i,  "=", response.getParameterValue( i )

print
print "Agent finished."
```

Sign-in 代理直接向 examples.pushtotest.com 服务器发送请求。PushToTest 是 TestMaker 的主要维护者，安装在这个服务器上。Sign-in 代理说明了一组典型的命令，这些命令一起为用户提供了使用 HTTP/HTML 协议的登录功能。该代理将这些命令分成以下 3 个部分：

- 获得 Sign-in 表单。
- 执行 POST 命令进行登录。
- 接收 cookie 并且处理来自服务器的重定向响应。

下面看看组成 Sign-in 代理的每个部分。代理脚本导入代理将使用的 TOOL、Python 和 Java 对象，声明一组该代理所用的变量以及它所使用的函数。

```
def hostdoc_decoder(  theurl  ):

    global host,  doc, params, http_ph

    # urlparse is a handy library function that
    # returns a tupple containing
    # the various parts of a URL,  including
    # host,  document,  parameters,  etc.

    parsed_tup = urlparse( theurl )

    # Validate the parsed URL, if it is invalid
    # return with host = null
    # which will signal that another URL is needed

    if  ( len( parsed_tup[1]  )  == 0 )  :
         host=""
         return

    host = parsed_tup[1]
    doc = parsed_tup[2]
```

```
params = parsed_tup[4]

http_ph.setHost( host )
if params == "":
    http_ph.setPath( doc )
else:
    http_ph.setPath( doc + "?" + params )
```

Sign-in 代理使用了以前在 Wanderer 代理中首次介绍的 hostdoc_decoder()函数的改动版本。hostdoc_decoder()不仅可以解码主机、文档和 URL 中的参数，而且还能将这些值填入到 HTTPProtocol 的全局对象中，该对象可以通过 http_ph 变量进行访问。

```
urlone = "http://examples.pushtotest.com/responder \
 /htmlresponder?file=file2.html"

http_ph = ProtocolHandler.getProtocol("http")

hostdoc_decoder( urlone )

response = http_ph.connect()
```

Sign-in 代理的第一步使用 HTTPProtocol 对象获得一个包含 HTML 登录表单的网页。在浏览器中会出现用户所需填写的身份和密码，以及单击的登录按钮。response 对象保存了来自服务器的响应，包括一系列在登录之后所需的 HTTP 头参数。

```
k = response.getParameterKeys();

for i in k:
    kv = k.next()
    print kv,":",response.getParameterValue( kv )
    print
```

变量 k 指向一系列服务器响应中的 HTTP 头参数。Jython 的 for 命令可以方便地遍历该列表，然后打印每个头参数的值。列表包含了每个参数的键或者标题，getParameterValue()函数可以返回任意给定键的数值。

接下来，Sign-in 代理向服务器发送登录请求。

```
http_ph.setType( HTTPProtocol. POST)

body = ProtocolHandler.getBody( "http" )

# send the sign-in values
body.addParameter( "login", "myname" )
body.addParameter( "passwd", "secret" )
```

代理脚本的这个部分再次使用了前面创建的 http_ph 的 HTTPProtocol 对象。setType 方法告诉 HTTPProtocol 对象向服务器发送 Post 请求，代理使用 addParameter 向请求体添加登录的 ID 和密码值。

与多数服务器不同，examples.pushtotest.com 服务器上的 htmlresponder servlet 提供了几个功能，它们用于说明真实测试代理遇到的问题。特别地，通过发送名为 redirect 的参数，服务器的响应会包括已定义的 URL 的 302 重定向命令。

```
body.addParameter( "redirect", urlthree )
```

通过发送 htmlresponder 和名为 cookie 的参数，服务器的响应会包括 cookie 的参数和值。

```
body.addParameter( "cookie", "sessionid" )
body.addParameter( "cookievalue", "388281v90981" )
```

最后，通过发送名为 file 的参数，服务器将返回在其上找到的文件内容。

```
body.addParameter( "file", "file2.html" )
```

htmlresponder servlet 还包括这里没有描述的额外命令。如果想要获得 htmlresponder 的全部命令列表，请访问 http://examples.putshtotest.com/responder/htmlresponder。

```
response = http_ph.connect( 0 )
```

在通常的操作下，当 TOOL 中的 HTTPProtocol 对象接收到 302 重定向响应的时候，它将自动地请求该重定向网页。为了解释 302 重定向响应，Sign-in 代理使用了 connect()方法，该方法不会自动地请求重定向的页面。

```
# Display the cookie name/value pairs received from the host
print
print  "jCookie values:"
for i in http_ph.getCookies():
    print "bop=",i
print response.getParameterValue("Cookie")
```

代理脚本然后使用 getCookies()方法查找用协议处理程序管理的 cookie 列表。在实际情况中，一个 HTTP/HTML Web 使能应用程序很可能一次会使用多个记录。TestMaker 使用 Sonal Bansal 的 jCookie 库处理 cookie，而且提供了处理 cookie 的几个简便方法。关于 jCookie 的详细内容请访问 http://docs.pushtotest.com。在通常情况下，当 HTTPProtocol 对象处理 cookie 时可能不需对 cookie 进行任何操作。

Sign-in 代理的最后一步使用了同一个 http_ph 对象来请求重定向网页。TOOL 的 HTTPProtocol 对象自动地将以前从服务器接收的响应的 cookie 值再次发送给服务器。

在这一小节中，我们介绍了 Sign-in 代理是如何使用 Tython 脚本和 TOOL 对象来处理 HTML 表单、会话和 cookie 的。下面一节将介绍 Validator 代理，并且介绍服务器响应数据是如何被搜索、分析和验证的。

6.3.2 验证响应数据

前面两节着重介绍了建立产生服务器负载以及在基于 cookie 会话中执行一组命令的测试代理。Validator 代理是一个智能测试代理，它用于测试服务器对有效数据的响应。Validator 说明了通过服务器响应进行查找的 4 种不同的技术，包括：

- 查找并发现使用 ResponseLink 方法的响应值。
- 通过使用脚本语言命令对响应数据进行查找。
- 分析使用 Tool 命令的 HTML 表单。
- 发现使用 JDOM 命令的 XML 数据。

下面是 Validator 代理的全部代码，其后是对 Validator 组件的详细解释。

```
# Agent name:  checker_agent.a
# Created on: May 17,  2002
# Author:  fcohen@pushtotest.com

print  "Agent running:  checker agent.a"
print  "Description:"
print  "  This agent shows how to find useful
print  "information in the response"
print  "  received from host that returns HTTP/HTML
print  "content. Three ways"
print  "  of searching through the response are
print  "presented,  including:"
print  "  1)  Search & find using ResponseLink methods"
print  "  2)  Searching through response data using
print  "the scripting language commands"
print  "  3)  Parsing HTML forms using Tool commands"
print  "  4)  Finding XML data using JDOM commands"

# Import tells TestMaker where to find Tool objects

from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  Header,  Body,  HTTPProtocol,  \
HTTPBody,  HTTPHeader

from com.pushtotest.tool.response import Response,  \
ResponseLinkConfig,  SimpleSearchLink

# Import useful Python and Java libraries
import sys
import java
from urlparse import urlparse
from java.util import Random

# hostdoc_decoder: Decodes a URL into the host name and doc-
ument  name

host = ""         # Holds the decoded host name from a URL
doc = ""          # and the document name from the URL
params = ""       # and the parameters of the call

def hostdoc_decoder( theurl ):

      global host, doc, params, http_ph

      # urlparse is a handy library function that
      # returns a tupple containing
      # the various parts of a URL,  including host,  \
      # document, parameters,  etc.

      parsed_tup = urlparse( theurl )

      # Validate the parsed URL,  if it is invalid return
      # with host = null
      # which will signal that another URL is needed

      if  ( len( parsed_tup[1]  )  == 0 )  :
```

```
            host=""
            return

        host = parsed_tup[1]
        doc = parsed_tup[2]
        params = parsed_tup[4]

        http_ph.setHost( host )
        if params == "":
            http_ph.setPath( doc )
        else:
            http_ph.setPath( doc + "?" + params )

#    print "host=",host," doc=",doc," params=",params

# Main body of agent
print
"============================================================"
print "Technique 1: Search & find using ResponseLink"
print

# This URL returns an HTML page that contains several
# <img> tags
urlone = "http://examples.pushtotest.com/responder \
/htmlresponder?file=file4.html"

http_ph = ProtocolHandler.getProtocol("http")

hostdoc_decoder( urlone )

# Establish search criteria

# Tool implements a simple system for searching
# and transforming an HTTP response. We begin by creating
# a ResponseLinkConfig object that defines the search
# parameters.  In this case we're looking for <img> tags.

responselink = ResponseLinkConfig()
responselink.setParameter(  "beginsearch",  "<img"  )
responselink.setParameter(  "endsearch",  '>'  )

# Create a search object to handle the actual search.
# And we tell search to use the responselink.
# For example one uses the same
# ResponseLinkConf object to search through XML data when
# making SOAP calls.

search = SimpleSearchLink()
search.init( responselink )

print "Sending request."

# Connect to the host, get the document and return
# a response object
response = http_ph.connect()

# Find the next document URL in the body of the response
```

```
found = search.handle( response )

print "Here is what we learned from the SimpleSearchLink"
print
foundcount = found.getParameterValue("sim-
plesearch.foundcount")
print "Count of <img> tags:",foundcount
print

if foundcount > 0  :

     # This gets a list of the found items
     foundlist = found.getParameterValues \
     ("simplesearch.founditems");

     print "Here is a list of the <img> tags we found:"

     for i in foundlist:
          print i

     print
     print "We choose at random this <img> tag:"

     r = Random()      # A basic random number generator

     print foundlist.get( r.nextInt( foundcount )  )
     print

print "================================================="
print "Technique 2)  Searching through response data"
print "using the scripting language commands"
print

# We will use the <img> tags we already received
# in Technique 1  (above)  to find the <img> tag that
# has the longest width value

if foundcount > 0  :

     # This gets a list of the found items
     foundlist = found.getParameterValues \
     ("simplesearch.founditems");

     # this keeps track of the widest <img> tag
     widest = 0
     widestkey = 0

     for i in foundlist:
        # There are 3 different conditions for finding
        # the width value in an <img> tag:
        # width=42    is the simplest,  the value begins
        # after = and ends with a space
        # width="42" the value begins after the first
        # quote and ends with the second quote
        # width=42>   the value begins after the = and
        # ends with the >
```

```
            # first we get rid of those pesky " quote
            # characters
            widstr = i.replace( '''', '')

            # then we look for width=
            lexerr = "width="

            lexlen = len( lexerr )

            mark = widstr.find( lexerr )

            rest = widstr[ lexlen+mark : ]

            theend = rest.find( " " )
            if theend == -1:
                theend = rest.find( ">" )

            widthstr = rest[ 0 : theend ]

            if int( widthstr ) > widest:
                widest =int( widthstr )
                widestkey = i

        print "widest <img> tag = ", widestkey
        print

else:
        print "Didn't get any <img> tags in Technique 1,"
        print " so skipping Technique 2."
        print

print "============================================================"
print "Technique 3) Parsing HTML forms using Tool commands"
print
# Technique 3 is going to use special Tool objects to
# handle HTML parsing
# and may possibly throw some Java exceptions

from com.pushtotest.tool.parser.html import \
HTMLParser
from java.net import URISyntaxException, \
MalformedURLException, URI
from java.io import IOException

urltwo="http://examples.pushtotest.com/responder \
/htmlresponder?file=file2.html''

try:
    uri =URI( urltwo )
    parser = HTMLParser( uri )
except URISyntaxException, ue:
    print "URLSyntaxException =",ue
    sys.exit()
except MalformedURLException, mfe:
    print "MalformedURLException =", mfe
    sys.exit()
```

```
except IOException,  ioe:
    print "IOException =",ioe
    sys.exit()

forms = parser.parse();

print "This URL contains",  len( forms ),"forms."
print

formnum =  0

for i in forms:
    formnum = formnum + 1

    print "Form",formnum, ":"
    print
    print "This is an HTML form that uses a",  \
    i.getMethod(),"command"
    print "to the host at:",i.getAction()
    print

    print "Here are all the <input> tags in the form:"
    print i
    print

    print "Here are the individual <input> tags:"

    for j in i.getElements():
        print " name=",j.getName()," default"
        print "value=",j.getValueAsString()
    print

print
print

print "=================================================="
print "Technique 4)  Finding XML data using JDOM commands"
print

# Technique 4 uses JDOM and Java StringReader objects to
# parse through XML data
from org.jdom.input import SAXBuilder
from java.io import StringReader

urlfour = "http://examples.pushtotest.com \
/responder/marketingfiles.xml"

hostdoc_decoder( urlfour )

print "Requesting XML document from host."
print

# Connect to the host, get the document and
# return a response object
response = http_ph.connect()

print "Here is the raw response document:"
```

```
print response
print

# Use the JDOM SAXBuilder object to create a new
# DOM object that
# will parse through the XML data we received
# from the host.

builder = SAXBuilder()
doc = builder.build(StringReader(response.toString()))

rootElement = doc.getRootElement()

print "The root element for this XML document is named:",
print rootElement.getName()
print

print "Here are the children of"
print " the",rootElement.getName(),"element:"

for i in rootElement.getChildren():
     print  " " ,i.getName()

     # now let's show the children element
     # for each employee element
     for j  in i.getChildren():
         print  " " ,j.getName(),"=",j.getText()
     print

print
print "Agent finished."
```

Validator 代理直接向 examples.pushtotest.com 服务器发送请求。该代理说明了验证、解析和分析服务器响应的 4 种不同的技术。

第一种技术使用了 TOOL 的 ResponseLink 对象和方法来查找服务器响应。其中的大部分代码在前一节的 Wanderer 中已经解释过了，因此我们把重点放在其他技术上面。

Validator 代理在来自服务器的 HTML 响应中查找<img>标记。这些标记以<img 开始，并以>结束。

```
responselink = ResponseLinkConfig()
responselink.setParameter(  "beginsearch",  "<img"  )
responselink.setParameter(  "endsearch",  '>'  )
```

ResponseLink 对象定义了查找参数，SimpleSearchLink 对象对结果进行真正的查找。

```
search = SimpleSearchLink()
search.init( responselink )
```

HTTPProtocol 对象在接收服务器响应的时候会返回 response 对象。

```
response = http_ph, connect()
found = search, handle( response )
foundcount = found.getParameterValue ( "sim-
plesearch, foundcount" )
```

从 response 对象中，代理可以访问 ResponseLink/SimpleSearchLink 中找到的条目。Simplesearch.foundcount 值返回在服务器响应中找到的条目数量。

```
r = Random()      # A basic random number generator
print foundlist.get( r.nextInt( foundcount ) )
```

在 foundcount 方法中结合使用 Java 的 Random 对象，代理能够随机地选取一个找到的条目。ResponseLink 和 SimpleSearchLink 对象可以在响应之间不断地重用。

在第一种技术中，Vallidator 向服务器发送一个请求并查找<img>标记的响应。<img>标记生成的列表形式如下：

- <img sre="images.pushtotest.com/logo.gif" width=401 height=58 border=0>
- <img src="images.pushtotest.com/empty, gif" width=l0 height= 1 border=0>
- <img src="images.pushtotest.com/man.gif" width=l0 height=8 border=0>

如果存在一个<img>标记的列表，代理如何发现最宽的图像呢？<img>标记本身是简单的文本字符串。解决这个问题要求代理分析其结果以查找每个<img>标记中的 width 属性。在测试 HTTP/HTML Web 使能应用程序的时候，分析 HTML 结果数据是一个很常见的问题。

Validator 代理使用内置的 Jython 命令处理字符串从而解决这种分析问题。

```
foundlist = found.getParameterValues("simplesearch.founditems");
```

代理首先得到所找到的条目列表，以下是完整的<img>标记。

```
widest = 0
widestkey = 0
```

代理建立 widest 变量，以保存最宽的<img>标记，widestkey 变量保存了在扫描列表时指向最宽<img>标记的指针。代理脚本使用 for 命令对列表中的所有标记进行循环。在每次循环中，脚本会发现标记中的 width 属性并且确定其宽度属性的数值。

HTML 是很灵活的标记语言。这种灵活性也会使人感到疑惑。下面的 HTML 表示法中的每个 width 属性都是合法的。

```
width=42
width='42'
width="42"
width=42>
```

Python 语言自带的功能强大的 String 对象可以处理文本数据。String 对象实现了查找、替换和改变文本数据的许多操作。代理脚本首先用 String 中的替换方法将引号改为双引号字符来查找 width 属性。然后，代理通过查找方法对 width 属性的开始部分进行查找。最后，代理使用 Sting 对象的一组混合方法确定 width 值的开始位置和结束位置。代理使用 int()操作符将 width 值的字符串转化为整数类型，并且和 widest 变量的值进行比较。在循环结束时，widest 变量保存了最宽<img>标记的宽度，而且 widestkey 变量保存了指向最宽<img>标记的指针。

HTML 内容的所有分析可能看起来有些错误，但事实的确如此！每个增加 HTML 标记新表单的高级功能都使得它更加难于分析 HTML，也不容易找到代理所需的数值。对于处理 HTML 内容的任何 Web 使能应用程序都会遇到这种相同的问题。XML 技术的许多驱动技术都来自处理 HTML 内容的软件开发人员的感觉。有关软件社区对 HTTP/HTML Web 使能应用程序的响应的更多信息参见第 7 章。

TOOL 为在 HTML 网页中查找<form>标记提供了一个简单的 HTMLParser 对象。Validator 代理的第 3 种技术使用了 HTMLParser 来查找<form>标记。

```
from com.pushtotest.tool.parser.html import HTMLParser
from java.net import URISyntaxException,  \
MalformedURLException, URI
from java.io import IOException
```

Import 命令表明在 TOOL 的何处可以找到 HTMLParser 对象。另外，我们导入了几个 Java 对象以处理在使用 HTMLParser 时可能出现的错误情况。这段脚本语言为处理使用对象和方法可能引起的错误情况提供了 try/except 机制。代码在 try 中执行直到有异常发生，这时程序将跳转到 except 块中。

```
try:
    uri =URI( urltwe )
    parser = HTMLParser( url )
except URISyntaxExceptien,  ue:
    print "URLSyntaxException =",ue
     sys.exit()
except MalformedURLException,  mfe:
     print  "MalformedURLException =",  mfe
     sys.exit()
except IOException,  ioe:
     print "IOException =",ioe
     sys.exit()
```

例如，当使用 URI(urltwo)函数时，如果 urltwo 包含一个无效的 URL，例如 httx://blah.com，那么这段脚本将会在 except MalformedURLException: 的块中继续执行。因此，异常消息被打印到输出窗口，并且该代理仍然存在。

```
forms = parser.parse();
```

parse.parse()方法返回在已分析的网页中所找到的<form>标记列表。Validator 代理使用 for 命令循环每个表单。每个表单的内容会出现在输出窗口中。

验证服务器响应数据的最后一种技术是在服务器响应中查看 XML 数据。这应该使您对后续章节的基于 SOAP 的 Web 使能应用程序产生兴趣。

```
from org.jdom.input import SAXBuilder
from java.io import StringReader
```

第 4 种技术使用了 Java 对象库 JDOM(http://www.jdom.org)，它用 Java 中心化的方式来处理 XML 数据，Java 的 StringReader 对象分析 XML 数据。JDOM 是针对 Java 的分析 XML 文档的面向对象接口。JDOM 以后会出现完全的 Java 版本。

```
builder = SAXBuilder()
doc = builder.build( StringReader(response.toString()  )  )
```

我们新建了 JDOM 的 SAXBuilder 对象，它通过 builder 变量进行引用。SAXBuilder 将代理在服务器响应中接收的 XML 文档读入到内部的文档对象模型(DOM)表示中。通过 DOM 格式，代理可以使用几种 XML 查询和操作方法来获得所需的数据。

XML 文档的形式如下：

```
<marketing>
   <employee>
      <id>582293</id>
      <firstname>Frank</firstname>
      <lastname>Cohen</lastname>
      <phone>408 374 7426</phone>
   </employee>
   <employee>
      <id>582343</id>
      <firstname>Betsy</firstname>
      <lastname>Hilbert</lastname>
      <phone>408 374 0000</phone>
   </employee>
</marketing>
```

JDOM 命令使得查找 XML 树中的元素、属性和数值变得更加方便，例如：

```
rootElement = doc.getRootElement()
```

doc.getRootElement()返回一个表示<marketing>标记及其所有后代的对象。然后，后续的函数可以查找<marketing>标记的后代。对于处理 HTML 或者 XML 数据的方法，您会选取哪一种呢？

6.4 本章小结

本章包含了 HTTP 和 HTML 的基本技术。介绍了 TestMaker 如何为处理 HTML 表单、cookie、会话和复杂的响应数据提供脚本语言命令和测试对象库。最后，本章涵盖了几种验证 Web 使能应用程序响应的策略，包括检验一个响应是否是使用 XML 进行编码的。

本章详细列出了互操作系统上的软件开发的发展过程。HTTP/HTML Web 使能应用程序的诱惑点是，通过使用简单的请求响应协议，就可以开发出功能强大的系统。掌握 HTTP/HTML 不久之后，开发人员想知道为什么自己还不能通过相同的 Web 连接来移动更复杂的请求和响应。实际上他们是可以做到的。这个事实引领了基于 SOAP、WSDL 和 UDDI 的 Web 使能应用程序的时代，这些将是第 7 章的主题。

第 7 章　协调 SOAP 和 XML Web 服务

在前面的章节中，我们介绍了许多软件开发人员在开发 Web 使能应用程序时所采用的过程。建立在 HTTP/HTML 协议和技术上的系统使用了 HTTP 的简单请求-响应协议，以在台式机和服务器之间传输数据。主管建立基于 HTTP/HTML 系统的开发人员常常感到很疑惑：为什么同样的网络互连不能传输更复杂的请求和响应。

例如，当一个基于 HTTP/HTML 的应用程序从浏览器端发送信用卡号码到服务器时，服务器需要专门的业务逻辑来确信该数字是正确的卡号。HTTP 和 HTML 不能定义发送和接受的数据类型。除非开发人员编写定制代码，否则服务器甚至不知道输入的值是否为信用卡号。使用 HTTP 协议时，信用卡号码和电话号码是一样的——它们都是发送到服务器上的字符串。怎样使得服务器能够分辨信用卡号码和电话号码之间的差别呢？开发人员会想到使用 XML 和 Web 服务。

世界上许多企业已经花费了数十亿美元，以使得他们的业务能够更加高效。现在，集成可以在很大程度上增加业务的效率和产量。XML 不仅可以发送数据，还能够发送关于数据类型的描述。所以企业中已部署的功能可以通过 XML 来识别，并利用 HTTP/HTML Web 服务基础架构来访问。

利用基于 XML 的协议交换数据，业务可以将存在的 Web 功能发送给合伙人、供应商、厂商和客户。使用 XML，开发人员可以通过集成几个系统的功能来部署下一代信息系统应用程序。开发人员也可以使用 XML 将原有系统和非 Web 使能应用程序集成到 Web 服务模型中。最终，软件开发人员在现有的网络基础架构中使用 XML 来寻找编程模型。

最早尝试提供 XML 协同编程模型的是客户端/服务器式的体系结构。客户端软件知道如何联系服务器，并发送包含 RPC 数据编码的 XML 文档。这样就产生了 XML-RPC。关于 XML-RPC 的详细信息请访问 http://www.xmlrpc.com。一般需要更通用的体系结构来减少系统的脆弱性和增强系统的可维护性。这样随后很快出现了 SOAP 和 Web 服务描述语言(WSDL)。详情请访问 http://www.w3.org/TR/ws-desc-reqs/和 SOAP http://www.w3.org/TR/soap12-part0/。

XML-RPC 和 SOAP 开创了软件技术的一个热点领域，这些技术使得一个系统中的多个服务之间能够协同工作。因此，需要很好地理解这些技术和体系结构所传递的解决方案，其中这些解决方案能够在各种情形和环境中进行很好的扩展、执行和运行。本章介绍了一些软件测试技术和工具的一个新的完整集合，这些工具能够查找并解决基于 XML-RPC 和 SOAP/WSDL 的 Web 服务的问题、bug 和可扩展性问题，并且能够在用户遭遇到这些问题之前就先查找并解决它们。

7.1　Web 服务简介

访问网页时，很可能会用到多个服务器软件应用包。对于提供个性化内容的动态网页尤其如此。例如，当您在 Internet 网站上购买书籍时，可能是一个服务负责登录系统，一个服务负责选择订单页，另一个服务负责发货和订单跟踪。如果做得不错，用户会遇到一个协调的、无缝连接的系统。

有时候，工程师们会竭尽全力以保持后端系统的互操作性。增加新的硬件(路由器、服务器和存储器)和修改服务器软件(WebLogic、WebSphere、SAP 和 Oracle)是常有的事。软件工程师通常编写昂贵且难以维护的定制软件使各系统内共享数据。

SOAP 和 WSDL 的发明是我们最大的希望，它们能够开发应用程序之间进行通信的通用语言。通过编写应用程序，使之能与其他的应用程序、平台和硬件自由通信。软件开发人员可以花费更少的时间来维护代码而集中精力解决更困难的问题。操作管理员可以更广泛地选择平台和硬件。用户可受益于更高级的应用程序。总之，SOAP 和 WSDL 传递了一种语言独立、基于标准的客户操作技术，它可以运行于多种不同操作系统和应用服务器上。

您可能在当地购买了某种软件，以用于特定的用途。该软件位于塑料封装的光滑的大盒子里，其中包括一个手册和一个包含上百兆字节程序代码的压缩光盘。有时候我很发愁，因为我领导开发组到一个又一个软件商店去购买软件包——包括 Norton Utilities、SoftWindows 和 TuneUp。我们编写的软件通过发行人或中间人递送，通过转售卖给用户，并安装在台式机上。整个过程可否简化？

起初软件行业喜欢这种方式。3 或 4 个字处理器软件、电子制表软件和数据库相互竞争。因为它们是定义了清晰界限的自包含软件包，所以工程师都喜欢编写这些产品。遗憾的是，对于软件开发人员，Mocrosoft 公司对桌面软件的垄断束缚意味着工程师很难发现投资者愿意编写新的字处理器、电子制表软件和数据库——至少对于台式机是这样。

毋庸置疑，由于桌面应用程序软件的垄断控制，未来 10 年的技术革命主要集中于软件应用程序运行需要的业务逻辑和表示库。图 7-1 显示了头 3 代可互操作的软件体系结构。

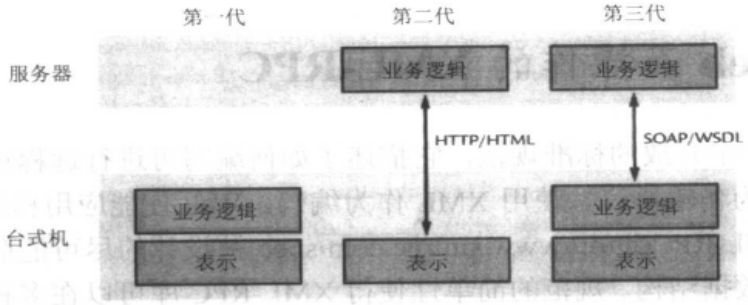

图 7-1　软件体系结构随着应用程序的业务逻辑和表示软件的变化而改变

20 世纪 80 年代末期，通讯工业的反常性开创了台式机间廉价 modem 互联的可能性。未来 10 年里，公共可访问的网络会被广泛地廉价使用。全世界的工程师会欣喜若狂。他们可以为服务器编写字处理器、电子表格和数据库，只需使用台式机作为瘦小的哑客户端，而他们的大部分代码都在服务器上运行。

截止到 2001 年，Microsoft 使用新的组件模型、普遍深入的 XML 支持以及作为中间件技术核心的虚拟机来革新他的工具、服务器软件和操作系统产品。Microsoft 统称这些为.NET(在第 9 章专门提供了与.NET 相关的测试策略和方法学)。Microsoft 构想运用组件使现有台式机上的字处理器、电子表格和数据库软件通过使用 XML Web 服务，可以和别的台式机上的字处理器、电子表格和数据库软件进行互操作，有时候甚至不需要服务器。这使得工程师很迷惑，开始尝试新的设计，如图 7-2 所示。工程师可以创建和部署台式机、精密器具和服务器上的计算能力的方案。软件开发人员、QA 分析师和 IT 管理员需要技术和策略来处理这些变化着的体系结构。

Microsoft 更长远的为.NET 考虑，使之支持 UDDI。UDDI 服务使软件组件能发现其他软件部件中的业务逻辑。UDDI 规范允许软件模型通过功能进行分类。组合 SOAP、WSDL 和 UDDI，可以使软件组件发现、绑定和远程调用完成该组件的工作所需的功能。在编写代码时，很少词典撰写者和分类专家致力于 UDDI 系统，因此还没实现基本的分类和查找函数的功能。

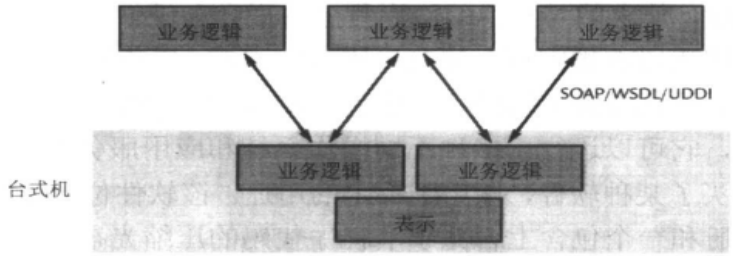

图 7-2 使用世界支持的 HTTP/HTML Web 基础架构，SOAP 成为世界互操作 Web 服务的公共语言，运行在台式机上并和一些内容资源进行通信

与 Microsoft 竞争的 Java 平台、应用服务器和工具的提供者也准备与 XML 协作。SOAP 和 WSDL 很快得到认可。每个公司都受益于 SOAP/WSDL。IBM 使用 SOAP 使 XML 继承客户软件，包括 COBOL 和 FORTRAN 应用程序。BEA 最先使用 Java Web Service(JWS)标准，使 Java 开发人员把现有的类和方法作为 SOAP 接口。SOAP 很快取得了平台中立的广泛业界支持。Microsoft 的最新 Web 服务方面的信息请访问 http://msdn.microsoft.com/Webservices/。

您需要很好地理解 XML-RPC 和 SOAP/WSDL 系统，以便理解如何测试基于 Web 服务的系统。下面介绍 XML-RPC 协议。

7.2 用于 Web 互操作的 XML-RPC

XML-RPC 是一个开放的标准规范，它描述了如何编写可进行远程过程调用的软件，这些软件使用 HTTP 作为传输协议、使用 XML 作为编码。Web 使能应用程序解码 XML 并调用服务器上的方法。XML-RPC(http://www.xmlrpc.com/spec)被设计的尽可能简单，用它可以传送、处理和返回复杂的数据结构。规范的简单性使得 XML-RPC 库可以在多种操作系统和编程语言(Perl、Python、 Java、Frontier、C/C++、Lisp、 PHP、 Microsoft .NET、Rebol、 Real Basic、Tcl、 Delphi、 WebObject 和 Zope)上实现。为了避免大家忘记，这里提醒一下，Userland 软件公司在基于脚本的应用程序框架上初次实现了 XML-RPC，在本章后面我们将看到，这对于 XML-RPC 演化为 SOAP 来说，非常重要。

针对 Web 服务的 XML-RPC 请求的结构类似于 HTTP/THML 的 POST 命令。请求的实体不是使用 HTML 的表单元素，而是使用 XML 编码的数据，如图 7-3 所示。

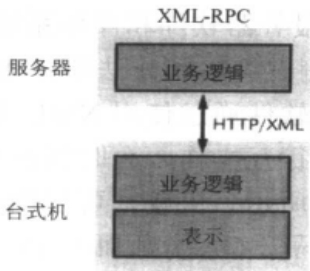

图 7-3 使用基础架构作为 HTTP/HTML 请求，XML-RPC 将采用 XML 编码的数据发送给服务

一个简单的 XML-RPC 服务安装在公共互联网 examples.pushtotest.com 上。这个服务从一个 XML 文档中获得两个参数，并将这两个参数值返回到一个 XML 响应文档中。使用一个计算机网络监视工具来演示数据在 XML-RPC 调用中的发送和接收(本章后面提供常用的网络监测器的 URL)。下面的内容是发送到这个服务的请求：

```
POST./.HTTP/1.1
Content-Length:.240
Content-Type:.text/xml
User-Agent:.Javal.4.0
Host: mayo.pushtotest.com:91
Accept:  text/html,.image/gif,.image/jpeg,.*;.q=.2,.*/*;q=.2
Connection:.keep-alive

<?xml.version="1.0".encoding="ISO-8859-1"?>
   <methodCall>
      <methodName>echo</methodName>
     <params>
        <param>
           <value>Tell.me.what. I.would.like.to.hear.</value>
        </param>
     <param>
        <value>Whisper.sweet.nothings.in.my.ear.</value>
     </param>
   </params>
</methedCall>
```

为使请求数据更加清晰易读，我改变了请求数据的间隔。在真实的请求中，仅使用回车字符隔离所有的 XML 和 HTTP 头数据。除此之外，这就是全部请求。它类似于 HTTP/HTML 请求，因为 XML-RPC 是在拥有同样传输协议和路由网络的 Web 基础架构上运行的。XML-RPC 请求要求业务逻辑存在于在连接的两端上运行的软件中。使用 JavaScript、 Applets 或 ActiveX 对象可以在浏览器中实现业务逻辑，使用某个 XML-RPC 库也可以在桌面应用程序中实现业务逻辑。

XML-RPC 响应在外观上也类似于 HTTP/HTML 响应，不过它还包含 XML 数据，如下所示：

```
HTTP/1.1.200.OK
Server:.Apache.XML-RPC.1.0
Connection:.Keep-Alive
Content-Type:.text/xml
Content-Length:.247

<?xml.version="1.0".encoding="ISO-8859-1"?>
<methodResponse>
      <params>
         <param>
            <value>
               <array>
                  <data>
                     <value>Tell.me.what. I.would.like.to.\
                     hear.</value>
                     <value>Whisper.sweet.nothings.in.  \
```

```
                        my.ear.</value>
                    </data>
                </array>
            </value>
        </param>
    </params>
</methodResponse>
```

这个响应使用了标准 ISO-8859-1 编码来处理字符串中的 Unicode 字符。响应中有一个包含了两个参数(在请求里发送的)的数组。因为使用的是 XML，所以响应数据不但能够包含简单的数据类型，也可以包含复杂的数据类型。关于 ISO-8859-1 编码的一个非常有用的指南可以在 http://www.cs.tut.fi/~jkorpela/chars.html 上找到。

观察 HTTP 编码协议的数据在浏览器与服务器之间的网络连接中的移动，可以帮助创建和调试智能测试代理。Windows NT、2000 和 XP 用户可以使用 Microsoft Network Monitor 工具来观察网络信息流通量。这些工具捕获数据包并显示它们，以显示 GET 和 POST 命令以及与它们相关联的数据值。Linux 用户可以使用流行的 tcpdump(http://www.tcpdump.org)、ethereal(http://www.ethereal.com/)和 NetWatch(http://www.slctech.org/~mackay/netwatch.html)utility 来观察网络信息流通量。

可以直接使用 XML-RPC 协议来编写 TestMaker 测试代理脚本，以请求某种功能。这里给出了完整的脚本，并且后面给出了它的工作原理。本书中的所有代码可以通过 http://www.pushtotest.com/docs 下载。

```
# Agent name: XMLRPC_Connect.a
# Author:  fcohen@pushtotest.com

thehost = "examples.pushtotest.com"
theport = 91
request1 = "Tell me what I would like to hear."
request2 = "Whisper sweet nothings in my ear."

print  "Agent running: XMLRPC_Connect"
print  "Description:  This agent will connect to a demonstra-
tion XML-RPC host"
print  "on the PushToTest server.  The host will echo back
whatever the agent sends."

print
print "Sending these values to the XML-RPC host on:  ",  the-
host,  "pert:  ",  thepert
print "request1 =  ",  request1
print "request2 =  ",  request2
print

# Import needed TestMaker,  TOOL and Jython libraries
from org.apache.xmlrpc import XmlRpcClient
from com.pushtetest.tool.protocolhandler import PretecolHan-
dler, Header,  Body,  XMLRPCProteco1,  XMLRPCBody,  XMLRPCHeader
from com.pushtetest.toel.respense import Response

# First we get an XMLRPC protocol handler object
```

```
protocol = PretecolHandler.getProtocel("xmlrpc")

# Then we put together the request document
body = XMLRPCBedy()

body.setTarget( thehost )
body.setTransport("http")
body.setMethod("echo")
body.addParameter( requestl )
body.addParameter( request2 )
protocol.setBody ( body )

protocol.setHost ( thehost )
protocol.setPath ( "" )
protocol.setPort ( theport )

print "Connecting..."

response = protocol.connect()

print "Here is the response from the host:"
print response

print
print "That request took ", response.getTotalTime(), "mil-
liseconds to complete."
print
print "Agent ended."
```

如果您理解了第 5 章的示例代理，测试 XML-RPC 服务看起来类似于测试 HTTP/HTML 应用。下面详细描述这个测试代理脚本。

```
from org.apache.xmlrpc import XmlRpcClient
from com.pushtotest.tool.protocolhandler import ProtocolHan-
dler, Header, Body, XMLRPCProtocol, XMLRPCBody, \
XMLRPCHeader
from com.pushtotest.tool.response import Response
```

这个测试代理首先导入需要的 TestMaker 对象，使用 XML-RPC 协议与 Web 服务进行通信。

```
protocol = ProtocolHandler.getProtocol("xmlrpc")
```

这个测试代理使用了一个 XML-RPC 协议处理程序和体对象。这些对象通过使用 HTTP 协议、发送和接收 XML 编码数据，来处理与 Web 服务间的实际会话。

```
body.setTarget( thehost )
body.setTransport("http")
body.setMethod("echo")
body.addParameter( requestl )
body.addParameter( request2 )
```

体对象确定 Web 服务的 URL 和调用方法的名称。向体中添加参数，能够使得 TestMaker 协议处理程序将 request1 和 request2 字符串编码到一个 XML 请求文档中，这个文档将被发送给 Web 服务以进行处理。

```
response = protocol.connect()
```

response对象包含来自Web服务的XML响应。将response对象打印到输出窗口来显示XML数据。TestMaker包括XML处理库，使得它很容易使用XML响应文档来验证Web服务响应。本章后面将详细介绍响应的验证。有关如何在服务器上构造Web服务的一个示例请访问http://www.pushtotest.com/ptt/kits/index.html。

7.3 XML-RPC的缺点

XML-RPC的简单性掩盖了一个重大的体系结构问题：XML-RPC系统的互操作性相当脆弱。考虑下面这些问题：

- 在XML-RPC请求或响应中，XML-RPC没有定义清晰的机制来传递XML文档本身。既然请求和响应本身就是XML文档，那么将XML编码的数据发送给服务以进行处理，这时会出现什么情况呢？
- 无法有规划性地改变请求或响应文档类型的格式。通常只在维护软件时改变参数和方法。在客户端或服务器的任何方法的变化都需要在客户端和服务器进行认可。这使得XML-RPC很脆弱。
- 怎样扩展XML-RPC以支持安全性、加密、数据压缩和其他特殊需要？目前XML-RPC只覆盖基本的HTTP/XML协议。
- XML-RPC不支持命名空间。每个域名都要惟一，否则XML-RPC客户端或服务器会混淆。通常对于只有几个参数的请求来说这不成问题，但如果在1000个响应里保证每个元素都惟一，是非常困难的事情。
- XML-RPC在HTTP协议之上工作。一些创新的基于Web服务的系统设计通过消息队列、数据库队列甚至邮件来传递RPC调用。

一度警惕的软件行业已经把XML-RPC当作一项新的发明：SOAP。

7.4 SOAP的全局互操作性

Userland软件是XML-RPC的早期元部件，并创建了XML-RPC作为标准。XML-RPC易学易用，它基于现有的Web基础架构，运行成本低。在Web使能应用程序发生变化和需要维护时，XML-RPC变得很脆弱。Userland使Microsoft公司、IBM公司以及后来的Sun公司创建新的XML-RPC互操作标准以解决可扩展性和互操作性问题。工作的成果就是：SOAP。

SOAP协议使用多步过程完成一个事务，如图7-4所示。在一个SOAP事务中，软件应用程序的业务规则向远程服务器上的SOAP方法发出一个请求。这个请求包含的参数通过一系列的串行器将本地语言(如Java、C、C++和Visual Basic)转化为一个字符串，串行化的参数成为XML文档的一部分，并且符合SOAP封装规范。一个SOAP封装包含一个头和体。体中包含XML编码的串行化参数。通常使用HTTP或SMTP协议将SOAP封装传送给Web服务。其他的传送方式(如RMI和IIOP)也是可用的，但是现今已很少使用。

图 7-4　SOAP 请求开始于 Web 使能应用程序的业务规则，并使用技术组合将请求编码到 XML 数据中，然后通过 Web 基础架构发送

在 SOAP 诞生不久后，Microsoft 和 IBM 公司首领 WSDL 为标准元语言，用于描述基于 SOAP 的 Web 服务。WSDL 完成方法、参数和数据类型的描述，用于制定一个 SOAP 请求和接收一个响应。WSDL 是机器可分析的，能使开发工具和应用服务器产生程序源代码。

SOAP 提供更好的可扩展性并通过引入图 7-2 中描述的 SOAP 堆栈层来减少 XML-RPC 的脆弱性。然而，因为拥有更高的灵活性，很可能出现不兼容性和可扩展性问题。

7.5　Web 服务可扩展性技术

SOAP 和基于 WSDL 的 Web 服务使用多步过程来完成一个事务。很多技术和系统体系结构试图提高 Web 服务的可扩展性和性能。理解这些技术对于验证测试中的结果是很重要的。

Web 服务请求通常开始于应用程序的业务逻辑，它从一个 WSDL 文档中获取要调用的方法和参数。例如，下面是一个公用 Web 服务的部分 WSDL 文档，给定一个美国邮政编码，它返回该邮政编码对应地区的天气。

```
<message name = "getTempRequest">
 <part name = "zipcode" type = "xsd:string"/>
</message>
<message name = "getTempResponse">
 <part name = "return" type = "xsd:float"/>
</message>
```

天气服务通过传入邮政编码字符串值来请求调用 getTempRequest 方法，并接收温度的浮点值作为响应。

因为 WSDL 很少变化，所以很多开发人员将 WSDL 定义嵌入到他们的代码中，以避免每次获取 WSDL 的开销。这样能提高性能并解决脆弱性问题，但是维护开销可能会变大，因为 WSDL 可能发生变化。避免维护问题的更好方法是在中心数据库中缓存 WSDL，周期性地检查 WSDL 的时间戳和版本号，以判定是否有新的可用的 WSDL。

软件开发人员提高性能的另一种方法是对 XML 不进行验证。对于不使用验证的系统，测试组件应该使用响应结果的轻型验证。例如，下面的 WSDL 定义了响应的模式。

```
<element name="zipcode"  type="int"/>
<element name="temperature" type="float"/>
<element name="remarks"  type="string"/>
```

调用此服务的结果类似于：

```
<zipcode>95008</zipcode>
<temperature>65 F</temperature>
<remarks>Storm warning</remarks>
```

如果温度值不是浮点类型，响应就会抛出异常。它事实上是一个字符串。在智能测试代理中验证响应通常比依赖于 DTD 或 XML 模式代码来验证 SOAP 响应要快很多。

SOAP 中的参数类型也可能存在可扩展性问题。SOAP 定义了简单数据类型：字符串、整数、浮点数和负整数。在 WSDL 中出现的简单数据类型(例如字符串)使用了如下所示的 XML 模式：

```
<message name =  "getTempRequest">
  <part name =  "zipcode"  type =  "xsd:string"/>
</message>
```

本章后面会提到，SOAP 请求和响应可能包括重要的新数据类型。假设温度 Web 服务需要检索图。此调用的模式可能如下：

```
<message name =  "getTempRequest">
  <part name =  "zipcode"  type =  "xsd:string"/>
</message>
<message name =  "getTempResponse">
  <part name =  "return"  type =  "xsd:float"/>
  <part name =  "map"  type =  "xsd:http://www.pushtotest.com/
wsdl/mapformat"/>
</message>
```

当读到响应时，XML 验证分析器会通知 pushtotest.com 主机来获取 XML 模式的图格式定义。当验证分析器不能缓冲模式定义时，请求的开销会导致可扩展性问题。

一般的性能规则都使用简单的 SOAP 数据类型，除非是不得不使用其他数据类型。每个新的数据类型引入串行化器以将 XML 值转化为本地程序语言(Java、C、C++和 Visual Basic)的值并返回。串行化器会导致性能问题或许多 bug。例如 Apache 和 Microsoft SOAP 实现都包含一个 BigDecimal 数据类型。但是，在 Java 1.3 和.NET 1.2 之前二者是不兼容的。

虽然 SOAP 被设计为在现有的 Web 使能应用程序环境中工作，但该协议会引入防火墙和路由器问题。不像通常使用 HTTP 的 Web 服务器，SOAP 消息使用 HTTP 作为传输，这与 HTTP 的表格提交是等价的。与 HTTP 的 GET 和 POST 相比，调用需移动更多的数据。这会影响网络的性能。需要保证对防火墙和路由器进行专门测试。例如，要谨慎检查防火墙的安全策略以确保它不会将 SOAP 请求作为 Web 信息流通量来监视。否则，防火墙会分流信息量，这就像 Denial of Service(DOS)攻击。

早期的 Web 服务非常简单：进行 SOAP 调用并获取响应。更高级的 SOAP 应用程序发出一系列请求和响应调用直到事务完成。事务性 SOAP 调用需要识别会话并缓冲会话的状态。SOAP 事务的缓冲机制存在潜在的可扩展性问题。

7.6 Web服务互操作性问题

步入 Web 服务的新领域，您会觉得问题就在附近。Web 服务真正到来了。很多平台提供者、独立的软件提供商和工具软件开发人员已经在其产品中实现了 SOAP 和 WSDL。开发人员不得不多次解释说明书的各个部分。解释使得互操作性问题深入到基于 SOAP 的 Web 服务中。

Web 服务的互操作性目标是提供软件应用程序之间的无缝和自动连接。SOAP、WSDL 和 UDDI 协议定义了一个自描述方法来发现和调用软件应用程序的方法，而不管平台的具体位置。数据被编组到 XML 请求和响应文档中，在软件包间使用 HTTP 或基于消息的协议移动数据。互操作性问题，例如 BigDecimal 中的专用平台的差异，常常出现在发现、定义和请求/响应机制中。

7.6.1 发现

在理想的 Web 服务世界里，每个软件应用程序都使用一个自发现和自分类方法进行编码。缺少必需的函数的软件查询服务的一个基于 UDDI 的注册，自动联系一个找到的 Web 服务来处理任务。一旦发现 Web 服务函数，WSDL 和 SOAP 就允许通信。问题是要对可能发现的函数按照其功能进行分类。UDDI 定义了多个 TModel，分别描述函数的位置、路径和特性。

UDDI 使得业务能够实现 Web 服务的在线注册。在公共互联网上，Microsoft、HP 和 IBM 提供了业务所需要的 UDDI 注册表。业务使用 UDDI 的 TModel 系统来分类 Web 服务。于是存在下面的问题：UDDI 允许多重分类并自己管辖注册中错误的入口。例如，假定 Web 服务在一个使用了 SIC 码的 UDDI 注册表中打印和发送货物单，但它不列出其地理信息。使用这样的 Web 服务在地球的另一边可以工作，但如果自己做标记的话会更容易些。

按照传统分类的提供者，包括 LCSH(Library of Congress Subject Heading)、 FAST(Faceted LCSH)、 DDC(Dewey Decimal Classification)和 LCC(Library of Congress Classification)都很好地使用和理解了 UDDI。直到分类专家增加开发和维护 UDDI 中公共目录结构的实用知识时，才需要考虑互操作性问题。

另一方面，私有的 UDDI 目录也是可行的。某些企业已经花费了数十亿美元来革新它们的供应经济学政策(supply-side)系统。随着这些革新，产品定义和业务过程标准也随之产生。这些过程可以很容易地被移动到一个 Web 服务的 UDDI 注册表中，从而从 UDDI 支持的 Web 服务中访问它们。

7.6.2 定义

Web 服务使用 WSDL 来定义如何向一个基于 SOAP 的方法发出请求。WSDL 假设公司间协作定义用户数据类型。这样的协作通过建立互操作测试组件的协作组织进行测试。SOAPBuilder 与 Web 服务厂商间存在宽松的关系，它的目的是为了证明互操作性。SOAPBuilder 也发布互操作性的测试组件以检查 SOAP 实现，网址为 http://www.xmethods.com/ilab/和 http://www.whitemesa.com/interop.htm。测试组件始于 SOAP 接口的 WSDL 定义。它们测试请求和响应文档的内容以验证数据。

这样就为 WSDL 注入了新的活力。新的技术如 Cape Clear CapeStudio 和 BEA WebLogic Workshop，能够自动开发 WSDL 文档，供基于 SOAP 的 Web 服务使用。这样的工具使得开发

人员所编写的 WSDL 文档具有良好的结构。另外，Java 在 Java Web 服务开发包里也能更好地处理 WSDL。详细情况请访问 http://java.sun.com/webservices/。

```
<?xml version="1.0" ?>
<definitions name="PushToTestService" targetNamespace
="http://www.pushtotest.com/pushtotestservice.wsdl">
<message name="testRequest">
   <part name="userName" type="xsd:string" />
   <part name="authenticationToken" type="xsd:string" />
   <part name="goodUnitl"  type="xsd:Date"  />
   </message>
...
```

WSDL 文档会产生互操作性问题。例如，考虑上面供测试的 Web 服务的 WSDL 代码片段。该 WSDL 定义了怎样发送 testRequest 命令。但是<definitions>元素没定义命名空间。正确的<definitions>元素应该如下：

```
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/"  targetNamespace="http://tem-
puri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/">
```

开发人员可能认为 Web 服务在默认情况下将使用 W3 SOAP 命名空间标准。对于元数据类型如字符串，这也许能够工作，但对于 Date 数据类型可能会存在互操作性问题，在本章后面可以看到这一点。如果没有指定命名空间，Web 服务就不能正确地处理数据类型。

7.6.3 请求和响应

SOAP 为多个软件应用程序定义了互相调用对方的方法并传递数据的标准方式。SOAP 请求是包含了命名空间、方法调用和数据描述的 XML 文档。XML 尽可能地灵活，以允许开发人员编写 XML 元素和定义。灵活性也带来了互操作性问题。

例如，一个典型的 SOAP 响应文档可能如下所示：

```
<?xml version="1.0"  encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/
XMLSchema" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<SOAP-ENV:Body>
<nsl:echoStringResponse xmlns:nsl="http://soapinterop.org/">
<result xsi:type="xsd:string">Hello!</result>
</nsl:echoStringResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

这个响应文档发送一个包含“Hello”的文本字符串。<result>元素也包括 xsi:type=“xsd:string”参数，它被反串行化为一个本地语言对象(例如，Java 字符串对象)。很多 SOAP 工具在请求和响应文档中增加了显式的类型信息。另一方面，某些 SOAP 库返回的响应不包含任何类型信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/
XMLSchema" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<SOAP-ENV:Body>
<nsl:echoStringResponsexmlns:nsl="http://soapinterop.org/">
<result>Hello, I'm a string!</result>
</nsl:echoStringResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

在上面的响应文档中，<result>元素包含无类型的信息。这种情况下，反串行化<result>值的 SOAP 库必须查询服务的 WSDL 描述，以发现返回类型的描述。如果 WSDL 没有定义响应类型，就应该抛出异常。但是据我的经验，很多 SOAP 库在默认情况下的 String 对象都是不正确的。

数据类型与 Web 服务的关系类似于橡胶与公路的关系。SOAP 使用串行化器对象和反串行化器对象将软件应用程序的本地语言转换为多个 SOAP 协议，这些 SOAP 协议在网线中移动请求。因此本地语言的数据存在依赖性问题。例如，Java 定义的 Date 对象不同于 Microsoft .NET C++的数据对象。对于同样的 SOAP 数据类型名称，存在不同的实现。这样就会产生互操作性问题。

通常不满足互操作测试的数据类型是浮点数和日期。

1. 浮点类型和十进制数据类型

SOAP 中的浮点数被表示为十进制数字的字符串。SOAP 定义的浮点数也支持科学计数法，例如 7.53E+10，用于处理几十年来工程师使用符号来表示的指数数字。一般情况下科学计数法很有效，但是有时浮点数也存在一些问题。

例如，原始 IBM SOAP4J 实现(现在为 Apache SOAP 和 Apache AXIS 库)使用 Java 的 toString 方法和构造函数将 SOAP 文档中的浮点值转换为 Java 对象。如果串行化浮点数“infinity”，Java 输出的字符串就为“Infinity”。另一方面，XML 模式将 infinity 串行化为“INF”。这样会使 SOAP4J 和其他 SOAP 工具包存在互操作性问题。

正如互联网诞生于网络管理员之间的合作，今天我们看到 SOAP 实现合作解决了互操作性问题。Apache SOAP，SOAP4J 的后继，可以接受“INF”作为有效的方式来串行化 infinity。

当太长时，十进制数据类型受害于语言的相关性。十进制数据类型可以表示 40 位精度的大数。在 SOAP 的请求或响应中依赖于 40 位的精度会产生问题，除非在服务器和客户机上使用相同的语言实现。对于日期中秒数的小数部分和尾部是零的十进制数也是如此。

BigDecimal 是解决语言依赖性缺陷的很好的示例。十进制数值是银行应用程序中的费用计算的必须部分，它需要大数。十进制数据类型的 XML 模式规范允许任意的精度。十进制数据

类型可以表示 1000 位的精度——也就是说它能表示包含 1000 个数字的字符串。Apache SOAP 是基于 Java 实现的 BigDecimal 数据类型。Java 的 BigDecimal 的数值精度具有上限，这依赖于底层操作系统(Solaris, Windows 等)。

XML 模式通过定义一个最小化的符合的实现规范来解决这些互操作性问题。在十进制数据类型的情况下，XML 模式需要至少 18 位的精度。Apache 和 Java 满足此需求。但是这并不意味着基于 SOAP 的 Web 服务也接受这个最小精度。

Microsoft .NET 实现处理 BigDecimal 数据类型可精确到 29 位数字。因此，如果 Apache SOAP 请求的 BigDecimal 数据类型接收.NET 响应，多余的精度位如何处理？不过，本地的 SOAP 串行化器和反串行化器的实现知道如何处理。关键是 SOAP 事务是有效的，但数据是错误的。开发人员需要充分警惕，要把数据测试和保护编写到软件应用程序中，以处理 SOAP 交换中的失效数据。

如果有充足的时间和精力，这种互操作性问题都可以由 Web 服务平台提供者解决。例如，Sun 更新的 Java 提供了与.NET 平台间的 BigDecimal 互操作性问题。从我们要着手工作的本性来看，软件开发人员、QA 技术人员和 IT 管理员都在第一线警惕着克服这种问题。

2. 日期数据类型

相对于浮点十进制数据类型，日期数据类型的互操作性问题更大。XML 模式定义了 dataTime 数据类型，它包含世纪、年、月、日、小时、分钟和秒。对于毫秒、微妙和更小的时间单位如何处理？在我们的日常生活中，一个 2GHz 的 Intel CPU 的价格不到 150 美元，而用户需要的 Web 服务最多只能执行 2~3 秒。对于 Web 服务性能的度量，需要精确到毫秒。

XML 模式指定任何秒之后的数值都要编码为 dateTime 数据类型，然而并没有一个应用程序必须支持的最小数值。Apache SOAP 使用 Java 日期类(Java.util.Date)来串行化和反串行化 dateTime 数据类型。Java 日期支持的精度为毫秒而.NET 日期数据类型使用毫微秒值。因此.NET 中日期数据类型可精确到纳秒。

7.6.4 展望

今天，Web 服务由 UDDI 核、WSDL 和 SOAP 协议来提供。在中间是定义工作流自动化的协议：Web 服务管理服务和垂直的市场协议。Web 服务帮助开发人员创建高度集成的解决方案。因此当工作流自动化 Web 服务和垂直的市场 Web 服务混合时，出现互操作性问题也不足为奇。

如果 Web 服务工具包通过改进以解决互操作性问题，那么相比于使用现有的标准(CORBA、DCOM 和 RMI)，客户、用户和业务将能更有效地解决系统集成问题。Web 服务工具包供应商正致力于解决互操作性问题。如果互操作性问题存在或更糟，我们就减慢采用并使用更专业的服务开销来实现集成系统。其间，Web 服务要先关闭再开始。

7.7 利用 TestMaker 来理解 WSDL

WSDL 是一个规范，它描述了基于 SOAP 的 Web 服务的相关参数、方法、数据类型和访问器。虽然没有 WSDL 也可以编写基于 SOAP 的 Web 服务，但是使用元语言将 SOAP 接口描述为一个方法也是可行的。

通常测试策略通过一个有效的检索来实现，该测试策略可以理解描述基于 SOAP 的 Web 服务的 WSDL 文档。现在，让我们看看 TestMaker 如何处理 WSDL 文档。下面是一个代理，它读取和分析运行在公共互联网 examples.pushtotest.com 上的基于 SOAP 的 Web 服务的 WSDL 定义。下面是这个代理的全部源代码，其后解释了它的工作原理。

```
# Agent name:  explore_wsdl.a
# Author:  fcohen@pushtotest.com

# ------------------------------------------------------
# Set parameters and variables for this agent

# Location of the WSDL for the responder Web Service
target_wsdl = \
"http://examples.pushtotest.com:92/axis/servlet/ \
AxisServlet/responder_rpc?wsdl"

# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.parser.wsdl import WSDLParser
from javax.wsdl import WSDLException

# Import useful Python and Java libraries
import sys
import java
from urlparse import urlparse
from java.util import Random

# Main body of agent

try:
    parser = WSDLParser( target_wsdl )
definition = parser.getDefinition()
except WSDLException,  ex:
    print "Something went wrong trying at:"
    print target_wsdl
    print
    print "The complete exception is:"
    print ex
    sys.exit(1)

print
print "Web Service Description"
print
print "At this location:"
print target_wsdl
services = definition.getServices()

for keys in services.keySet().iterator():

    current = services.get( keys )
    print "there is a Web Service named",  \
    curent.getQName().getLocalPart(),  "offering \
    these services:"
    print
```

```
        ports = current.getPorts()
        for pkeys in ports.keySet().iterator():
            pcurrent = ports.get( pkeys )
            bind = pcurrent.getBinding()
            print "Service:", pcurrent.getName()
            print "Binding:", bind.getQName()
            for extenelem in \
            pcurrent.getExtensibilityElements():
                print "Target URI:", extenelem.getLocationURI()

            print
            print "The",pcurrent.getName(),"service offers \
            these methods:"
            print

            for ops in bind.getBindingOperations():
                if ops.getName()!="class$":
                    print ops.getName(),  "which needs these \
                    parameters:"
                    print

                    mcur  =  \
                    ops.getOperation().getInput() .getMessage()
                    if mcur != None:

                        parts = mcur.getParts()
                        for partkeys in \
                        parts.keySet().iterator():
                            part = parts.get( partkeys )
                            print  " ",part.getName(),  "that \
                            is of type", part.getTypeName()
                            print
    print

print
print "Agent finished."
```

运行本代理能产生一个已格式化的报表，以描述基于 SOAP 的 Web 服务的 WSDL 文档所提供的信息。下面是代理的部分输出：

```
Web Service Description
At this location:
http://examples.pushtotest.com:92/axis/servlet/AxisServlet/
responder_rpc?wsdl
there is a Web Service named responder_rpcService offering
these services:
Service:  responder_rpc
Binding: http://examples.pushtotest.com:92/axis/servlet/
AxisServlet/responder_rpc:responder_rpcSoapBinding
Target URI: http://examples.pushtotest.com:92/axis/servlet/
AxisServlet/responder_rpc
The responder_rpc service offers these methods:
Callback which needs these parameters:
   host that is of type http://www.w3.org/2001/
XMLSchema:string
```

```
    times that is of type http://www.w3.org/2001/
XMLSchema:long
    path that is of type http://www.w3.org/2001/
XMLSchema:string
    method that is of type http://www.w3.org/2001/
XMLSchema:string
    target that is of type http://www.w3.org/2001/
XMLSchema:string
    port that is of type http://www.w3.org/2001/XMLSchema:int
Bomb which needs these parameters:
    delay that is of type http://www.w3.org/2001/
XMLSchema:long
Respond which needs these parameters:
    wordcount that is of type http://www.w3.org/2001/
XMLSchema:long
    delay that is of type http://www.w3.org/2001/
XMLSchema:long
snooze which needs these parameters:
    thesecs that is of type http://www.w3.org/2001/
XMLSchema:long
Agent finished.
```

使用从 WSDL 文档获取的信息，我们能创建一个并发的测试代理，它调用一个 SOAP 方法。例如，我们通过传递 wordcount 和 delay 值来调用 respond 方法。

为了理解 WSDL，使用浏览器来获取一个示例 Web 服务的 WSDL，其网址为 http://examples.pushtotest.com:92/axis/servlet/AxisServlet/responder_rpc?wsdl。这是一个公共可用的基于 SOAP 的 Web 服务，驻留在 PushToTest.com 中。下面看一下测试代理是如何学习 WSDL 定义的。

```
# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.parser.wsdl import WSDLParser
from javax.wsdl import WSDLException
```

这个代理首先使用一个 import 语句，来标识 TestMaker 的 TOOL 中的 WSDLParser 对象。有关 TOOL 的详情请访问网址 http://docs.pushtotest.com。WSDLParser 提供了几种方法来发现 WSDL 文档的内容。

```
services = definition.getServices()
for keys in services.keySet().iterator():
    current = services.get( keys )
    print "there is a Web Service named", cur-
rent.getQName().getLocalPart(), "offering these services:"
    print
```

definition.getServices()方法返回一个服务对象，该服务对象包含了在 WSDL 文档中描述的已定义的基于 SOAP 的 Web 服务。很多 WSDL 文档定义单一的 Web 服务。但是 WSDL 规范允许描述多种服务。代理使用一个 for 循环来处理多个服务。

```
ports = current.getPorts()
   for pkeys in ports.keySet().iterator():
       pcurrent = ports.get( pkeys )
       bind = pcurrent.getBinding()
       print "Service:", pcurrent.getName()
       print "Binding:", bind.getQName()
```

然后，代理找到每个服务的端口。WSDL 端口描述了 Web 服务的传输层信息。在这种情况下，getBinding()方法返回一个 URL，这个 URL 指向将代理绑定到 Web 服务所需要的多个方法。响应器是一个简单的 RPC 样式的服务，因此绑定是一个供调用的简单的 URL。您可以使用绑定信息来显示 Applet、客户端软件，甚至是调用 Web 服务的安全证书。

```
for ops in bind.getBindingOperations():
    if ops.getName()!="class$":
        print ops.getName(),  "which needs these \
        parameters:"

        mcur  =  \
        ops.getOperation().getInput()  .getMessage()
        if mcur  != None:
            parts = mcur.getParts()
            for partkeys in \
            parts.keySet().iterator():
                part = parts.get( partkeys )
                print  " " ,part.getName(),  "  \
                type", part.getTypeName()
```

下面我们使用 for 循环来处理 Web 服务中的参数和方法。

TestMaker 自带的新建代理向导自动编写测试代理脚本的框架。对于测试基于 SOAP/WSDL 的 Web 服务的代理，新建代理向导通过给定的 URL 读取 WSDL 文档，创建 Jython 脚本，该脚本调用处理 SOAP 服务所必需的 TOOL 命令。框架是功能性的，用户可以在它的基础上添加额外的测试逻辑。详情请参考 TestMaker 文档。

如例中所示，TestMaker 提供了理解已发布的 WSDL 文档内容的工具。从 WSDL 中我们可以了解可用的服务和可调用的方法。下面研究如何使用 SOAP 发送请求和接收响应。

7.7.1 构造 SOAP 调用

SOAP 是建立互操作性系统的规范。SOAP 规范覆盖了不同类型的请求和响应、不同样式的参数编码以及各种服务体系结构。您需要很好地掌握 SOAP 样式以有效地测试基于 SOAP 的 Web 服务的可扩展性、功能和性能。本章的其余部分详细阐述了很多 SOAP 样式，并提供一个 TestMaker 测试代理作为示例。

不同类型的 SOAP 参数编码

SOAP 被设计为能够跨平台、语言和系统进行互操作。SOAP 实现了提交接口，用于向一个服务发送请求，并向其传递多个参数，这些参数包含字符、数值、日期和复杂数据类型。本章的前面我们探讨了所研究的底层平台存在的互操作性问题。在本节我们将看到 SOAP 所支持的不同数据类型，以及软件开发人员如何扩展基本数据类型，使其成为 SOAP 调用中的复杂数据类型。

SOAP 定义了所有实现必须支持的基本数据类型集。这些基本数据类型包括 string、boolean、decimal、float、double、datetime、binary 和 URIs。每个 SOAP 实现负责基本数据类型的实现。例如，ASP.NET 提供了 SOAP 数据类型，并可以把它映射为表 7-1 中所示的 XML 的数据类型。

表 7-1 映射 SOAP 参数编码

ASP. NET 数据类型	Java 数据类型	XML 数据类型
string	String	string
boolean	Boolean	boolean
float(single)	Float	Float
double	Double	double
decimal	Decimal	decimal
long	Long	Long
int	Int	Int
short	Short	short
byte	Byte	unsignedByte
ulong	Ulong	unsignedLong
uint	Uint	unsignedInt
ushort	String	unsignedShort
sbyte	Boolean	byte
DateTime	Float	date
DateTime	Double	Time
DateTime/Date	Decimal	TimeInstant
String[]	Long	ArrayOfString
boolean[]	Int	ArrayOfBoolean
float[]	Short	ArrayOfFloat
double[]	Byte	ArrayOfDouble
decimal[]	Ulong	ArrayOfDecimal
long[]	Uint	ArrayOfLong
int[]	String	ArrayOfInt
short[]	Boolean	ArrayOfShort

在 TestMaker 中，您可以使用下面的格式来传递基本数据类型：

```
body.addparameter( "wordcount", java.lang. Long, 1839, None)
```

addParameter 方法使用了内置的串行化器，将 long 数据值 1839 转换为一个 XML 表示，并将其发送给服务。这很容易实现，因为所有的底层 Web 服务都支持编组(marshaling)、编码和传送基本数据类型。

很多时候事务需要使用复杂数据类型来进行通信。图 7-5 描述了复杂数据类型。本书中将结构和组合数据类型作为复杂数据类型。

图 7-5 SOAP 允许在请求和响应中使用复杂的数据类型

SOAP 和 TestMaker 允许在请求和响应中定制数据类型。

```
ef = employeeFile.getFile( 38183 )
body.addparameter("EmployeeFile", com.pushtotest.employeeFile,ef, None)
```

这种情况下，对 SOAP 服务的请求包含 employeeFile 对象的串行化版本，该对象由变量 ef 引用。TestMaker 搜索已注册的串行化器对象，它将 com.pushtotest.employeeFile 对象转化为 XML 表示。为了使用 TestMaker 来注册新的串行化器，必须编写 JavaBean 对象来响应特定数据类型的串行化和反串行化命令。

7.7.2 不同类型的 SOAP 调用

SOAP 在数据类型方面对 XML 规范起着杠杆作用，它使用串行化器对象和反串行化器对象来编组和解组平台间的数据。SOAP 也支持几种不同样式的请求和响应，以提供更多的灵活性。SOAP 支持 RPG 请求和文档样式的请求。

1. RPC SOAP

RPC 样式的 SOAP 调用将一个简单的请求发送到远程服务器上某个对象的特定方法。RPC 请求是与 Java 和 C++编程模型最近的模型。例如，考虑下面的对象：

```
public class responder_msg
{
    public String Respond( long wordcount,  long delay )
    {
      ...
    }
}
```

responder_msg 是运行在远程服务器上的 Java 对象，它有一个简单的方法 Respond，该方法包含两个 long 数据类型的参数。目前这个 responder_msg 对象运行在 examples.pushtotest.com 上，该 Web 服务在公共互联网 PushToTest.com 上。下面，我们介绍一个 TestMaker 测试代理，它使用 SOAP 请求和 responder_msg 对象进行互操作。

```
# test_rpc.a
# Author:  fcohen@pushtotest.com

# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  SOAPProtocol,  SOAPBody,  SOAPHeader
```

```
from com.pushtotest.tool.response import Response
from java.lang import Long

# First we set-up the basic information describing
# the name,  location and path to the Web Service host.

host = "examples.pushtotest.com"
port = 92
path = "axis/servlet/AxisServlet"
endpoint = host + ":" + str( port )  + "/" + path

print "SOAP service is at ",endpoint

protocol = ProtocolHandler.getProtocol("soap")
body = SOAPBody()
protocol.setBody(body)

# Set the endpoint values

protocol.setHost( host )
protocol.setPath( path )
protocol.setPort( port )

body.setTarget(  "responder_rpc"  )
body.setMethod(  "Respond"  )

# Responder takes these parameters:
# wordcount = the number of jibberish words to return
# in the response XML document. Each word is approximately
# 5-8 characters long and each response is randomly
# unique.
# delay =  (optional)  the number of milliseconds
#          the Responder service will wait before
#          responding to the request.
# bomb =  (optional)  the number of milliseconds the
#         Responder service will wait until throwing
#         an exception, which is returned as a 500 Servlet
#         Error and XML fault.
# callback =  (optional)  see the test_callback.a agent
#             for details

# For this test agent we want 150 jibberish words with
# a 100 millisecond delay

body.addParameter(  "wordcount",  Long,  150, None )
body.addParameter(  "delay", Long,  100, None )
print "Sending request to server..."

response = protocol.connect()

print
print "Here is the response:"
```

```
print response

print
print "Agent done."
```

当运行这个代理时，TestMaker 将一个 RPC 样式的 SOAP 请求发送给此服务。TestMaker 输出窗口显示了响应文档。下面查看此代理来观察 RPC 样式的请求是如何构造的。

```
# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import ProtocolHan-
dler, SOAPProtocol, SOAPBody, SOAPHeader
from com.pushtotest.tool.response import Response
from java.lang import Long
```

Import 命令通知 TestMaker 使用 TestMaker 的 TOOL 中的 SOAPProtocol 对象。通常，我们不必导入基本数据类型，如 java.lang.Long，因为 TestMaker 脚本语言提供了它自己的 long 数据类型。在这种情况下，我们需要在请求的参数中发送一个 long 数据类型。

```
host = "examples.pushtotest.com"
port = 92
path = "axis/servlet/AxisServlet"
```

这些变量定义了目标服务器以及 Web 服务的位置。

访问 http://examples.pushtotest.com:92/axis/servlet/AxisServlet/responder_rpc?wsdl，可以获取 WSDL 文档中关于 responder_rpc 服务的详细内容。

```
protocol = ProtocolHandler.getProtocol("soap")
body = SOAPBody()
protocol.setBody(body)
```

我们向 TOOL 申请一个 SOAPProtocol 处理程序对象和一个 SOAPBody 对象以构造请求。SOAPBody 包含了请求的参数和请求的目标。

```
body.setTarget( "responder_rpc" )
body.setMethod( "Respond" )
```

目标值定义了包含了将要访问的对象和方法的 Web 服务。我们也可以从 WSDL 文档中获取这种 Web 服务。

```
body.addParameter( "wordcount", Long, 150, None )
body.addParameter( "delay", Long, 100, None )
```

Responder 方法包含多个参数，wordcount 参数是从响应的 XML 文档中返回的乱语单词(gibberish word)数目。每个单词大约 5~8 个字符长，每个响应都是随机惟一的；delay 是可选参数，它包含了在响应请求前 Responder 服务要等待的毫秒数；bomb 是可选参数，它表示在抛出异常前 Responder 服务要一直等待的毫秒数，所抛出的异常为 500 Servlet Error 和 XML 错误。

```
response = protocol.connect()
print response
```

最后，准备发送请求。connect()方法发送请求并返回一个 response 对象。

这个简单的代理发出一个单一的 SOAP 请求，并返回 SOAP 响应。根据这个简单的代理，我们可以编写一个完成如下任务的代理：

- 发出不同大小和延迟的多个请求
- 操作多个并发线程的请求脚本以增加服务器的负载
- 在多个机器上运行多线程代理的多个副本
- 为代理添加简单的日志功能，用于确定 TPS 和 SPI 值

2. 文档样式的 SOAP 消息

当应用程序在一种对象格式中具有 XML 数据，而且并不需要将该对象数据转换到个体参数时，文档样式的 SOAP 请求可以很好的工作，如前面一节中的 SOAP RPC 示例。

examples.pushtotest.com 上运行了一个文档样式的基于 SOAP 的 Web 服务，这个服务位于公共互联网 PushToTest.com 上。该服务使用与 RPC 样式的 SOAP 请求示例相同的参数。文档样式的 SOAP 请求不采用单个参数，而把参数作为 DOM 树的元素。

我们创建一个测试代理，该代理需要 75 个乱语单词，有 25 毫秒的 Web 服务延迟。为此需要创建一个发送给服务器的 XML 请求文档。完成后的文档如下所示：

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <Respond xmlns="urn:responder_msg">
      <delay xmlns="urn:responder_msg">25</delay>
      <length xmlns="urn:responder_msg">75</length>
    </Respond>
  </s:Body>
</s:Envelope>
```

下面是测试代理的完整代码，其后是一些关于构造请求和接收响应的所需步骤的讨论。

```
# test_rpc.a
# Author:  fcohen@pushtotest.com

# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  SOAPProtocol,  SOAPBody,  SOAPHeader
from com.pushtotest.tool.response import Response
from java.lang import Long

# This agent also uses JDOM APIs to handle XML data
from org.jdom import Document,  Element, JDOMException,  \
Namespace, DocType
from org.jdom.output import DOMOutputter, XMLOutputter

# First we set-up the basic information describing
# the name,  location and path to the Web Service host.

host = "examples.pushtotest.com"
```

```
port = 92
path = "axis/services/responder_msg"
endpoint = host + ":" + str( port )  + "/" + path
print "SOAP service is at ",endpoint

protocol = ProtocolHandler.getProtocol("soap")
body = SOAPBody()
protocol.setBody(body)

# Set the endpoint values

protocol.setHost( host )
protocol.setPath( path )
protocol.setPort( port )

body.setMethod(  "Respond"  )

xmlnsl = "urn:responder_msg"

# Create the request document by first creating the Respond
element.
elOne= Element(  "Responder", xmlnsl)

# Add child elements for <delay>...
elDelay = Element(  "delay", xmlnsl )
elDelay.addContent( "12"  )
elOne.addContent( elDelay )

#  ... and <length>
elLength = Element(  "length", xmlnsl)
elLength.addContent(  "25"  )
elOne.addContent( elLength )

# then tell the SOAPProtocol handler about the request document.
doc = Document( elOne )
body.setDocument( doc )

print "Here is the finished XML request document:"
xo = XMLOutputter()
print xo.outputString( doc )
print

print "Sending request to server..."

response = protocol.connect()
print response
print "Agent done."
```

下面是我们可能从 **response_msg** 服务获取的示例。

```
<?xml version="1.0" encoding="UTF-8"?>
  <responder>
     <duration>103</duration>
     <message>lingo-text</message>
  </responder>
```

构造文档样式的 SOAP 请求的步骤与构造 RPC 样式的请求代理的步骤基本相同，但有一点例外：该脚本需要构造一棵表示 XML 请求数据的 DOM 树，以创建 SOAP 请求。下面是它的工作原理：

```
# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler, SOAPProtocol, SOAPBody, SOAPHeader
from com.pushtotest.tool.response import Response
from java.lang import Long
```

首先，从 TestMaker 的 TOOL 中导入 SOAPProtocol 处理对象。

```
# This agent also uses JDOM APIs to handle XML data
from org.jdom import Document, Element, JDOMException, \
Namespace, DocType
from org.jdom.output import DOMOutputter, XMLOutputter
```

我们也导入 JDOM 对象来处理 XML 数据的构造和分析。

```
host = "examples.pushtotest.com"
port = 92
path = "axis/services/responder_msg"
endpoint = host + ":" +str( port ) + "/" + path
```

这些变量定义了目标服务器和 Web 服务的位置。详细情况请参考 responder_msg 服务的 WSDL 文档，该文档可通过 http://examples.pushtotest.com:92/axis/servlet/AxisServlet/responder_msg?wsdl 访问。

```
protocol = ProtocolHandler.getProtocol("soap")
body = SOAPBody()
protocol.setBody(body)
```

我们向 TOOL 请求 SOAPProtocol 处理器对象和 SOAPBody 对象以构造请求。SOAPBody 包含了请求的参数和请求的目标。

- Responder 从一个 XML 请求文档中获取这些参数：wordcount 是一个 long 类型的数据，它包含从响应的 XML 文档中返回的乱语单词的数目。每个单词大约有 5~8 个字符长，每个响应都是伪随机惟一的。
- Delay 是一个 long 类型的数据，它包含在响应请求前 Responder 服务需等待的毫秒数。

```
xmlnsl = "urn:responder_msg"
```

这里引用了请求中参数的命名空间。在文档样式的 SOAP 请求中，TestMaker 使用第一个元素<Respond>的命名空间来决定目标服务名。最终获得对正确服务器的请求，第一个元素的命名空间能够使我们获得正确的 Web 服务。

使用 JDOM API 在 DOM 对象中创建 XML 请求。

```
# Create the request document by first creating the Respond
element.
elOne = Element( "Responder", xmlnsl )

# Add child elements for <delay>...
```

```
elDelay = Element( "delay", xmlnsl )
elDelay.addContent( "12" )
elOne.addContent( elDelay )

# ... and <length>
elLength = Blement( "length", xmlnsl )
elLength.addContent( "25" )
elOne.addContent( elLength )

# then tell the SOAPProtocol handler about the request document.
doc = Document( elOne )
body.setDocument( doc )
```

接着，向 SOAPBody 对象传递 DOM 对象 doc ，从而对服务器进行 SOAP 调用。

```
response = protocol.connect()
print response
```

最后，准备发送该请求。connect()方法发送请求并返回一个 response 对象。Import 命令通知 TestMaker 使用 TestMaker 的 TOOL 中的 SOAPProtocol 对象。

对于不想使用 JDOM API 的人来说，TOOL 提供一种便利的方法，利用字符串值来设置 XML 文档。

```
myXMLDocument = '<s:Envelope '
myXMLDocument = \
'xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">'
myXMLDocument += '<s:Body>'
myXMLDocument += '<Respond xmlns="urn:responder_msg">'
myXMLDocument += '<delay xmlns="urn:responder_msg">25</delay>'
myXMLDocument += '<length xmlns="urn:responder_msg">75</length>'
myXMLDocument += '</Respond>'
myXMLDocument += '</s:Body>'
myXMLmocument += '</s:Envelope>'
body.setDocument( myXMLDocument )
```

在本例中，myXMLDocument 字符串包含了要发送到主机的 XML 请求。这在调试基于 SOAP 的 Web 服务时很方便。但是该方法很容易出错，因为需要手工构建一棵有效的 XML 树。

本代理只是一个开始。扩展的代理可以发送包含不同大小和延迟的多个请求、操作请求脚本的多个并发线程以增加服务器的负载、在多个机器上运行多个代理的多个副本，以及添加代理的日志功能，所以我们可以确定每秒完成的事务数(transactions-per-second——TPS)和 SPI 值。

3. 使用已格式化的 XML 数据

考虑您已拥有格式化的 XML 文档的情形。下面的代理演示了怎样向基于 SOAP 的 Web 服务发送 XML 格式的文件编码内容。

```
from com.pushtotest.tool.protocolhandler import ProtocolHan-
dler, SOAPProtocol, SOAPBody, SOAPHeader
from com.pushtotest.tool.response import Response
from java.lang import Long
from org.jdom.input import SAXBuilder
```

```
from org.jdom import Namespace, Document

builder = SAXBuilder()
doc = builder.build( 'c:/fileexchange/billing2.xml' )
elEnvelope = doc.getRootElement()
ns=Namespace.getNamespace("env", \
  "http://schemas.xmlsoap.org/soap/envelope/")
  elBody = elEnvelope.getChild( "Body", ns )
  nsm = Namespace.getNamespace("m", \
 "http://wwwoxerox.com/dcs/billing" )
elSend = elBody.getChild( "send", nsm )
doc2 = Document(elSend.detach() )
print doc2
```

在这个测试代理中使用了 Import 命令，以访问 TestMaker 的 TOOL 对象。

```
builder = SAXBuilder()
doc = builder.build( 'c:/fileexchange/billing2.xml' )
elEnvelope = doc.getRootElement()
```

SAXBuilder 对象有个特殊的方法，它采用文件名作为参数。builder 方法读取文件的内容并返回一个包含目录内容的 DOM 树的 XML 对象。

```
elBody = elEnvelope.getChild( "Body", ns )
nsm = Namespace.getNamespace("m", \
"http://www.xerox.com/dcs/billing" )
elSend = elBody.getChild( "send", nsm )
doc2 = Document(elSend.detach() )
```

接着，我们将在文件中找到的 XML 数据附加给 SOAPBody 对象。现在我们准备发送请求和接收响应。

7.8 验证响应数据

在处理 XML 数据时，验证基于 SOAP 的 Web 服务的响应，可以提高软件开发人员的效率。验证响应数据需要编写脚本来分析包含在响应中的 HTML 代码。这种分析很容易出错，并且需要艰苦的编写和维护工作。下面是一段等价的 XML 验证代码，用于检验来自 SOAP Web 服务的响应。

```
builder = SAXBuilder()
doc = builder.build( StringReader(resp.toString() ) )
rootElement = doc.getRootElement()
bodyElement = rootElement.getChild( "Body", ns )
for i in bodyElement.getChildren():
    print "Child = ", i
    print "Child name = ", i.getName()
    print "Child content = ", i.getContent )
```

7.9 使编写代理更容易

TestMaker 自带的新建代理向导能够自动编写测试代理脚本的框架。为了使代理测试基于 SOAP 的 Web 服务，新建代理向导通过给定的 URL 读取 WSDL 文档，并创建 TestMaker 脚本来调用请求 Web 服务功能所必需的 TOOL 命令。框架是功能性的，可以由用户添加额外的测试逻辑。详情请参考 TestMaker 文档。

除了新建代理向导，TestMaker 还提供了一些测试代理示例。这些代理显示了怎样调用基于 SOAP 的 Web 服务。这些示例也可以作为您自己的测试代理，也可以把其中的示例代理作为模板。

最后，在测试基于 SOAP 的 Web 服务时，当 HTTP 协议的数据在浏览器和服务器间的网络连接中移动时，查看该数据，有助于创建和调试智能测试代理。Windows NT、2000、XP 用户可以使用 Microsoft Network Monitor 工具来监视网络信息流通量。该实用工具捕获数据包并显示它们，以显示 GET 和 POST 命令及其相关联的数据值。Linux 用户也可以使用流行的 tcpdump (http://www.tcpdump.org)、ethereal(http://www.ethereal.com)和 NetWatch(http://www.slctech.org/~machay/netwatch.html)来查看网络信息流通量。

7.10 资源

若需进一步了解关于 SOAP 和 WSDL 标准的信息，请访问：

- WSDL, http://www.w3.org/TR/ws-desc-reqs/
- SOAP, http://www.w3.org/TR/soap12-part0/
- XML-RPC, http://www.xmlrpc.com/
- 通用 Web 服务, http://www.w3.org/2002/ws/

7.11 本章小结

本章主要覆盖了基本的 SOAP 和 WSDL 技术。我们学习了 TestMaker 如何提供脚本语言命令和测试对象库，以处理 WSDL 文档和一些 SOAP 请求的多种风格和格式。最后，本章证明了相对于分析原始的 HTML，分析 XML 响应更加容易。

在第 8 章将介绍如何使用智能测试代理来创建测试，包括 TestMaker 代理如何从数据库中直接读取数据。

第 8 章　状态、设计和测试

在前面章节的示例中，假定对 Web 使能应用程序的每个请求都是无状态的。在无状态的系统中，每个请求独立于其他请求。当然，我们知道在 Web 使能应用程序中，如果不是非常必需的话，很多时候有状态的请求是很重要的。本章阐述了为什么状态对于 Web 使能应用程序很重要，并且讨论了一些用于测试有状态系统的方法，最后给出了一个实例来说明如何配置有状态系统以供测试。

8.1　状态问题

Web 技术引入了一系列协议、提供商和语言。可能这些中最大的特征就是 Web 协议的无状态性。在无状态的系统中，每个请求和响应都是原子的——不依赖于前面或后面的请求或响应。对服务的请求，以及对请求的响应都相当于事务。有状态的系统的优点包括：

- 容易规划容量。例如，通过测试 100~500 个并发用户的性能就可以知道 1000~10 000 个用户并发时的系统性能。
- 可进行内容缓冲。据我的经验，无状态的系统对于很少变化的内容很有效。内容越为静态的，分布式缓冲服务器就越能够有效地向用户传递内容。
- 分布的数据中心更加公平地共享大量用户的负载。因为使用哪一个数据中心来处理请求是无关紧要的。这就增加了整个系统的稳定性。

随着 Web 技术的成熟，很多地方都需要有状态的技术来创建 Web 使能应用程序。考虑一个系统，它需要用户登录，以从服务器接收个性化的或确认的响应。在无状态 Web 使能应用程序中，用户需要在每个 Web 页填写用户名和密码。在很多时候这样做是不合适的，它与绝大多数用户理想中的 Web 工作方式相反。

通过添加一些状态——例如，利用一个 cookie 值来追踪登录会话——用户更容易操作且底层协议仍具有可扩展性。遗憾的是，这种情形如同潘多拉盒子。现在有状态的盒子已打开，软件开发人员、QA 技术员和 IT 管理员不得不努力处理与有状态系统相关的大量问题、难题和挑战。下面是我所特别喜欢的一些状态：

- cookie——提供简单的方式以确定 HTTP/HTML 请求组之间的会话。cookie 值通常是到表的索引。表存储在 Web 服务器的存储器上。这会导致很多潜在的问题：如果并发用户的请求被路由到不同的服务器，服务器就不知道会话的信息。如果用户被路由到不同的服务器，且服务器是应用集群的一部分，那么所有的服务器都可能接收用户的请求，并同步会话数据。存储 cookie 和同步集群服务器之间的会话通常需要配置、存储空间和存储器。
- URL 重写——除了在请求的 HTTP 头中存储 cookie 值外，还需要重写 URL，以包含会话参数。URL 重写能够避免 cookie，但它会遇到上面所提到的同样的问题。另外，使

用 URL 重写，在 Web 使能应用程序中没有静态 URL，这通常使得缓冲和索引更加困难。最后，每个 Web 页需要动态生成，因此所有超链接都包括会话参数。

- SAML Artifact——SAML 是一个安全协议，它使得 Web 使能应用程序能够验证用户和数据。设计 SAML 很大程度上是为了 Web 使能应用程序，其浏览器接口在 HTTPS/HTML 协议之上提供了一个安全功能接口，并且在 Web 服务上也很有用处。当 SAML 需要共享数据，且这些数据太大以至于无法作为一个 URL 参数时，它使用 Artifact。在设计时需要存储 Artifact，这样在后面就可以获取和使用它。这需要软件来配置 Artifact、内存和存储器。通过访问 http://www-106.ibm.com/developerworks/xml/library/x-samlmyth.html 可以看到我撰写的关于 SAML 的文章。

在 Web 使能应用程序中设计的状态越多，越需要开发更多的工作区来使用 Web 技术的无状态属性。对于软件开发人员、QA 技术员和 IT 管理员来说，这些工作背景通常比较混乱。当测试一个有状态的系统时，在增加使用负载的情况下，我们需要注意系统性能的两个方面：

- 垂直可扩展性——允许系统按时间次序处理单个事务。
- 水平可扩展性——系统并发处理所有请求的能力。

在这个讨论里，想象一个事务需要跨越几个步骤。例如，在一个事务里用户登录到一个 Web 使能应用程序，读取第一个等待消息，删除该消息，然后检查其余的等待消息。图 8-1 演示了这个示例。我们运行同一个事务两次。在比较这两次事务运行时，提出了下面的问题：

- 所有的时间性能是用户可接受的吗？
- 事务的哪些部分影响性能？
- 反复测试的变化多吗？
- 我希望变化吗？如果是这样的话，我希望变化多少？
- 变化会影响事务的其余部分吗？

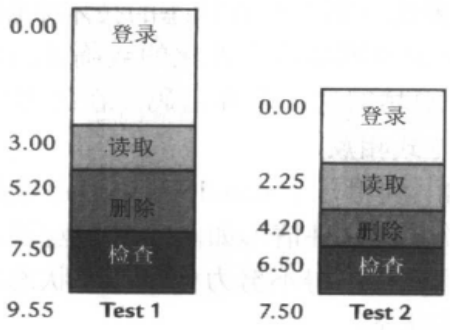

图 8-1 系统规模有两个方向：垂直方向为提高每个请求的响应服务能力，水平方向为提高处理增长的请求数量的服务能力。本图显示了垂直可扩展性测试，其中同一个事务的两次运行产生了不同的响应次数

以我的经验，同样的问题可以应用到我设计和测试的所有 Web 使能应用程序中。利用一个测试代理日志来评测软件和了解系统的性能，我们能够观察到系统的哪些地方需要优化。系统一旦在以后发生了变化，同样的测试也能够证明性能的改进。

观察一组单个请求和响应事务，如图 8-1 所示，一些事务能够很快被处理，而另一些事务要花费很长的时间。理想的垂直可扩展系统的快速响应和慢速响应之间具有固定的比例。当系统出现多个请求，从而导致处理事务的平均时间增多时，会出现垂直可扩展性问题。

水平可扩展性是系统处理并发请求的能力。图 8-2 演示了对水平可扩展性示例的测试。服务器需要资源来响应请求。当资源耗尽时，服务器不得不拒绝额外的连接和请求。负载平衡器、集群技术和多播服务被证明是支持服务器组作为系统来提供水平可扩展性的解决方案。

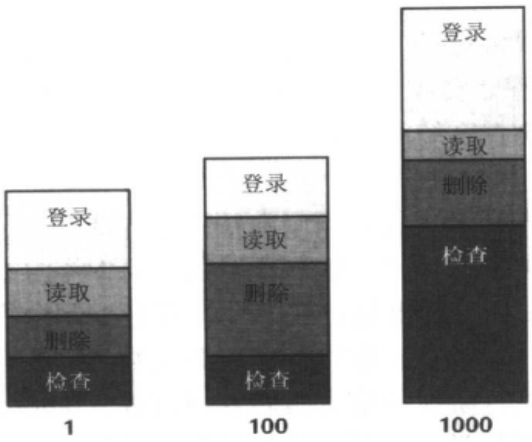

图 8-2　水平可扩展性测试系统处理增长的负载的能力。注意，在不同的负载情况下，一个事务的每一部分的执行情况是不同的

以我的观点，将 Web 技术扩展为有状态系统是为了解决糟糕的性能和可扩展性问题。系统结合的状态越多，在测试前配置系统就越复杂。例如，考虑有状态系统在使用一定量之后所展示的性能问题。下一节将详细阐述这个示例。

8.2　测试有状态系统的生命周期

开发智能测试系统代理非常类似于开发软件应用程序，包括对可重复生命周期过程的需求。测试系统需要一定的准备、安装、执行和分析。设计良好的测试需要围绕生命周期进行设计，它要求从配置到报告结果的所有过程都自动进行。图 8-3 演示了测试生命周期的整个过程。

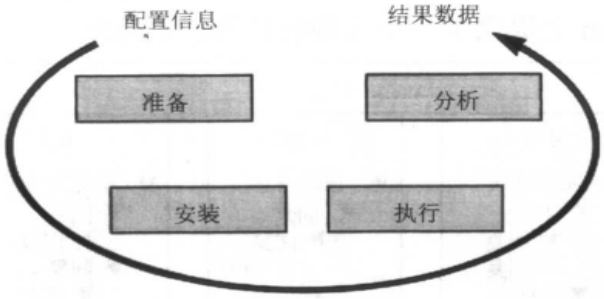

图 8-3　测试有状态系统的生命周期从配置系统开始，经过多个生命周期步骤，直到报告结果

在测试的准备阶段，测试设计者收集配置系统所必需的步骤，以在测试前将系统配置为已知状态。这里的目标不是配置系统(下一步是安装阶段)，而是理解在一定抽象层次上对系统进行测试所需的步骤。例如，测试某个商业系统，该系统显示商业电话客户的折扣电话服务计划，系统需要获取当前服务计划列表以模拟客户的活动。准备阶段将确定获取当前的服务计划所需的命令。

准备阶段的优点是使得测试变得更加动态化和自动化。通过对安装和执行阶段的抽象活动，对于不同的测试，测试代理需要不同的准备，而安装、执行和分析不需要变化。另外，准备阶段的抽象活动意味着任何代理、脚本或程序能够根据这些活动来配置系统，以进行自动化的测试。

接下来的两步——安装和执行——完成测试的艰苦任务。安装执行准备阶段定义的所有任务，包括与 Web 使能应用程序、数据库、文件和消息传递系统的通信，以配置目标测试系统。安装中也执行验证操作，以检查系统的实际准备是否成功。如果定义的配置参数不满足，安装步骤将禁止全部测试。

执行阶段负责按照测试设计者定义的顺序和频率来完成测试步骤。执行阶段通过控制台从用户那里或通过安装阶段的配置文件来接收测试时的运行时参数。执行阶段的输出格式是在分析阶段定义的。对于简单的测试，输出显示在一个计算机控制台窗口中。更高级的测试代理会创建分隔的数据文件，或者将结果数据存储在一个关系数据库中，供分析阶段使用。

在分析阶段，来自测试执行阶段的信息被转换成可控告(actionable)的信息。分析阶段首先收集原始的结果数据。当管理员学习了系统质量和性能的评价尺度之后，数据就变得有价值。这些数据可以使工程师洞察他们那些运行在开发系统中同样的不可用代码。正如 SunONE 应用服务器组的一个管理员所说："我们需要体验一下最近的构建性能。"在分析阶段结束时，将原始数据编译为一个简练的报表，以提供管理活动的结果。

在下一节，我们将学习怎样将生命周期应用到一个研究可扩展性和性能的示例中。

8.3 建立状态的技术

考虑内容发布系统中的线程讨论。通常需要读取很多消息，首先需要几个新的讨论，订阅者可能发送许多回复消息。公司把发布系统放到主机上并扩展其使用，以满足新的市场项目。管理员需要知道数据中心的容量，并进行预算，以购买处理新用户所需增加的设备。图 8-4 演示了所讨论的系统的特征。讨论按照主体分类，每个讨论包含一个顶层消息和多个线程化的 reply 消息。The Server Side 对于 J2EE 开发人员来说，是一个很受欢迎的 Web 站点，站点 http://www.theserverside.com 上提供了一个这种特征的讨论系统。

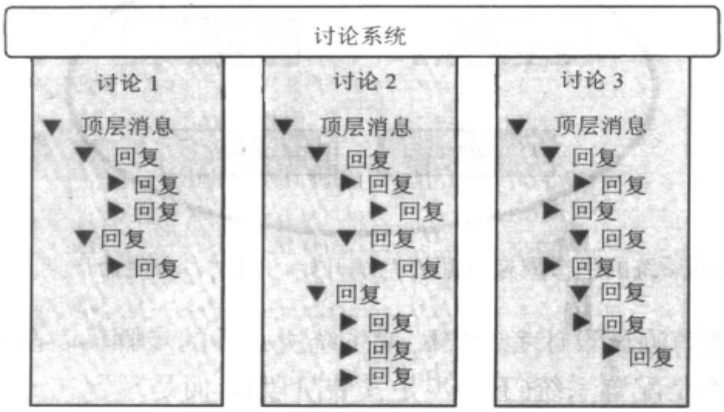

图 8-4 一个示例讨论系统，由多个线程化的讨论组成

如果无状态系统和有状态系统之间存在一个标尺，我们将发现这个示例系统更倾向于有状态系统一边。考虑执行一个可扩展性和性能测试所需的状态问题：

- 必须确定用户在创建新的讨论、顶层消息或回复消息时，是否拥有发送特权。
- 为了创建回复消息，用户必须首先浏览顶层消息或回复消息。浏览这些消息后，设置回复消息的上下文。
- 消息表现为立即回复消息。用户需要浏览消息列表来查看所有的响应。

测试讨论系统提出了几个关于测试的经典问题。这种测试的开端是一个现存系统需要在增长级别的负载情况下进行性能评估。根据它们的特征，有状态系统的测试改变了正在测试的系统。因此在现有系统上运行几千个并发运行的智能代理将导致整个系统的性能下降。下面介绍在现有的有状态系统测试中存在的一些可能环境：

- 对目前讨论的软件状态做一个快照，并将快照安装到复制的数据中心。这里测试将始于匹配当前系统状态的基线。通过在复制的数据中心上进行测试，用户之间不会发生冲突。这种测试存在两个缺点：第一，从花费和时间上看，对于测试，复制一个数据中心是不切实际的。第二，在实际的数据中心上进行测试可能会揭示在复制的系统上不能显示的问题——创建一个严格的副本可能成为终身的追求。
- 离线测试产品系统。通知现有的用户，测试将在非高峰使用期进行。在进行系统备份后，对刚刚安装的应用程序软件进行测试。这种测试的缺点是测试环境可能与实际的产品环境不一样。例如，不能测试存储器对系统的影响，因为备份和恢复操作删除了内存泄漏。
- 持续的测试——在系统运行时测试，并划分系统，但这使得测试数据对现有的用户不可用。在讨论的系统示例中，测试创建了新的讨论组，并仅仅对测试代理可见，对现有的用户不可见。测试代理将消息传递给可见的讨论组，测试活动效果的度量被显示为来自现有用户的通常活动之上的增量活动。

不考虑所选择的测试场景，将测试运行几次，并且每次测试都需要配置相同的状态。使用可重复的准备和安装技术，提高测试的速度，因为这样最小化了初始系统在每次测试运行变化时的风险，这往往会导致结果的变化。图 8-5 演示了对所测试的系统建立状态的技术。

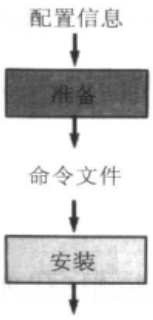

图 8-5　在准备阶段和安装阶段共享了一个命令协议，以在测试有状态系统之前建立状态

在讨论系统示例中，对测试系统进行配置的安装阶段具有以下几个命令：

- 重置整个系统
- 删除讨论组
- 开始新的讨论组
- 创建顶层消息
- 回复消息

这些组成了简单命令协议的基础，测试的安装阶段能够处理它们以准备系统的测试。

在讨论系统示例中，我们将命令代码赋值给每一个配置命令，如表 8-1 所示。

命　令	任　务
1	重置整个系统
2	删除讨论组
3	开始新的讨论组
4	创建顶层消息
5	回复消息

根据这个命令协议，我们可以通过处理下面的指令来配置新的讨论组：

```
1, 3, 4, 5
```

依次处理这些命令后，将配置所测试系统的状态。

准备和安装技术为配置信息源提供了很大的灵活性。图 8-6 演示了从文件、数据库，甚至是 Web 使能应用程序中读取数据作为配置指令源，从而建立要测试系统的状态的技术。

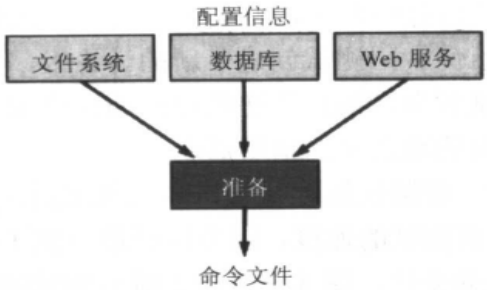

图 8-6　测试的准备阶段从几个源处获取输入，包括存储在文件系统、数据库，甚至是 Web 使能应用程序中的准备指令，并生成一个安装阶段所使用的命令文件

在下一节中，我们将使用智能测试代理，实际练习准备和安装技术。

8.4　准备和安装代理

下面介绍一个 TestMaker 测试代理，该测试代理执行供测试的示例讨论系统的配置准备阶段。下面是整个代理的源代码，及其工作原理。本书中的所有代码都可以通过 http://www.pushtotest.com/docs 下载。

```
# Prepare_setup_agent.a
# Author:  fcohen@pushtotest.com

print "Agent running:  Prepare_Setup_agent.a"
print "Description:"
print "This is the first of two agents that"
print "demonstration how to automate a Web-enabled"
prznt "application test:"
```

```
print
print "   1)  Prepare_setup_agent.a - creates a"
print "       delimited data file containing"
prznt "       instructions for the setup_agent.a"
print         agent to follow."
print "   2)  Setup_agent.a - issues commands to an"
print "       example ecollaboration"
print "       Web-enabled application to create the"
print "       needed state to run the test."
print

# Parameter settings

# The file to store the commands

filename =  "d://agents//experimental//commands"

# The host to set-up

dest_url = "examples.pushtotest.com"
dest_params =  "responder"

# Import tells TestMaker where to find TOOL objects

from com.pushtotest.tool.util import Lingo
import random
import sys

# A mini-function that writes the command to the file and
# handles exceptions. The file contains instructions for
# the setup_agent to follow.  Setup_commands uses the
# following format:
# command-type,  url,  parameters,  message_size

def write_a_command( file,  command ):

    # The global command makes tells the function to use
    # variables defined outside of this function. Otherwise
    # this function would define it's own dest_url variable
    # locally

    global dest_url,  dest_params,  tick

    thecommand = str( command )  + "," + dest_url + "," + \
    dest_params +  ","  + str(random.randrange(100,150)  )  \
    +   "\n"
    print "writing this command:",thecommand

    try:
        file.write( thecommand )
    except Exception,  ex:
        print  "Uh,  oh:",ex
        print  "Closing file and exiting."
        file.close()
        sys.exit

    tick += 1
```

```
# Main:
# Start by first creating a data file for the commands

fn = filename +  ".db"
print  "Building:",  fn

# The w means write, which creates a new file,  deleting
# any existing file with the same name. One could also
# use a for append, which appends entries to the end of
# an existing file.  r opens an existing file to be read.

fl = open( fn,  "w")

tick = 0     # This will keep a count of the commands we write
print

# Define the constants used to create the command file.
# A copy of these constants will be found in
# Setup_agent.a too.

new_thread = 0
new_node = 1
pop = 2
reply = 3

# Write the configuration commands to initialize a new
# discussion group to a known state.

write_a_command ( fl, new_thread )
write_a_command ( fl, new_node)
write_a_command ( fl, reply )
write_a_command ( fl, pop )
write_a_command ( fl,  new_node)
write_a_command ( fl,  new_node)
write_a_command ( fl,  reply )
write_a_command ( fl,  reply )
write_a_command ( fl, pop )
write_a_command ( fl, pop )
write_a_command ( fl, new_node )

print
print "Wrote",  tick,  "commands in total."
# Close the file

print
print "Closing the file."
fl.close()

# Run the Setup_agent.a and then return back to this agent

print
print "Running the Setup_agent.a agent."

execfile(  "d://agents//thebook//chapter5-setup//
Setup_agent.a"  )
```

```
print
print "Agent done."
```

Prepare_Setup_Agent 是两个代理中的第一个，它阐述了怎样为测试自动配置一个有状态的系统。该代理创建了一个分隔格式的数据文件，文件中包含 Setup_agent 指令，以用于配置讨论系统。

Prepare_Setup_Agent 首先创建一个变量，以定义安装代理的命令文件的路径和文件名。Prepare_Setup_Agent 代理将配置命令存储在该文件中。

```
filename = "d://agents//experimental//commands"
```

注意：在本例中使用了 Windows 风格的路径和文件。代理很容易使用 Unix 风格的路径，例如：

```
filename = "/home/fcohen/commands"
```

代理导入了 sys 对象，如果发生错误或异常状态，该对象将终止代理。

```
import sys
```

write_a_command()函数是一个小的实用函数，该函数把配置命令写到文件里，并处理文件相关的异常。文件中包含了 Setup_agent 所遵守的指令，并使用下列格式：command-type, URL, parameters, message_size。逗号隔开了各个字段，每个记录以回车字符终止。

```
def write_a_command(  file,  command ):
```

全局命令通知 write_a_command()函数使用在该函数外部定义的变量。否则，该函数会定义和仅仅使用它自己的 dest_url、dest_params 和 tick 局部变量：

```
global dest_url,  dest_params,  tick
```

write_a_comman()函数接受对已经打开的文件处理程序(file)的引用。下面的命令创建一个命令并将其写到文件中：

```
   thecommand = str( command )  + "," + dest_url + "," +
dest_params +  ","  + str(random.randrange(100,150)  )  +  "\n"
```

tick 变量跟踪写入文件中的命令数。这里我们使用+=来表示当前的 tick 值加 1，然后赋给 tick 变量。

```
tick += 1
```

接下来的是代理的主体。代理使用“w”选项打开文件，以创建一个新文件，如果有同名文件存在，则删除该同名文件。代理也可以使用“a”选项，将记录项追加到现存文件的末尾。“r”选项用于打开一个已存在的文件，并读取该文件。

```
fl = open(  fn,  "w")
```

Prepare_Setup_Agent 代理定义了一个简单的命令协议，Setup_agent 将使用该协议配置讨论系统。代理也定义了一些常量，用于标识那些已经被添加到 command 文件中的命令。Setup_agent 中具有这些常量的副本。这些常量实现了如下测试命令：

- new_thread——开始一个新的讨论线程，并发出最顶层消息。
- new_code——回复最后发出的消息，它记录了当前消息的位置，供后续的 pop 命令使用。
- pop——回复前面 new_code 命令创建的消息。
- reply——回复最后发出的节点。

Prepare_Setup_Agent 创建了一个讨论组，如图 8-7 所示，其方法是使用了这些命令的组合。

图 8-7　在准备配置文件中定义的新讨论组所期望的最终状态

代理将这些命令写到配置命令文件中，以创建图 8-7 中演示的讨论：

- new_thread——创建 1
- new_node——创建 2
- reply——创建 3
- pop——移回到 1
- new_node——创建 4
- new_node——创建 5
- reply——创建 6
- reply——创建 7
- pop——移回到 4
- pop——移回到 1
- new_node——创建 8

然后代理关闭该命令文件。如果不关闭该文件，Setup_agent 在试图打开该命令文件时，就会产生文件输入/输出错误。

```
f1.close()
```

最后，代理使用 execfile 命令来运行 Setup_agent。execfile 运行代理，并在 Setup_agent 结束时返回。execfile 简化了编写运行代理的主代理的任务。

```
execfile( " d://agents//thebook//chapter5- \
-setup//Setup_agent.a" )
```

Setup_agent

前面一节演示了 Prepare_Setup_Agent 如何创建包含配置命令的 commands 文件，以在示例讨论系统中创建正确的状态。下面是 Setup_agent 的全部源代码及其工作原理的解释。

```
# Setup_agent.a
# Created on:  June 30,  2002
# Author:  fcohen@pushtotest.com
```

```
print "Agent running:  Setup_agent.a"
print "Description:"
print "  This the second of two agents that"
print "  demonstration how to"
print "  automate a Web-enabled application test:"
print
print "  1) Prepare_setup_agent.a - creates a delimited"
print "      data file containing"
print "      instructions for the setup_agent.a agent"
print "      to follow."
print "  2) Setup_agent.a - issues commands to an"
print "      example ecollaboration Web-enabled application"
print "      to create the needed state to run the test."
print

# Parameter settings

# The file to store the commands

commandfilename = "d://agents//experimental//commands"

# The host to set-up

dest_url = "examples.pushtotest.com"
dest_params = "responder"

host = "examples.pushtotest.com"
port = 92
path = "axis/servlet/AxisServlet.
endpoint = host +  ":"  + str( port )  +  "/"  + path

# Import the sys object to use the exit() method for errors
import sys

# Post a message function
# input the parent_message number
#
# Note: This is just a simulation of a Web-enabled applica-
tion request.
#       Someday I may get around to coding a real service.

def post_message( msg_num,  theindent,  type, url, params ):
   print indent*'    ',  type, msg_num,  ":", url, params

# Simple Stack implementation
# Used to keep track of nodes in the message hierarchy

def push( value ):
    global stack
    stack +=  [ value ]

def popone():
    global stack
    return stack.pop()
# Main:
```

```
# Let's start by first opening the command data file

try:
    target =  commandfilename + ".db"
    print "Opening file:  ",  target
    print

      dbfile = open( target )

except IOError,  e:
    print "Error while opening the file:"
    print e
    print
    sys.exit

# Define the constants used to create the command file.
# Prepare_setup.a uses the same constants when it builds
# the command file.

# Starts a new thread of discussion and posts
# the top-most message
new_thread = 0

# Reply to the last message posted,  this remembers the
# current message location for a subsequent pop command
new_node = 1

# pop replies to the message created with previous
# new_node command
pop = 2

# Replies to the last node posted
reply = 3

# So if we wanted to create a message tree like this:
# top-message 1
#    reply 2
#      reply 3
#    reply 4
#      reply 5
#         reply 6
#         reply 7
#    reply 8

# we could use the following commands:
# new_thread - creates 1
# new_node - creates 2
# reply - creates 3
# pop - moves back to 1
# new_node - creates 4
# new_node - creates 5
# reply - creates 6
# reply - creates 7
# pop - moves back to 4
# pop - moves back to 1
```

```
# new_node - creates 8

# And that's just what Prepare_setup.a did to build the
# command.db file. So we will now loop through each record
# in the command.db file and process each command.

# Create the variables

tick = 0            # Counter for the number of commands pro-
cessed
next_message = 0    # Identifier for the next new message
indent = 0          # Used to figure out the indentation of a
reply
stack = []          # used to build the tree

# Process the commands

for line in dbfile.readlines():
    commands = line.split(",")
    command = commands[0]
    handled = 0

    if int( command ) == new_thread:
        handled=1
        tick += 1
        node=post_message(next_message,indent, \
           "top-message",commands[1],commands[2])
        push( indent )
        next_message += 1
        indent += 1

    if int( command ) == new_node:
        handled=1
        tick += 1
        node=post_message(next_message,indent, \
       "node-reply",commands[1],commands[2])
        push( indent )
        next_message += 1
        indent += 1

     if int( command ) == reply:
        handled=1
        tick += 1
        node = post_message(next_message, indent, \
        "reply", commands[1], commands[2] )
        next_message += 1

    if int( command ) == pop:
        handled=1
        tick += 1
        indent = popone()

    if handled==0:
        print "Unknown command!"
        print line
```

```
# Cleaning up

print
print  "Processed",  tick,  "commands in total."

print
print  "Closing the file."
dbfile.close()

print
print  "The collaborative Web-enabled application is cor-
rectly set-up"
print  "and ready to test."
print
print  "Agent done."
```

setup_agent 从 commands 文件中读取命令，并每次执行一个命令。Setup_agent 将 Web 使能应用程序请求发送给示例讨论系统，以创建一个新的讨论组，且该讨论组被回复消息树填充。

```
def post_message( msg_num,  theindent,  type, url, params ):
```

然而 post_message()函数实际上并没有将请求发送给 Web 使能应用程序，它将消息打印到 TestMaker 的输出窗口，包括打印正确的缩进空格(indentation)。indent*' '命令让 4 个空格字符的字符串和 indent 变量值相乘。如果 indent 等于 2，就输出 8 个空格字符。

```
print indent*'  ',  type,  msg_num,  ":",  url,  params
```

在前一节中，Prepare_Setup_Agent 创建的 commands 文件包含了一个命令列表。commands 文件使用 push 和 pop 命令来跟踪何时将回复消息添加到回复的“分支”上。图 8-8 演示了分支的位置。例如，当写完“回复 7”之后，commands 文件持续发出 pop 命令，直到我们回到“顶层消息 1”，然后它命令 Setup_agent 写“回复 8”。

Setup_agent 实现了一个简单的 push/pop 堆栈，用于跟踪一个讨论组层次中的分支。push()将当前的分支号压入一个值堆栈中，每个分支用一个值表示。

```
def push( value ):
    global stack
    stack +=  [ value ]
```

popone()函数把最近压入堆栈中的分支号从堆栈顶部弹出。

```
del popone():
    global stack
    return stack.pop()
```

该 push/pop 堆栈可以被应用到其他代理中。为了在操作中查看堆栈，您可以添加下面的命令来打印堆栈的内容。

```
for stack_item in stack:
    print stack_item
```

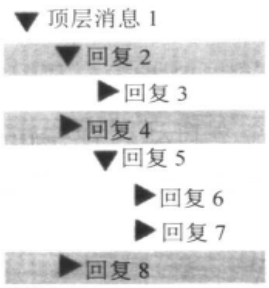

图 8-8　回复消息 2，4，8 是顶层消息的分支。在创建讨论组的时候，代理必须记住分支的位置，以知道如何处理创建新分支的命令

代理首先打开 commands 文件。

```
dbfile = open( target )
```

默认情况下 open 命令打开的文件具有读访问权限。

然后代理复制 Prepare_Setup_Agent 中最初定义的命令常量。代理遍历 command.db 文件中的每一个记录，并处理每一个命令。虽然 post_message()函数实际上并不将请求发送给 Web 使能应用程序，但是它将带有正确缩进空格的回复消息打印到屏幕上。

接着代理创建处理命令期间使用的变量，包括一个所处理的命令数的计数器，下一个新消息的一个标识符和用于跟踪回复消息树中下一个回复消息的缩进空格的一个整数值。

Setup_agent 处理 commands 文件中的每一个命令。for 命令遍历 commands 文件的每一行。

```
for line in dbfile.readlines():
```

通过从 commands 文件中读取一行，并使用逗号分隔符将其分成单独的值，代理创建了一个名为 commands 的列表变量。

```
    commands = line.split(",")
```

commands 变量是一个列表，因此 commands[0]为 commands 文件中当前行的第一个值。在讨论系统示例中，第一次循环后，commands[0]包含一个 new_thread 命令。

代理处理命令并采取相应的操作。当处理一个 new_thread 命令时，代理指示命令已被处理，增加处理命令的数目，使用 post_message 函数来发送消息，并使用 push()函数将缩进空格(indentation)值压到堆栈中，接着增加消息和空格值的数目。

```
if int( command ) == new_thread:
    handled=1
    tick += 1
    node=post_message(next_message,indent,  \
    "top-message",commands[1],commands[2])
    push( indent )
    next_message += 1
    indent += 1
```

handled 变量是一个简单的标志，代理使用它来确保每个变量都被识别并执行。如果在下一代码块的结尾处 handled 变量仍旧为零，则将抛出一个错误。

```
if handled==0:
   print "Unknown command!"
   print line
```

Prepare_Setup_Agent 使用 execfile 命令自动运行 Setup_agent。当使用 execfile 时，Setup_agent 退回到 Prepare_Setup_Agent，并在 execfile 命令之后继续执行。

8.5 使用数据库配置测试

在本章的开始部分，我们学习了怎样在测试的准备阶段将配置指令格式化到命令文件中。配置指令可以被手工编码到一个从文件或数据库中获取的准备测试代理中。本节将介绍内置于 TestMaker 中的智能测试代理如何与关系数据库系统进行交互。

TestMaker 提供了创建智能测试代理的强大环境，但是本质上 TestMaker 是用 Java 语言编写的一个应用程序。Java 通过 JDBC API 提供数据库连接。JDBC 驱动器可用于商业数据库和开源数据库，包括 Ocacle、Microsoft SQL Server 和 PostgreSQL。

下面是一个测试代理，它演示了怎样在测试代理环境中使用 JDBC 函数。

```
# DB_meister.a
# Created on: July 3,  2002
# Author:  fcohen@pushtotest.com

print "Agent running: DB_meister.a"
print "Description:"
print "  Shows how an agent may use a relational"
print "  database to receive or record data."
print
print "  NOTE: THIS AGENT DOES NOT ACTUALLY RUN since"
print "  there is no relational database source"
print "  available and this agent print depends on"
print "  a JDBC driver that you must supply."
print

# Please configure this agent and then remove the
# sys.exit command

print
print "Stopping agent:  Please configure this agent to your"
print "own JDBC driver and relational database source."
print
sys.exit

# Import tells TestMaker where to find TOOL, JDBC objects
import sys
from java.sql import DriverManager

# JDBC Driver information

# Note: You must provide these parameters to
# use your provided JDBC driver. You must also add the JDBC
# driver code to the TestMaker classpath.  Set the classpath
```

```
# by changing the testmaker/testmaker/bin/ide.cfg file.

JDBCDriverClassName = "com.ashna.jturbo.driver. Driver"
jdbcURL = "jdbc:JTurbo://myrelationaldatabase:1410:MSSQL"
jdbcUserID = "myuserid"
jdbcPassword = "thepassword"

# Load the JDBC driver

try:
    java.lang. Class.forName( JDBCDriverClassName )
    myConn = DriverManager.getConnection( jdbcURL,  jdbcUse-
rID,  jdbcPassword )
except:
    print
    print  "Could not load JDBC driver:", \
    JDBCDriverClassName
    sys.exit

# Show example of reading from a relational database

print
print "================================================="
print "Information from the Customers table:"

# Customer count

query1 = "select count(*)  from Customers"

try:
statement = myConn.createStatement()
results = statement.executeQuery( query1 )
results.next()
except Exception,  ex:
print "Exception while counting customers:  ",ex
sys.exit

print
print "Customer count:  ",  results.int

# Customer list

query2 = "select customer_number,  last_name,  "
query2 += "salutation,  city, year"
query2 += "from table"
query2 += "order by last_name"

try:
statement2 = myConn.createStatement()
r2 = statement.executeQuery( query2 )
except Exception,  ex:
print "Exception while getting customer list:  ",ex
sys.exit

print
print " Complete customer list  (sorted by last name):"
```

```
while results2.next():
print  " ",r2.getString,r2.getString,r2.getString,  \
    r2.getString(4),r2.getString(5)

#-----------------------------------------------------------
# Inserting data into a table

insertl = "insert into customers files  (id, name,"
insertl += "manager, phone)  "
insertl += "VALUES  (1001,  'Simpson',  'Mr.',  "
insertl +=  "'Springfield',  2001)"

try:
il = myConn.createStatement();
irl= s2.executeQuery( insertl );
except Exception,  ex:
print "Exception while inserting a record:  ",ex

print
print "Agent ended."
```

下面通过查看代理脚本的各部分来学习脚本是如何与关系数据库进行通信的。和本书中的其他测试代理不同，DB_meister 代理实际上并没有运行，因为在编写本书时还不存在本代理能够连接上的可用公共关系数据库源。可以通过访问 http://examples.pushtotest.com，可以查看现在是否有可用的示例数据库源。

```
sys.exit
```

在默认情况下本代理开始运行，然后结束。为了运行本代理，请配置代理，将代理连接到一个可用的数据库，并删除 sys.exit 命令。

import 命令告知 TestMaker 在何处寻找标准 Java 库，这些 Java 库用于实现 JDBC 驱动程序的接口。

```
from java.sql import DriverManager
```

JDBC 驱动程序实现了所有与指定类型的数据库进行通信的代码，和一个符合 Sun 公司为 Java 平台发布的 JDBC 规范的接口。数据库提供者和第三方驱动程序制造商提供了 JDBC 驱动程序。驱动程序被打包为 JAR 文件。为了使测试代理访问驱动程序的函数，需要将 JAR 驱动程序文件添加到 TestMaker 类路径中。有关将资源添加到类路径的指令，请参考 TestMaker 文档。

测试代理设置了 4 个基本参数，用于告知 JDBC 驱动程序在哪里和如何连接到数据库。JDBCDriverClassName 指向由驱动程序制造商定义的 Java 命名空间中的驱动程序对象。jdbcURL 参数描述了使用哪一个 JDBC 驱动程序(JTurbo)、所使用数据库的 URL 和数据库的名称(ORCL)。

```
JDBCDriverClassName = "com.ashna.jturbo.driver.Driver"
jdbcURL = "jdbc:JTurbo://myrelationaldatabase:1527:ORCL"
```

驱动程序和数据库制造商定义了 JDBCDriver ClassName 和 jdbcURL 参数的格式。下面给出一个通用 JDBC 驱动程序及其参数的简短列表。

对于 IBM DB2(http://www.ibm.com)，代理使用：

```
JDBCDriverClassName = "COM.ibm.db2.jdbc.net. DB2Driver"
jdbcURL = "jdbc:db2://myrelationaldatabase.com/test"
```

对于 Oracle(http://www.oracle.com)，代理使用：

```
JDBCDriverClassName = "oracle.jdbc.driver. OracleDriver"
jdbcURL = "jdbc:oracle:thin:@myrelationaldata-
base.com:1527:ORCL
```

JTrubo 用于 MS SQL server 的一个 JDBC 驱动程序(http://www.newatlanta.com/index.jsp)：

```
JDBCDriverClassName = "com.ashna.jturbo.driver.Driver"
jdbcURL = "jdbc:JTurbo://myrelationaldatabase.com:1433/ \
dbhost/sql70=true"
```

JDBC 驱动程序使用 Java 的类加载器函数来发现和访问驱动程序的函数。注意：代理的代码使用 try/except 格式来捕获加载驱动程序时可能发生的错误。可能的错误包括：Java 没有发现驱动程序、驱动程序没有连接到数据库，或者错误的配置信息。

```
try:
   java.lang. Class.forName( JDBCDriverClassName )
   myConn = DriverManager.getConnection( jdbcURL,  \
   jdbcUserID,  jdbcPassword )
except:
   print
   print "Could not load driver:  ", JDBCDriverClassName
   sys.exit
```

加载驱动程序后，代理利用几个 SQL 函数从数据库中的 Customers 表中获取数据。Customers 表具有如下几个字段和描述：

- customer_number long
- last_name char(50)
- salutation char(10)
- city char(50)
- year char(10)

第一个 query 语句使用 SQL 选择语句来计数 Customers 表中的客户数。

```
queryl = "select count(*)  from Customers"
```

执行这个 query 语句需要一个连接对象和 statement 对象。代理使用 try/except 格式来捕获在处理数据库函数时发生的错误。

```
try:
statement = myConn.createStatement()
results = statement.executeQuery( queryl )
results.next()
except Exception,  ex:
```

```
print "Exception while counting customers:  ",ex
sys.exit
```

调用结果返回到结果对象中。使用 next()方法来获取数据库结果的第一行。在这种情况下只有一个单行，该行被格式化并被打印到 TestMaker 的输出窗口中。

```
print "  Customer count:  ",  results.int
```

第二个 query 语句演示了一个更加复杂的数据库命令，该命令返回多于一行的结果。query2+= ""格式根据多个部分构建查询语句。

```
query2 = "select customer_number,  last_name,  "
query2 += "salutation,  city, year"
query2 += "from table"
query2 += "order by last_name"
```

代理连接到数据库并返回 results2 对象，该对象保存了来自数据库的响应。while 命令遍历响应中的所有行。

```
while results2.next():
print  " ",r2.getString,r2.getString,r2.getString,\
    r2.getString(4),r2.getString(5)
```

第三个示例显示了代理如何将新行插入到 Customers 表中。数据库命令的构建与前两个查询语句一样。

```
insert1 = "insert into customers files  (id, name,  "
insert1 += "manager, phone)  "
insert1 += "VALUES  (1001,  'Simpson',  'Mr.',  "
insert1 += "'Springfield',  2001)"
```

为了进行查询，代理创建一个新的连接对象，并使用 executeQuery 来处理 Insert 命令：

```
try:
il = myConn.createStatement();
irl = s2.executeQuery( insert1 );
except Exception,  ex:
print "Exception while inserting a record:  ",ex
```

TestMaker 脚本语言提供了对绝大多数 JDBC 方法的访问。您可以参考驱动程序文档，查看其他 JDBC 函数的示例。

8.6 使用 Lingo 使测试内容更接近真实

在本章前面讨论的系统示例中，Setup_agent 通过发出消息和回复消息来创建一个新的讨论组。经验表明，在消息内容相同时，底层系统(Web 使能应用程序服务器、数据库服务器、网络路由器和负载平衡器)尽量使用优化技术来增强系统的性能。然而在现实中很难发现一个讨

论包含同样内容的消息。因此，TestMaker 包含了一个 Lingo 实用程序来自动产生消息主题行和消息体的伪英语样式的内容。

例如，假定测试代理需要创建一个新消息。Lingo 实用程序对象产生以下代码段：

```
Lipsem it quantos surbatton ditchek surbatton deplorem
ventes so surbatton quantos infreteres ipsem aqua sit
ventes. Campus ad surbatton ad infreteres ad novus surbatton
deplorem it delorum novus. Campus quantos ditchek quantos.
Via surbatton novus surbatton. Novato quantos ditchek deplo-
rem novus ditchek ventes.
```

重要的是每次调用 Lingo 时，它都会产生不同的代码段。因此运行消息内容相同的测试时，系统试图优化性能就会失败。

下面是测试代理的全部源代码，它演示了 Lingo 的很多用途。在代理后面有 Lingo 功能的解释。

```
# Lingo_meister.a
# Author: fcohen@pushtotest.com

# Import tells TestMaker where to find TOOL objects
from com.pushtotest.tool.util import Lingo

print
print "================================================="
print "Technique 1: A simple message example"

myLingo = Lingo()

print "Subject:",myLingo.getSubject( 4 )
print "Message:"
print myLingo.getMessage()
print "-end of message-"

print
print "================================================="
print "Technique 2: URL encoded message example"
print
print "If we needed to include a message in an HTTP POST"
print "command the message would need to be URL encoded."
print "Here is a Lingo message that is URL encoded:"
print
print myLingo.getMessageEncoded( 100 )

print
print "================================================="
print "Technique 3: A complete Lingo letter"
print
print "Here is an example business memo using several"
print "Lingo methods."
print
```

```
print "Dear Mr.",myLingo.getSingleCap(),":"
print
print "We have found the manuscript you were searching"
print "for and it includes the following excerpt:"
print

# Next we use a special version of getMessage
# to include a starting capital letter, ending period
# and encoding. The format is:
# getMessage(int size, boolean startCap,
# boolean endPeriod, boolean encoded)

print myLingo.getMessage(15,  1, 1,  0)

print
print "As you can see we are not certain what dialect the"
print "text is written in. Please do your best to let us"
print "know what it means."
print
print "Sincerely,"
print
print myLingo.getSingleCap()

print
print "================================================="
print "Technique 4: Making your own Lingo language"
print
print "Here is the same example business memo from"
print "technique 3 but using French words to form the"
print "Lingo gibberish."
Print

caps = [ "Francois", "Sanjournais", "Berjerack", "Mon", \
"La", "Le", "Incroyable! "]
tokens = [ "devrait","ouvrir","le","cortEge,","entourE", \
"d'une","bonne","partie","de","ses","adjoints","et","des", \
"dEputEs","parisiens","de","gauche.","les","Ecologistes", \
"seront","fortement","reprEsentEs","la","droite","elle,\
","ne","dEpIchera","que","les","quelques","membres", \
"d'on","est" ]

frenchLingo = Lingo( caps, tokens )

print "Dear Mr.",frenchLingo.getSingleCap(),":"
print
print "We have found the manuscript you were searching"
print "for and it includes the following excerpt:"
print

# Next we use a special version of getMessage which
# has options to include a starting capital letter,
```

```
# ending period and encoding. The format is:
# getMessage(int size, boolean startCap,
# boolean endPeriod, boolean encoded)

print frenchLingo.getMessage(15,  1,  1,  0)

print
print "As you can see we are not certain what dialect the"
print "text is written in.  Please do your best to let us"
print "know what it means."
print
print "Sincerely,"
print
print frenchLingo.getSingleCap()

print
print "Agent done."
```

Lingo_meister 是一个简单的代理，它显示了 TestMaker 的 TOOL 中提供的 Lingo 对象的功能。Lingo 提供了类似英语的乱语，类似英语的乱语很适合于已发出的测试消息的内容。Lingo 对象通过 import 命令来创建，该命令将 Lingo 对象标识到 Jython 脚本语言中。

```
from com.pushtotest.tool.util import Lingo
```

实例化 Lingo 对象后，就可以通过访问它的很多方法来使用它。

```
myLingo = Lingo()
```

getSubject()函数返回以回车字符终止字符串的乱语单词的字符串。在 getSubject()中的可选值标识了结果中无用单字的数目。

```
print "Subject:",myLingo.getSubject( 4 )
```

getMessage()函数返回乱语字单词落的随机数。每个段落以两个回车字符结束。

```
print myLingo.getMessage()
```

在 HTTP/HTML 环境中工作时，到主机的请求必须被 URL 编码。在 URL 编码中，<and> 字符将使 HTML 代码无效，因此它们被编码到&lt 和&gt 中；主机将解码这些代码。getMessageEncoded()函数返回一个 URL 编码的乱语单词段落组。

```
print myLingo.getMessageEncoded( 100 )
```

Lingo_meister 中的第三个示例给出了一个 Lingo 对象如何用于打印命令中，以便生成样例便笺信(memorandum letter)。

```
print "Dear Mr.",myLingo.getSingleCap(),":"
```

getSingleCap()方法返回单个大写乱语单词。便笺的主体使用了一个特殊版本的 getMessage()函数，它的选项包含了起始大写字母、结束周期和编码。该特殊方法的格式为：

```
getMessage(int size, boolean startCap, boolean endPeriod,\
boolean encoded)
```

Lingo_meister 中的最后一个技术给出了代理如何定义它自己的乱语单词字典。Lingo 需要一个大写单词和小写单词的列表。当代理实例化一个新 Lingo 对象时，这些单词会被传递给 Lingo。

```
frenchLingo = Lingo( caps,  tokens )
```

现在，Lingo 从大写和小写单词的新列表中选择出了一些乱语单词。

8.7 本章小结

本章讨论了系统状态，包括可重复的生命周期。生命周期定义了测试的准备、安装、执行和分析阶段。本章给出了 TestMaker 测试代理脚本的示例，在测试前该示例使用数据库系统资源初始化系统的状态。

第 9 章中将介绍在 Microsoft .NET 环境中和安全环境中如何使用智能测试代理。

第 9 章　集成.NET Web 服务

Microsoft 提供了一个广泛的可理解的解决方案，企业利用该方案可以创建、部署和运行基于 Internet 的 Web 服务。Microsoft 支持很多流行的开放 Internet 协议，甚至使得它们能够提供与非 Windows 系统的互操作技术。随着时间的推移，Microsoft 构建了 Windows NT 和 XP 技术之上的企业解决方案。在.NET 出现之前，Microsoft 将所有的企业解决方案技术都缩写为 DNA，即分布式网络体系结构(Distributed iNternet Architecture)。那么，什么是.NET 呢？

.NET 是一个产品策略，它革新了 Microsoft 开发环境和工具并将它们提升到一个新的层次，方便了使用，并且比以前的 Microsoft 开发工具更具有互操作性。在.NET 里，Microsoft 在面向对象的开发工具中添加了一个新的公共语言运行库(Common Language Runtime，CLR)，并在 Microsoft 的所有产品中添加了 XML 技术。

Microsoft 和欧洲计算机制造商协会(ECMA)合作推出了 CLR(TG3)开放标准和 C#语言(TG2)。ECMA Web 站点上保存着这些文档：http://www.ecma-international.org/。其他的信息位于 http://www.dotnetexperts.com/ecma/index.html 和 http://msdn.microsoft.com/net/ecma/。

.NET 定义了 Microsoft 中间语言(Intermediate Language，IL)代码。CLR 是 IL 运行库虚拟机环境，它提供了内存管理、自动垃圾回收、安全和线程。CLR 使得 Microsoft 的开发工具支持各种面向对象语言，包括 C、 C++、 VB.NET、 C#和 J#。简单的 CLR 运行库能运行内置于这些可以输出 IL 的语言的组件。下面是在 Microsoft Visual Studio .NET 中编写的一个 C#和 C++程序的示例。Visual Studio .NET 将这个程序编译成 CLR 运行的单个单元。

```
# We begin with a method defined using C# ...
class Vector
{
   /// <summary>
   /// Compute the length of the vector
   /// </summary>
   public double Length
   {
      get
      {
         return(Math. Sqrt(x * x + y * y));
       }
    }
}
#  ... then add some C++ code to the mix ...
unsafe static int AddArrayUnsafe(int[] numbers)
{
    int result = 0;
    for  (int it = 0;  it < Iterations;  it++)
    {
        fixed (int* pNumber = numbers)
        {
```

```
            int* pCurrent = pNumber;
            int* pLimit = pNumber + numbers.Length;
            while  (pCurrent < pLimit)
            {
                result += *pCurrent;
                pCurrent++;
             }
          }
      }
return result;
}
```

这个示例代码显示了.NET 机制在 C#的不同存储管理安全环境中管理 C++指针的方式。同样的机制在其他的 CLR 兼容语言中也起作用。

在许多方面，.NET 是 Microsoft 使用 XML 的最大动力。XML 使得 Microsoft 开发工具能够提供标准化的配置、描述和部署定义。例如，当 Microsoft BizTalk Server、 SQL Server 和 Exchange Server 仍旧使用 Windows 注册表时，他们也使用 XML 来配置和管理功能。Microsoft 产品策略将 XML 构建到它的所有技术中，其方式是将配置信息编码成 XML 格式，并存储到文件和数据库中。XML 也成为 Microsoft 的公共语言。实际上，软件开发人员期望任何 Microsoft API 都可以作为基于 SOAP 的 XML 接口是不合理的。

.NET 方便了 IT 管理员、软件开发人员和 QA 分析师：

- 因为 XML 的原因，配置和管理文件变得更加开放和标准化。第三方的系统管理解决方案，如 HP 的 OpenView 和 IBM 的 Tivoli，能够更加容易和迅速地支持 Microsoft 服务器。
- QA 分析师发现测试.NET 应用程序也比测试以前的 Microsoft-built 应用程序容易，因为这些应用程序的 API 自描述了有效数据。

如果开发人员、IT 管理员和 QA 分析师提供使用 Microsoft 开发工具创建的产品，Microsoft 采用 XML 并提供单个运行库引擎，就能够增加生产率和效率。当然，像所有的新技术一样，.NET 产品也存在问题，需要被理解和测试。

9.1 互操作性和集成

使用.NET 开发系统不会出现大的故障，这对于 Java 开发人员来说，无疑是个好消息。在大多数地方，使用.NET 的系统都提供了很好的与基于 Java 的系统的互操作性。对于 Java 开发人员来说，较大的问题通常是集成。

考虑一个示例，一个产品供给公司提供一个 Web 服务来处理定购单。服务接受 SOAP 编码的订单。

```
<order_request>
   <name>DemandLine Purchasing,  Inc.</name>
   <product>123456</product>
   <price>$59.95</price>
</orderrequest>
```

图 9-1 描述了一个使用.NET 编写的 Java 应用程序如何使用这个购买服务。

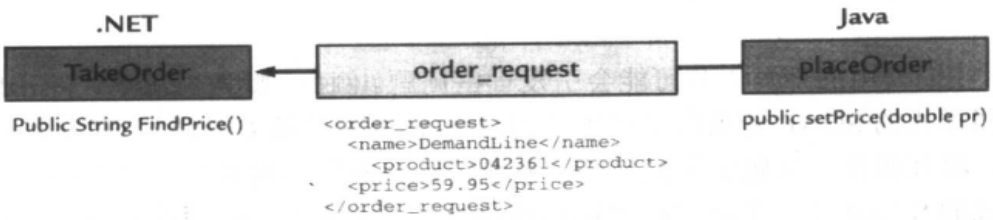

图 9-1　Java placeOrder 函数对已发布的 Web 服务的 TakeOrder 方法发出一个 SOAP 请求

Visual Studio .NET 使得编写和公开本服务都很容易。利用通常的 Java 开发工具很容易编写 placeOrder 方法来调用 Web 服务。这个示例存在 3 个集成问题：

1. Java 客户函数使用双精度浮点变量来保存价格，而.NET 函数使用字符串。

2. 不论 Java 客户还是.NET 服务都不显式地知道价格元素的货币表示方式。例如，Java 客户可能发送美元表示的价格，而.NET Web 服务可能期望为欧元。

3. 对于那些起始字母为大写字母的方法名(通常保留为操作系统调用)，Java 将对它们使用命名约定。另一方面，.NET 的约定是使用大写字母定义所有的方法名。这并不像其他两个问题那样棘手，但也会成为 Java 和.NET 开发人员的燃眉之急。

这些实际上不是真正的互操作性问题，它们是系统集成问题，通常我们每天都不得不处理这些问题。

XML 和 SOAP 的.NET 支持提供了问题 1 和 2 的解决方案。通过使用 XML 模式，.NET Web 服务提供了请求方法的数据类型。下面的 XML 模式定义了字符串类型的价格元素。

```
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/
XMLSchema">
   <xs:element name="order_request">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="name"  type="xs:string"/>
            <xs:element name="product"  type="xs:int"/>
            <xs:element name="price"  type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

.NET 语言使用一个通用工具集来发布 Web 服务，因此您可以期望找到该服务的一个 WSDL 文档。在这个示例中，Java 开发人员需要实现一个符合模式的请求。这很可能意味着，需要在请求中修改客户端的代码，以将价格表示为字符串，并且需要将响应解码为一个浮点双精度数值。

对于您可能遇到的其他与 SOAP 相关的问题，请参考第 7 章，包括数据类型互操作性、对标准的坚持和底层平台差异等。本章的后面部分将讨论更多的、关于 WSDL 的.NET 使用。

9.2 .NET 的不同之处

本质上，我们使用的开发工具可能会引发一些问题。例如，默认情况下，Visual Studio .NET 通过使用文档样式的 SOAP 请求自动产生访问给定 Web 服务的 C#代码。开发人员仅依靠自动的开发工具，将互操作性问题引入到他们的系统中，并不知道问题发生时在哪里寻找。

文档样式的 SOAP 是一个明智的体系结构决策，正如我们在后面将看到的一样。但是，从测试角度来看，如果您的.NET Web 服务使用文档样式的 SOAP 编码(在 ASP.NET 中默认)，这样和其他的仅使用 RPC 编码的 SOAP 通话的 SOAP 客户就存在互操作性问题。

第 7 章介绍了各种 SOAP 请求和响应。两种最流行的样式是 RPC 和文档。大多数开发环境都包含一些实用程序，用于检查源代码文件中的方法声明，其中源代码文件是用来定义对象的。然后它们自动创建所需的代码，将对象的方法作为 Web 服务来提供。局部对象调用运行在远程服务器上的特定方法和对象，并传递在方法声明中定义的参数。例如，下面的 C#代码被自动部署为基于 SOAP 的 Web 服务。

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace myService
{
      public class myService: System. Web. Services.WebService
      {
               public HelloService()
               {
                 //CODEGEN: This call is required by
                 //the ASP.NET Web Services Designer
                       InitializeComponent();
                }

                [WebMethod]
                public string greetingService
                (String strMyName)
                {
                   return "Hello,  "+ name +",  \
                   and welcome to my service.";
                 }

                [WebMethod]
                public string SayGoodBye()
                {
                        return "Goodbye!";
                }
      }
}
```

这段代码被作为一个 Web 服务，它具有一个 greetingService 请求接口。SOAP 调用接受一个字符串作为参数，并在其返回值中响应字符串。

RPC 样式的 SOAP 请求很受开发人员欢迎，因为产生 SOAP 接口和 WSDL 描述的 Web 服务工具为开发人员做了大量的工作。例如，下面的方法返回一个复杂数据类型。

```
public class myComplexService{
   public Hashmap greetingService( firstname name )  {
         Hashmap ht = new Hashmap();
         ht.put(  "Event", name );
         ht.put(  "Keycode",  keycode );
         ht.put(  "PcAssemblyNumber", pcassembly );
         ht.put(  "SoftwareDottedString",  dottedsw );
         ht.put(  "CurrentTime", new Date()  );
         ht.put(  "CommandKey",  "7"  );
         return ht;
```

在一个 RPC 样式的 SOAP 请求中，开发人员可以依靠一个开发工具来构建 SOAP 请求代码，该 SOAP 请求代码将一个 Hashmap 对象编组到 XML 中，而 XML 被发送到一个对服务器的 SOAP 请求中。这个代码也使用定制的串行化器来转化对象，例如将 Hashmap 转化为 XML。

文档样式的 SOAP 请求始于已定义的 XML 文档。文档样式的 SOAP 请求不依赖于开发工具，而是通过在应用程序代码中填充 XML 树来构建。

```
public class myComplexService{
  public void callGreetingService( firstname name )  {
        body = new SOAPBody();
        body.addElement (  "Event"   name );
        body.addElement (  "Keycode", keycode );
        body.addElement (  "PcAssemblyNumber", pcassembly );
        body.addElement (  "SoftwareDottedString",dottedsw);
        body.addElement (  "CurrentTime", new Date()  );
        body.addElement (  "CommandKey", ""  );
        ...
        SOAPRequest.send( body );
```

.NET 是一个文档样式的环境。默认的.NET 开发工具创建文档样式的 SOAP 请求。虽然.NET 工具提供了一个使用 SOAP RPC 调用的开关，但是.NET 对 RPC 的支持很少。如果一个应用程序对.NET Web 服务发出一个 SOAP RPC 调用，而该 Web 服务期望一个文档样式的请求，那么.NET 会返回一个错误——Web 服务异常。

Microsoft 对文档样式 SOAP 的选择成为开发人员追求的选择。很多 Web 服务开发人员来自于由 CORBA、RMI 和 COM 领导的组件软件世界。SOAP RPC 是连接组件软件体系结构和 Web 服务的桥梁。另一方面，业务到业务(business-to-business—B2B)交换基础架构从一开始就是基于 XML 的，因此也自然具有文档样式的 SOAP。

下面看一个生动的示例，该示例描述了 J2EE 中的 Web 服务开发人员和.NET 阵营间的分歧。可以到 XMethods.net 上查看已发布的公共可用的 Web 服务列表。这个列表有两个特征：SOAP RPC Web 服务是用 Java 编写的；文档样式的 Web 服务是用.NET 编写的。

RPC 和文档样式的 SOAP 调用都需要考虑的是，SOAP 标准不能覆盖比数组和结构更复杂的数据元素。在前面的示例中，我们可能已经发现，对于 Hashmaps，Microsoft 具有自己的编码方式，这种编码方式和其他的 Web 服务平台不能互操作。开发人员不得不编码数组的数组。

我们已经看到 SOAP RPC 和文档样式的 SOAP 之间在样式上的差别。SOAP 的实现存在着可扩展性和性能问题，因此要选择是使用 RPC 还是选择文档样式的 SOAP。

9.2.1 文档样式的可扩展性

不考虑 SOAP 调用样式，每个 SOAP 消息都经过相同的创建阶段。创建过程始于对象中存储的一个值。当这个值被传递给一个 Web 服务时，该对象就被转换为一个文档对象模型元素，并存储在存储器中。当请求被编组，以发送给 Web 服务器主机时，元素被向上舍入(round up)并被串行化为 XML 格式的字符串。然后最后一步收集 XML 格式的字符串，并将组合的组通过一个传输装置发送到 Web 服务主机。图 9-2 演示了这个过程。

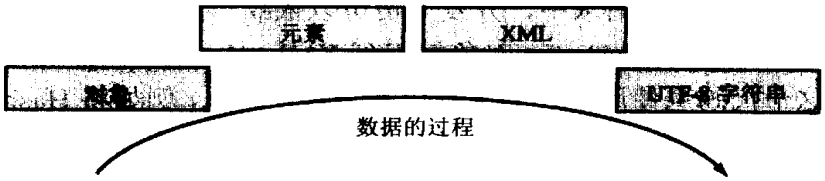

图 9-2 SOAP 请求以一个对象中的数据开始，对象被转换为 DOM 元素，然后将元素串行化为 XML 字符串。最后，它们组合所有的字符串，并通过一个传输装置将组合后的组发送给 Web 服务主机

图 9-2 描述了 4 个可能产生可扩展性和性能问题的地方。对象到元素的转换可能产生 CLR 表示数据方式的问题。元素到 XML 的转换可能会产生问题，因为很多 XML 语法分析器的性能都是不充分的。XML 到 UTF-8 转换的串行化器可能产生 bug 和性能问题。还有很多其他问题也会降低系统的可扩展性和性能。

在测试 Web 服务应用服务器时，早期的系统存在很大的可扩展性和性能问题，尤其是在比较 SOAP RPC 和文档样式的 SOAP 时。这些测试的结果包含了下面的几个方面：

- 文档样式的 SOAP 操作胜过 SOAP RPC。
- 随着请求和响应的有效载荷大小的增加，SOAP RPC 存在明显的可扩展性问题。

表 9-1 给出了一个流行的应用服务器上 TPS 结果的比较。

表 9-1 TPS 结构的比较

有效载荷大小(单位字节)	RPC	文　档
600	168	410
3000	145	399
9000	100	404
2400	49	411
57000	18	401
96000	10	406

该测试测量了在客户端处从发出一个请求到接收到一个响应的往返时间。为了在您的环境中运行这个测试，测试代码和其他很多可用的资料可以免费通过 http://www.pushtotest.com/ptt/kits/index.html 下载。

注意在每一种情况下，SOAP RPC 事务的 TPS 值都比文档样式的调用慢。另外，文档样式的 TPS 值在增加有效负荷大小时，仍保持稳定，而 SOAP RPC TPC 值却明显下降。

根据这些测试，并不能推出文档样式的 SOAP 比 SOAP RPC 好，或是相反。但是，它们证明了 SOAP 实现的测试具有很多缺陷和局限。当有可用的更好的新实现时，就需要新的 SOAP 堆栈的可扩展性和性能测试来理解性能特征是如何变化的。

9.2.2 .NET 中的 SOAP 头

在 Web 服务的形成时期，Microsoft 是 XML 模式、WSDL 和 SOAP 头的最早支持者。其他的 Web 服务平台很少令人信服。因此对于 WebSphere、 Apache SOAP 和 BEA Weblogic 的开发人员来说，使用 Microsoft 产品创建的 Web 服务中，WSDL 文档的 XML 模式和 SOAP 头的内容，可能会出现问题。在第 4 章，通过一个通用 SOAP 值和格式的解释最初引入了 SOAP。本节将介绍 Microsoft 如何处理 SOAP 头。

在介绍.NET 如何使用 SOAP 头之前，您可能想先回顾一下 SOAP 头元素的定义。SOAP 1.1 规范(通过 http://www.w3.org/2000/xp/Group 可以找到该规范)和 SOAP 1.2 草案指出，一个 SOAP 消息具有 3 个可能的元素：顶层 Envelope 元素和两个子元素——Header 和 Body。下面是一个 SOAP 消息的示例，它具有以上 3 个元素：

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Header>
      <PriorityHeader
         xmlns="http://msdn.microsoft.com/AYS/6/2002/">
         <Priority>HighPriority</Priority>
      </PriorityHeader>
   </soap:Header>
   <soap:Body>
      <PlaceOrder
         xmlns="http://msdn.microsoft.com/AYS/6/2002/">
         <quantity>100</quantity>
      </PlaceOrder>
   </soap:Body>
</soap:Envelope>
```

Body 元素包含了请求的主要数据，Header 包含了 Body 元素的所有元数据。Header 元素是可选的，然而它在.NET 应用中使用的很广泛。随着时间的推移，新的 Web 服务标准将会更多的利用 Header 元素，包括即将产生的安全、信任壮、路由和提名的标准。.NET 也使用了很多的 Header 元素。

SOAP 规范定义了 3 个可选的属性，它们能够应用于头块：encodingStyle 属性、actor 属性和 mustUnderstand 属性。虽然 encodingStyle 属性是可选属性，但是.NET 服务器和 SunONE 应用服务器期望 encodingStyle 属性出现。

SOAP 像 servlets 和其他中间件一样，经历了相同的变革。虽然单个独立的对象能够响应一些 Web 服务请求，但新的体系结构实现了几个较小的对象，每一个对象处理 Web 服务请求的一个方面，并且它们被连接在一个关于处理程序(handler)的字符串中。当 SOAP 消息遍历处理程序链时，SOAP 的 Header 元素由单独的.NET 程序处理。

SOAP 消息已经被定义为 XML 格式，因此不像 HTTP 请求中的头，SOAP 头不必描述 SOAP 消息中的数据。SOAP 头，尤其是在.NET 里，帮助处理体中的数据。例如，在银行业务环境中，一个 SOAP 头传输认证信息和事务信息来鉴别发送 SOAP 消息的人的身份。另一个示例是航空预定服务，SOAP 头在验证 SOAP 消息中飞机票价的有效性上，定义了一个时间限制。

当 SOAP 规范提供任何有效的 XML 数据作为 SOAP 头时，Microsoft 实际上抬高了 SOAP 头的作用。因此，可以放心的假设，在.NET Web 服务中，SOAP 头将在提高可扩展性和性能方面扮演着重要的角色。

很多时候，一个应用程序中的 SOAP 头被该应用程序所专用。但是在.NET 中，有些情况下 SOAP 头需要被所有的 Web 服务处理。例如，考虑一个登录服务，它对单个服务器所处理的所有 Web 服务请求是公用的。因为这个原因，.NET 提供了一个公共库，称为 SOAP Extensions。您所注册的对象会在 SOAP 消息串行化的前后被调用。虽然对于.NET 开发人员来说，这是注册处理程序的一个便利方法，但是它也容易引入性能瓶颈和可扩展性问题。在其他的平台中，您需要查看串行化器来发现性能问题。另外，在.NET 中，您也需要查看处理 Web 服务请求和响应的 Soap Extensions。

当.NET 使用 WSDL 定义 SOAP 头时，.NET Soap Extensions 机制会产生副作用。考虑到安全因素，.NET 开发人员不想通过发布的 WSDL 来部署可用的 Soap Extensions 信息。因此，一些.NET 开发人员会忽视 WSDL 的头定义，并手工将头添加到他们的请求中。其他人会关闭 WSDL 生成器，手工编辑一个静态的 WSDL 文件，以包含头信息。这些技术能很好地检查配置问题，它们应当被正规地测试和监测。

SOAP 头可能会成为测试 Web 服务的梦魇。更坏的情况下，SOAP 头可能会成为与一个 Web 服务相关的所有事物的垃圾堆积场。创建智能测试代理来理解和验证 SOAP 头是测试和监测 Web 服务的一个很重要部分。

9.2.3 WSDL .NET 样式

第 7 章将 WSDL 描述为一个基于 XML 的文档格式，用于描述 Web 服务提供的服务，因此工具可以访问和使用它们。服务的 WSDL 描述了服务的位置、可用服务的名称、它们的方法名、参数和类型，以及返回值。Microsoft 在很多.NET 开发工具中都认可 WSDL 标准，这些工具都使用 WSDL 来描述 XML 数据的一个 SOAP 接口。

Microsoft 在描述 SOAP 接口方面，具有两个额外的成就：Web 服务元语言(Web Services Meta Language，缩写为 WSML)和 Web 服务发现工具(Web Services Discovery Tool，Disco)。WSML 是另一种由某些 Web 服务工具(Microsoft SOAP 工具包)使用的基于 XML 的文档格式，它提供“Web 服务方法与它们的参数”和“提供了服务的底层代码对象(例如一个 COM 组件)”之间的映射。

Disco 是一种 XML 文档格式，它包含了描述可用 Web 服务的 WSDL 资源的入口指针。Disco 为 IT 管理员提供了一种集中多个 Web 服务提供者的列表的方式。Disco 的所有内容都位于单个文档中，工具能够使用该文档来选择所需要的 Web 服务，并获得对该服务的引用。

当平台供应商在它们的实现中试图扩展(stretch)、缩减(strain)，甚至改变(bully)一个标准时，作为测试者，我们需要识别出来。WSML 和 Disco 是这种行为的示例。即使这种努力的结果可能并没有成为标准的一部分，但对于测试基于.NET 的 Web 服务来说，对标准的 Microsoft 扩展将成为您的测试计划的一部分。考虑 Microsoft 如何将它的痕迹烙印在编写.NET 服务的 WSDL 描述的风格中。

为了演示在一个 Web 服务的.NET WSDL 描述中所发现的不同点，考虑 Service Objects 公司的一个.NET Web 服务，该服务位于站点 http://www.servicesobjects.com 上。Services Objects 公司提供了几个由站点和门户使用的服务，以为站点的浏览者提供个性化和感兴趣的内容。下面的示例集中描述了 GeoCash 服务。GeoCash 接收一个 U.S.邮政编码作为参数，并且返回该邮政编码所代表地区附近的自动柜员机(ATM)列表。Service Objects 利用 Web 站点、协作门户和应用程序供应商，将这种服务出售给人们。

GeoCash 服务的 WSDL 完整定义位于 http://ws.serviceobjects.net/gc/GeoCash.asmx?WSDL。我第一次接触 WSDL 文档时，眼花缭乱、脑袋发晕。我竭尽所能地理解它。浏览 WSDL 和查看 Web 服务花了我大约六个月的时间。下面给出了这个 WSDL 文档的各个片断，以便您熟悉 GeoCash 服务。

第一个提示是，这是最后一节中的一个.NET Web 服务。

```
<binding name="GeoCashSoap" type="s0:GeoCashSoap">
  <soap:binding transport =
  "http://schemas.xmlsoap.org/soap/http"
  style="document" />
   <operation name="GetATMLocations">
      <soap:operation soapAction =
      "http://www.serviceobjects.com/GetATMLocations"
      style="document" />
      <input>
        <soap:body use="literal"  />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
   </operation>
</binding>
```

在上面的代码段中 GeoCashSoap 绑定定义了一个文档样式的 SOAP 请求和响应，它们使用了字面量编码(literal encoding)。字面量编码通知 SOAP 堆栈将 SOAP 请求和响应体当作已经格式化的 DOM 树，并立即将它串行化为 XML。

另一个提示是，创建这个 WSDL 文档的.NET 是<types>分支。

```
<types>
  <s:schema elementFormDefault="qualified"
  targetNamespace="http://www.serviceobjects.com/">
    <s:element name="GetATMLocations">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
```

```
            name="strInput" type="s:string"  />
            <s:element minOccurs="0" maxOccurs="1"
            name="strLicenseKey, type="s:string"  />
          </s:sequence>
        </s:complexType>
      </s:element>
```

.NET 使用 XML 模式为请求和响应的参数定义类型信息。类型元素给出了有效的参数，以允许发出服务的请求。在这个示例中，请求使用两个字符串值来存储邮政编码和一个 License 密钥。我们直接从 Service Objects 处获得 License 密钥。

下一组类型定义定义了出现在响应文档中的一个复杂数据类型。响应文档包含了一个 GetATMLocationsResult 元素……

```
<s:element name="GetATMLocationsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
      name="GetATMLocationsResult"
      type="s0:ATMLocations"  />
    </s:sequence>
  </s:complexType>
</s:element>
```

该元素包含一个 ATMLocations 元素……

```
<s:complexType name="ATMLocations">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
    name="ATM" type="s0:ATM" />
    <s:element minOccurs="0" maxOccurs="1"
    name="Error" type="s:string"  />
  </s:sequence>
</s:complexType>
```

而 ATMLocations 元素包含一系列的 ATM 元素。每个 ATM 元素包含关于每个银行的信息，包括地址、位置和有关 ATM 机的说明。

.NET 创建的 Web 服务的一个很方便的特征是，一个友好的、能够描述和访问 Web 服务的 HTML 前端。甚至根据 GeoCash 服务的 WSDL 描述，大多数时候都能够很方便地得到该服务的 HTML 描述。关于 GeoCash HTML 描述的示例请访问 http://ws.serviceobjects.net/gc/GeoCash.asmx?op=GetATMLocations。

9.3 .NET 环境的测试代理

下面将介绍一个 TestMaker 测试代理，该代理能够向 GeoCash 服务发出一个文档样式的.NET Web 服务请求。这个代理的很多值可以在 GeoCash 服务的 WSDL 文档中找到，该服务已在本章的前面一节中介绍过。下面是这个代理的全部源代码及其工作原理的解释。

```
# GeoCash_agent.a
# Author:  fcohen@pushtotest.com

# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  SOAPHeader,  SOAPBody,  SOAPProtocol
from com.pushtotest.tool.response import Response

# This agent also uses JDOM APIs to handle XML data
from org.jdom import Document,  Element, JDOMException,\
Namespace, DocType
from org.jdom.output import DOMOutputter, XMLOutputter
from org.jdom.input import SAXBuilder
from java.io import StringReader

# Variables definitions

zipcode = "95008"

license = "0"

host = "ws.serviceobjects.net"
port = 80
path = "gc/GeoCash.asmx"

# Helper function to build a request XML document

def addElement( top, name,  content, ns ):
    newelement = Element( name, ns )
    newelement.addContent( content )
    top.addContent( newelement )

# Main body of agent

print "Processing..."
print

# Set-up a SOAPProtocol object and populate it with the
# info to make a request to the GeoCash service

soap = ProtocolHandler.getProtocol("soap")

soap.setHost( host )
soap.setPath( path )
soap.setPort( port )

body = ProtocolHandler.getBody("soap")

body.setMethod("GetATMLocations")

soap.setActionURI(  "http://www.serviceobjects.com/GetATMLo-
```

```
cations" )

# Build the request XML document

xmlnsl = "http://www.serviceobjects.com/"

# Create the request document by first creating
# the Respond element ...
elOne = Element( "GetATMLocations", xmlnsl )

# ... add the input elements ...
addElement( elOne, "strInput", zipcode, xmlnsl )
addElement( elOne, "strLicenseKey", license, xmlnsl )

# ... then tell the SOAPProtocol handler about the request
document.
doc = Document( elOne )
body.setDocument( doc )

# Finally, attach the body object to the soap protocol han-
dler object
soap.setBody(body)

print "Here is the XML request document:"
xo = XMLOutputter()
print xo.outputString( doc )
print

# Send the request to the .NET host and receive the response

response = soap.connect()

# Parse through the responses

print "Here is the raw response:"
print response
print

builder = SAXBuilder()
doc = builder.build( StringReader(response.toString() ) )

print "Here are the elements in the response:"
print

envelopeElement = doc.getRootElement()

# Document-style SOAP responses will use the
# document-literal type of namespace for the Body element
soap_ns = Namespace.getNamespace \
("soap", "http://schemas.xmlsoap.org/soap/envelope/.)

bodyElement = envelopeElement.getChild("Body", soap_ns)
```

```
# The entries in the body use the Service Objects namespace
# which also tells us which .NET service responded
so_ns = Namespace.getNamespace( " ",  "http://www.serviceob-
jects.com/")

atmlocrespElement = bodyElement.getChild( \
"GetATMLocationsResponse",  so_ns )
print "atmlocrespElement=",atmlocrespElement

atmlocresultElement = atmlocrespElement.getChild( \
"GetATMLocationsResult",  so_ns )
print "atmlocresultElement=",atmlocresultElement

# Notice that GetATM
atmlist = atmlocresultElement.getChildren(  "ATM",  so_ns )
print "atmlist=",atmlist
print

for i in atmlist:
    print  " "  ,i.getName()
    print  " "  ,i.getChildText("Bank",  so_ns)
    print  " "  ,i.getChildText("Address",  so_ns)
    print  " "  ,i.getChildText("Location",  so_ns)
    print  " "  ,i.getChildText("City",  so_ns),  \
    i.getChildText("State",  so_ns),  i.getChildText("Zip",\
    so_ns)
    print "   Longitude:",i.getChildText("Longitude",  \
    so_ns)," Latitude:",i.getChildText("Latitude",  so_ns)
    print  " "  ,i.getChildText("Notes",  so_ns)
    print

# All done

print
print "Agent ended."
```

TestMaker 运行这个代理。该代理连接到 Service Objects 主机，并请求 California Campbell 处的 ATM 位置。响应应当包含 Campbell 的 10 个不同位置处的 ATM 机器。让我们看看下面的代码以了解这个代理是如何工作的。

```
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler,  SOAPHeader,  SOAPBody,  SOAPProtocol
from com.pushtotest.tool.response import Response
```

首先导入来自 TestMaker 的 TOOL 的 SOAPProtocol 处理对象。

```
from org.jdom import Document,  Element, JDOMException,\
Namespace, DocType
from org.jdom.output import DOMOutputter, XMLOutputter
from org.jdom.input import SAXBuilder
from java.io import StringReader
```

我们也导入 JDOM 对象来构造和分析 XML 数据。JDOM 是一个操作 XML 数据的简单接口。

```
zipcode = "95008"
license = "0"
```

上面的变量定义 U.S.邮政编码和许可证号，它们将被发送到 Web 服务主机。Service Objects 提供许可证，以将服务提供给那些付费客户。为了支持本书，Service Objects 提供了专用的许可证号，用于测试本章中提供的代理。请不要滥用这个特权，多次使用这个许可证号或创建大量的对 Service Objects 主机的请求。

```
host = "ws.serviceobjects.net"
port = 80
path = "gc/GeoCash.asmx"
```

这些变量定义了目标服务器和该 Web 服务的位置。我们在本章前一节的 WSDL 定义中已学习了这些值。

```
def addElement( top, name,  content, ns ):
    newelement = Element( name, ns )
    newelement.addContent( content )
    top.addContent( newelement )
```

addElement 是一个辅助函数，它将一个 DOM 元素作为参数，并在其上添加一个新的子元素。这是一个方便的函数，它使得后续的 addElement()命令更清晰。

```
soap = ProtocolHandler.getProtocol("soap")
body = ProtocolHandler.getBody("soap")
```

TestMaker 中的 TOOL 提供了一个 SOAPProtocol 处理程序对象和一个 SOAPBody 对象。我们使用它们来构造请求文档，并和主机进行通信。

```
body.setMethod("GetATMLocations")
soap.setActionURI( \
"http://www.serviceobjects.com/GetATMLocations" )
```

WSDL 文档告诉我们，对于一个给定的邮政编码，使用 GetATMLocations 方法可以请求 10 个 ATM 位置的列表。

```
xmlnsl = http://www.serviceobjects.com/
```

上面的代码引用了请求中参数的命名空间。在文档样式的 SOAP 请求中，TestMaker 使用第一个元素<Respond>的命名空间来确定目标服务器名称。最终获得对正确服务器的请求。第一个元素的命名空间使得我们可以得到正确的 Web 服务。

WSDL 告诉我们，GeoCash 需要一个邮政编码和一个许可证元素。这个代理使用 JDOM API 来创建 DOM 对象中的 XML 请求。

```
elOne = Element(  "GetATMLocations", xmlnsl )
addElement( elOne,  "strInput",  zipcode, xmlnsl )
addElement( elOne,  "strLicenseKey",  license, xmlnsl )
```

在上面的代码中，代理使用 addElement 方法来添加两个请求元素。

```
doc = Document( elOne )
body.setDocument( doc )
soap.setBody(body)
```

接着代理将 DOM 对象 doc 传递给 SOAPBody 对象，以对服务器进行 SOAP 调用。

```
response = soap.connect()
```

上面的 connect()方法发送对 Service Objects 主机的请求，并返回一个 response 对象。这个 response 对象使用 XML 格式。代理使用 JDOM API 来查询响应文档，以查找和验证 ATM 的位置数据。

```
builder = SAXBuilder()
doc = builder.build( StringReader(response.toString()  )  )
```

使用 JDOM SAXBuilder 对象创建一个新的 DOM 对象，来分析我们从.NET 主机接收到的 XML 数据。回忆一下，响应文档的 WSDL(http://ws.serviceobjects.net/gc/GeoCash.asmx?WSDL)告诉我们，响应 XML 应当具有如下形式：

```
Envelope
   Body
     GetATMLocationsResponse
       GetAMTLocationsResult
         ATM
         ATM
         ...
```

看一下代理如何使用 JDOM API 来分析响应文档的内容。

```
envelopeElement = doc.getRootElement()
```

首先，我们找到根元素。通常，每个 SOAP 响应文档将根元素命名为 Envelope。

```
soap_ns = Namespace.getNamespace("soap",  \
"http://schemas.xmlsoap.org/soap/envelope/")
```

这个代理创建了一个 Namespace 对象，它用于获取 Body 元素。SOAP 响应的 Body 被编码为文档-字面量(document-literal)样式，如 XML 模式定义在 http://schemas.xmlsoap.org/soap/envelope/处定义的那样。

```
bodyElement = envelopeElement.getChild("Body",  soap_ns)
```

在此我们使用文档-字面量样式的命名空间来获取 Body 元素。

```
so_ns = Namespace.getNamespace(  " ",  \
"http://www.serviceobjects.com/")
```

代理创建了第二个 Namespace 对象，来获取 Service Objects 主机发出的响应文档。

```
atmlocrespElement = bodyElement.getChild( "GetATMLocations-
Response", so_ns )
atmlocresultElement = atmlocrespElement.getChild( \
"GetATMLocationsResult", so_ns )
```

代理向下跳转到两个嵌套子元素，GetATMLocationsResponse 和 GetATMLocationsResult。

```
atmlist = atmlocresultElement.getChildren( "ATM", so_ns )
```

现在我们跳转到 GetATMLocationsResult 元素，它的子元素是一个 ATM 元素列表，一个 ATM 元素包含了一个售卖机的信息。

```
for i in atmlist:
      print " ",i.getName()
      print " ",i.getChildText("Bank", so_ns)
```

代理使用了一个 for/in 语句，来循环处理所有的子元素。每次循环时，变量 i 指向下一个 ATM 元素。

这个代理只是一个开始、一个框架。一个扩展的代理能够发出多个可变大小和延迟的请求；能够操作请求脚本的多个并发线程，以增加服务器的负载；能够在多台机器上运行多线程代理的多个副本；能够向代理添加简单的日志记录函数，这样我们就可以确定 TPS 和 SPI 值。第 11 章和第 12 章中介绍了使用智能测试代理创建完整的测试系统的详细示例。

9.4 近期需要考虑的事项

在将来的很长一段时间里，.NET 将是出售中的新生儿。因此当设计和测试 Web 使能应用程序时，需要熟记一些近期要考虑的事项。

首先要考虑解决可互操作性问题对于不同的平台是不同的。当一个开发人员面对 Java 平台的互操作性问题时，通常能够通过获得底层代码来发现并解决问题。Microsoft 开发工具、应用程序和操作系统与这些公共审查(Public Scruting)密切相关。

考虑到 Microsoft 最初不开放源码，我发现 Microsoft 向开放源代码迈出的第一步，这令我印象深刻。Microsoft 现在提供了一个“共享资源”的概念，在特定许可下，可以获得他们工具的源代码。详细情况请访问 http://www.microsoft.com/licensing/sharedsource/default.asp 和 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Dndotnet/html/mssharsourcecli.asp。某些 Microsoft 库甚至提供它们源代码的公共视图。例如，SOAP 的 C++ ATM 实现——它并不运行在 CLR 上——可以在 Microsoft 的 Web 站点上找到。

计算机系统、软件应用程序和操作系统之间的互操作性比科学更具有艺术性。实践的越多，互操作性就越好。因此，.NET 的互操作性将随着时间的推移而改善，其前提是 Microsoft 继续改进.NET。Microsoft 定期地参加 SOAPBuilders(一个 Web 服务供应商的松散联盟)，它们的目标是保证互操作性。SOAPBuilders 也发布一个互操作性测试组件，来检查 SOAP 实现。您可以通过 http://www.smethods.com/ilab/和 http://www.whitemesa.lcom/interop.htm 来访问这些组件。

最后考虑 Windows 开发团体正在经历的一个重大变革。Visual Basic 开发人员正从面向脚本的语言转向面向对象的编程语言，例如 C#。学习对象编程通常是一件很困难和长期的任务，这将影响对 Web 使能应用程序的测试。作为一个示例，我参加了 Microsoft 的一个会议，该会议向 Visual Basic 开发人员介绍.NET 开发工具。其中一个陈述展示了.NET 开发工具如何允许 C#编写的程序包含面向脚本的 Visual Basic 命令，该 Visual Basic 命令被插入到 unsafe 标记封装的代码块中。一个 VB 开发人员谈到，关于 C#，他的理解是 unsafe 标记之间的 Visual Basic 代码。

Microsoft 打赌，通过介绍.NET 开发工具如何使得开发人员的产出率更高，能够确保面向脚本的开发人员学习面向对象的编程。转向 VB.NET 的 Visual Basic 开发人员将从现在可用的 Web 服务世界里获取更多的好处。例如，VB.NET 开发人员可以使用 Passport 来取代编写一个安全管理器。

面向对象的开发人员新手编写的 VB.NET 软件通常存在下面几个问题，这些问题可以在测试中检测到：

- **函数产生多个线程**。脚本语言通常为开发人员处理线程。因此线程死锁、并发问题和紊乱情况很容易出现在面向对象的编程中。
- **具有复杂数据类型的方法声明**。面向对象程序中的方法声明能够使用复杂数据类型，这是它的优点之一——例如，一个客户记录包含数字、日期和购买历史——这可能造成过于复杂的 SOAP 请求。
- **边界条件中的异常处理**。面向对象程序遵循一个从一个对象到另一个对象的调用堆栈。对于初学者来说，在抛出异常时展开堆栈常会产生问题。

9.5 本章小结

在本章中，我们学习了 Microsoft 如何采用 XML 并提供单个运行时引擎，将 Microsoft 开发工具提升到一个新的层次中，使得它们更加灵活、生产率以及效率更高。本章覆盖了测试.NET 中内置的 Web 服务的一些问题。

在将来，人们将兴奋地看到 Microsoft 的 CLR 引擎是如何处理产品环境的。今天，真正大型的.NET 可扩展性仍充满了很多未知数。请考虑.NET 位于 Windows XP 技术之上——而 Windows XP 本身又是新技术，以及位于具有很少历史的硬件之上。

另外，人们将兴奋地看到开源组织是如何帮助.NET 成为跨平台的平台。很多人正在致力于创建.NET 开发框架的一个开源实现，例如 Go-Mono。详情请访问 http://www.go-mono.com。

第10章　创建与测试Intranet和安全环境

前面的章节介绍了使用多种互连协议测试 Web 使能应用程序的技术。这些技术假定了测试机与主机之间具有开放和免费的连接。在 2001 年 9 月 11 日之前，计算机行业利用廉价的路由和防火墙设备就能够做得很好，这样既保护了网络资源，又为网络管理员提供了负载平衡和故障保护功能。另外，数字认证、公钥加密、公钥基础架构(PKI)和应用程序安全技术逐渐提高并被采用。恐怖主义的袭击、炭疽热的惊吓、世界范围内阴谋的怀疑和新的全球激进主义者使得软件开发人员、QA 技术员、IT 管理员和用户将安全提升到一个全新的优先级别。在安全环境中，测试 Web 使能应用程序需要理解安全基础架构对于测试的效果，以及需要理解测试安全的 Web 使能应用程序所需要的技术。

10.1　基础知识

本章不是关于 Web 使能应用程序中安全的全面论述。因为现今的环境里安全技术变化迅速。因此，本章主要致力于为开发人员、QA 技术员和 IT 管理员提供在安全环境中进行测试的一个入门，其方式是介绍测试安全环境中的 Web 使能应用程序的实用技巧和技术。

以我的经验，安全基础架构很少影响系统的可扩展性、性能和功能。测试很少针对防火墙和路由设施，因为测试常常先发现其他的可扩展性和性能瓶颈。在进行功能测试时，大多数安全基础架构技术一旦首次运行成功，将反复运行。一般地，网络层次的安全设备(防火墙和路由器)能够满足服务于系统的所有组件的最大带宽和容量。

遗憾的是，在安全基础架构保护我们远离黑客和恐怖主义的同时，它也有自己的缺点。测试者的座右铭是“在验证之后信任”，而安全专家的座右铭是“不信任所有人”。这渗透到安全软件和设备中，在安全世界里友好的图形界面是不受欢迎的。

以我的经验，安全基础架构的配置和安装是测试中最耗时的部分。由于设备和软件在安全基础架构中很少发现异常，所以这提供了多种复杂、精致和非标准的配置协议。因此足以令人感到灰心和枯燥，在我早期的职业生涯里遇到过这种情况。

本章介绍了一些测试问题，这些问题是网络路由、虚拟专用网和网络分段为了获得有意义的测试结果，所必须面对的。然后，研究了使用 SSL 和 HTTP 基本认证协议的传输层安全。本书提供了新兴的安全标准，包括 SAML、WS-Security 和 XML 签名。如果想阅读其他有关 TestMaker 如何支持高层安全标准的资料，请访问 PushToTest Web 站点。

10.1.1　路由安全

为了理解网络层安全，我介绍了对于 TCP/IP 互连和路由随着时间的变化时，一个网络管理员是如何进行选择的。图 10-1 演示了最基本的 Web 环境，其中 Web 客户端通过公共 TCP/IP 网络直接连接到一台运行 Web 服务的计算机上。在这个配置中，互联网上的任何设备都可以

对这个主机开放一个 socket。开始时，Internet 将这个配置当作主要配置，但并没有持续很久，因为不久之后证明这种配置存在问题。

图 10-1　开放 Web 站点的基本网络互连直接将客户端连接到服务器上

20 世纪 90 年代中期，Internet 网络掌握在少数主要提供商手中。当一个主要提供商犯了路由配置错误时，就会成为头号新闻。破坏路由表就可能将所有的网络信息流通量导向单个开放的服务器，这使得该服务器上的 Web 使能应用程序无法被访问。很快，网络操作者和 Internet 服务提供商开始使用可以灵活控制访问策略的路由器。例如，一个常用的路由策略是阻止来自同一个源的多个 ping 请求，以防止拒绝服务攻击。今天网络管理员实现了上百个路由策略，作为安全网路信息流通量计划的一部分。当开始测试一个 Web 使能应用程序系统时，经验表明，与网络管理员进行勾通是一个明智的选择，可以避免后面可能发生的许多问题。

另一个巨大的变化来自于早期的 Internet，包括使用网络地址转换(Network Address Translation——NAT)和动态主机控制协议(Dynamic Host Control Protocol——DHCP)。图 10-2 显示了一个网络环境，其中客户通过 NAT/DHCP 路由器连接到 Internet。DHCP 服务赋予客户机一个有效的 TCP/IP 地址。当客户端使用 TCP/IP 地址发出请求时，路由器将这个 IP 地址转换为公共可访问的 Internet 地址，并将请求作为一个代理发送给客户机。

图 10-2　路由器提供网络地址转换和防火墙问题，也提供安全的形式来保护客户机免受黑客攻击

在 Internet 的早期，NAT 里是很有争议的，因为它使网络管理更加困难。例如，如果一个网络管理员企图从服务器 ping 客户机，ping 的响应来自于 NAT 路由器而不是客户机。今天，NAT 被广泛使用。例如，2Wire 制造了一个流行的家用 DSL 调制解调器/路由器，它包含了 NAT 和 DHCP 服务器，并被主要的 ISP 销售。

防火墙、NAT 服务和路由设备都是面向网络层解决路由问题；但是它们直接影响了安全，因为它们使得基于 IP 地址的安全变得非常困难。因此，很多致力于实现 Web 使能应用程序安全的努力已经转移到应用层。

10.1.2　虚拟专用网

虚拟专用网(Virtual Private Network—— VPN)是一个网络(通常为一个公司的内部网络)利用另一个网络(通常为 Internet)的一种隧道传输方式，因此就好像是您直接连接到公司的网络上一样。为了保护穿越 VPN 的数据，大多数 VPN 使用数字认证和加密技术来保护隧道数据。

图 10-3 显示了使用 VPN 技术从客户端到服务器的通讯示例。

在开始的时候，VPN 通过运行客户机和服务器上的安全软件来创建。后来的 VPN 设备——小的类似路由器的设备——提供了 VPN 加密和验证功能，但是不满足客户端和服务器的

带宽。今天，VPN 软件大量出现，包括运行在客户端设备上的 VPN 软件、专门的 VPN 设备以及作为防火墙与路由器的一部分的 VPN。

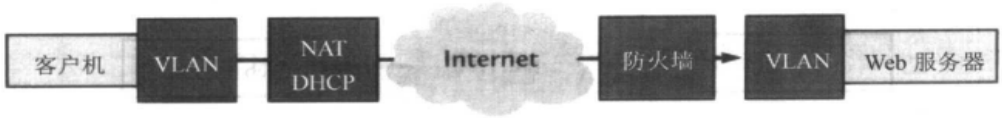

图 10-3　在客户端的软件和服务器设备解码数据，并通过 Web 基础架构通信形成 VPN

遗憾的是，每种类型的 VPN 系统通常都没有可供测试代理配置 VPN 所使用的编程接口。例如，我在测试一个 XML-RPC SSO(Single Sign-On：单点登录)设备时，就遇到了这种问题，该设备将一个智能测试代理连接到一个私人安全协作网络上的一台服务器上。在这个设置里，XML-RPC 在网络外面发出请求，并需要存储在协作数据库里的 VPN 授权密钥。在这个示例中，解决方案是使用一个批处理脚本来获取密钥代码的一个小缓存，该批处理脚本创建了一个数据文件。随着测试的进行，测试代理查询数据文件来获取所需的密钥。

由于人们常常认为应用服务器和数据库服务器是最大的性能和可扩展性瓶颈，所以 VPN 的可扩展性和性能常被忽略。使用智能测试代理有助于您了解内置于 VPN 设备、软件和实现中的性能开销有多大。经验向我们展示了测试一个使用或不使用 VPN 的系统的令人惊讶的结果。

10.1.3　网络段和子网

网络管理员对一种额外的安全级别的追求，已经将一个系统的多个部分放置到彼此不相连的独立网络中。图 10-4 演示了一个流行的使用一个 Web 服务器、应用服务器和数据库服务器来满足用户需求的服务器端体系结构。通过在每台机器中安装多网络接口卡(NICs)，每个服务器都被放置到物理上独立的网络中。即使有人黑掉了 Web 服务器，由于应用服务器位于一个不同的物理网络上，因此再黑掉应用服务器还需作另一番努力。

图 10-4　聚焦一个安全的 Web 环境的服务器端，Web 服务器、应用服务器和数据库服务器常常被放置在与外部局域网相分离的网络段中，因此无法直接访问它们

当然图 10-4 中引入的网络段对于测试来说，带来了新的问题。假设运行一个功能测试时，在一个请求的应用服务器阶段出错。因为在不同的网络上，智能测试代理没有直接连接到应用服务器上。一种可行的解决方法是把智能测试代理放到每个网络段中，作为服务监视器。在这样的系统中，智能测试代理作为应用服务器段的一员。另一种解决方案是在单台机器上运行测试代理，该机器包含多个 NIC——每个网络段都有一个网络接口——用于 Multihoming。

下面描述了一个网络的可扩展性设计，它使用一台负载平衡器和几个等同的主机来处理到达的请求，该结构称为 Flapjacks 体系结构。图 10-5 演示了安全环境中的 Flapjacks 体系结构，其中使用了 VPN 技术来溯流负载平衡器。

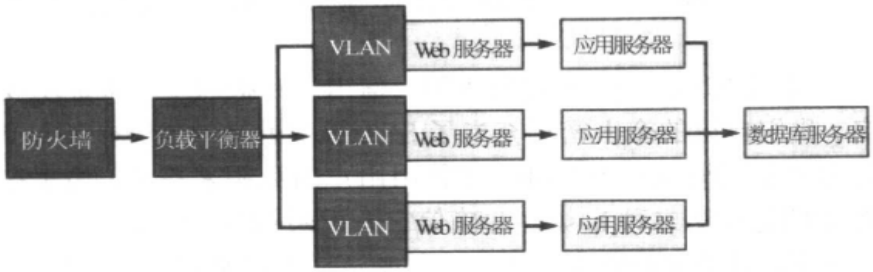

图 10-5　负载平衡器是一种动态路由器，在用户会话期间，它分配并管理用户机器对主机的仿射

几种负载平衡器也支持 VPN 和 SSL 技术，使得它们能够将 CPU 带宽之外的 VPN 请求卸载到专门的负载平衡设备上。在 Flapjacks 体系结构里 Multihome 是很重要的。如果没有 Multihoming，由智能测试代理生成的信息可能被负载平衡器路由到单个 Web 主机上。遗憾的是，这样不能提供有意义的测试结果。

负载平衡通常希望具有它自己的处理器。例如，我向一个客户提供了多种测试服务，使用负载平衡器将负载共享到一打基于 XML-RPC 系统的应用服务器中。它们的应用软件仅仅是适度可靠的。因此，网络管理员将在每天夜晚循环每个应用服务器。在循环一个服务器之前，网络管理员会等待该应用服务器上的所有当前会话结束。用户在 NAT 路由器之后发出请求。负载平衡器误认为来自 NAT 路由器之后的用户请求共享同样的会话。对于网络管理员来说，服务器从不停止会话服务。切换负载平衡器，以使用基于 cookie 的会话管理，该会话管理提供了精确的会话管理，即使是针对来自一个 NAT 路由器的信息流通量。

到目前为止，本章主要集中阐述了局域网或广域网中的网络级和用户级的安全。接着我们将介绍传输级安全。

10.1.4　传输安全

当业务匆忙地将基于 Web 的应用集成到它们的信息系统中时，它们需要 Web 基础架构具有传输层安全。SSL 协议被广泛地使用，它是保证应用和最终用户间信息安全交换的非官方标准。

SOAP 和 HTTP 规范的意图不在于描述安全的请求和响应机制。在 SOAP 的安全里，出现了几种安全技术和策略，包括 SLL 连接上的 HTTP、SLL 连接上的 SOAP、.NET 护照认证和 HTTP 基本认证。在 Web 服务里，安全需要符合 WS-Security 和 SAML。

1. SSL 连接上的 HTTP

SSL 是现今世界众多业务中信息安全技术的基础。在业务应用中使用的 SSL 提供了下面一些特征：

- 验证服务器是它所说的哪一个
- 保持应用和用户之间任何通信的私有性

另外，只需要一些努力，数字认证 SSL 可以用于证明传输的数据仍没被损害，还可以向服务器证明用户的身份。

SSL 使用数字密钥来完成这些目标。这些密钥通常基于大素数——如 RSA 和 Verisign。SLL 使用两种密钥：只能由您访问的私钥和可以由任何人访问的公钥。

私钥和公钥同时工作。例如，使用 RSA 密钥，使用私钥加密的消息只能被公钥译码。密钥越长，越难解密。

在现实世界里，我们通过证书——司机执照、信用卡和护照——来证明我们的身份，并具有惟一的手写签名。我们自己的个人密钥充当了 SSL 数字世界中的一个签名。为了对一个文档进行数字签名，利用该文档组成一个 hash，并使用用户的私钥来签名。为了在后面验证该文档的签名，您需要对文档运行第二种 hash，并确信它匹配最初的 hash。

在 20 世纪 90 年代，计算机行业支持 X.509 的建议，来定义数字认证的标准。X.509 实际上是一个国际电话通讯团体(ITU—International Telecommunication Union)建议，并没有被官方定义或赞成。结果导致不同的公司以不同的方式实现该标准。例如，Natscape 和 Microsoft 使用 X.509 认证来实现 Web 服务器和浏览器中的 SSL。但是由 Netscape 生成的 X.509 认证不能由 Microsoft 的产品读取，反之亦然。对于 ITU 的细节，请访问网站 http://www.itu.lint/home/index.html。本章的后面将描述，在 TestMaker 框架中如何使用 X.509 认证。

由谁来验证密钥呢？这取决于数字认证授权。用户生成私钥，在验证用户身份之后，认证授权使用其自己的私钥来签名用户的公钥。用户的公钥和认证授权签名的组合完成数字认证。

假设用户想要发送一封机密邮件。用户在接收者的公钥的帮助下加密信息，其中接收者的公钥存储在接收者认证中。接收者使用他或她的私钥来解密消息。因为只有接收者能够访问私钥，所以只有他或她能够解密信息。

在 Web 服务环境中，具有相同的 SSL 系统。当使用浏览器在线购买产品时，公钥数字签名技术提供了 HTTP/HTML 环境中的安全和授权服务。图 10-6 描述了客户机和服务器的 SSL 交互。

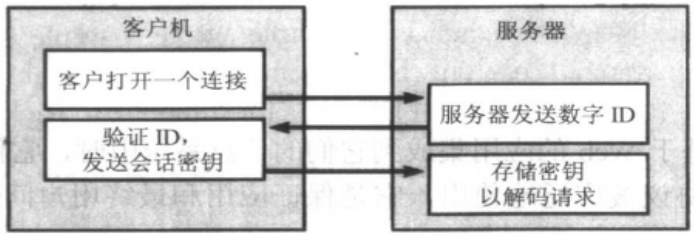

图 10-6 SSL 连接之上的 HTTP 请求的过程为：客户机和服务器交换数字认证来互相授权，然后他们使用相同的认证来加密数据

考虑在一个商业应用中的 SSL 示例，假设用户需要通过 Internnet 连接来安全地订购一个产品。浏览器打开 Web 主机的一个连接，并请求服务器的数字认证。服务器响应包含其公钥的服务器认证的公钥副本。浏览器验证该密钥，并发送一个仅当前会话期间有效的会话密钥。服务器存储该会话密钥，并使用这个密钥来解码后续的请求。

TestMaker 在它的协议处理程序对象中支持 SSL 授权和加密，这些协议包括 HTTP、HTTPS 和 SOAP。下面是一个代理，演示了如何使用 HTTPS 协议向 Web 主机发出请求，并使用 SSL 加密：

```
# HTTPS_Connect.a
# Author:  fcohen@pushtotest.com

print "Agent running: HTTPS_Connect.a"
print "Description:"
```

```
print " Connect to a Web host using an SSL connection."
print

from com.pushtotest.tool.protocolhandler import \
HTTPProtocol
from com.pushtotest.tool.response import Response

# Create an HTTP object to communicate with the host
# over an SSL encrypted connection
protocol = HTTPProtocol()
protocol.setScheme( "https" )

# Tell it where the host and document are located
protocol.setHost("citibusinessonline.da-us.citibank.com")
protocol.setPath("basprod/citiiwt/BAbuscde.html")

# Get the document
response = protocol.connect()

print "Received this response from the secure host:"
print response.toString()

print "Timing values for this request:"
print "Total:  "+ Long(response.getTotalTime()).toString()
print "Data :  "+ Long(response.getDataTime()).toString()
print "Setup:  "+ Long(response.getSetupTime()).toString()

print "Agent complete."
```

HTTPS_Connet 代理连接到 Citibank 的在线门户登录(sign-in)屏幕。该代理向登录页发出一个简单的 HTTP GET 请求，然后显示、响应，最后退出。

```
from com.pushtotest.tool.protocolhandler \
import HTTPProtocol
```

import 语句通知 TestMaker，代理将使用基本的 HTTPProtocal 对象。有关 HTTPProtocal 对象及其许多功能的细节，请参考第 3 章。

```
protocol = HTTPProtocol()
protocol.setScheme( "https" )
```

httphandler 变量现在指向一个新的 HTTPPrococal 对象。该对象处理与 HTTP/HTML Web 使能应用程序的所有通信。protocal.setScheme()方法通知 HTTPProtocal 对象，使用 HTTPS 协议保证对主机的请求的安全，其中 HTTPS 协议使用了 SSL 数字认证。TestMaker 使用了 Java 1.4 中的 Java 安全套接字扩展(JSSE—Java Secure Sockets Extension)库来提供 HTTPS 协议。有关 JSSE 的详细信息请访问 http://java.sun.com/products/jsse/index-14.html。

```
protocol.setHost("citibusinessonline.da-us.citibank.com")
protocol.setPath("basprod/citiiwt/BAbuscde.html")
```

setHost()和 setPath()方法告知 HTTPProtocal 在哪里能找到主机和文档。

```
response = protocol.connect()
```

connect()方法通知 HTTPProtocal 对象发出对主机的请求。HTTPProtocal 对象返回一个新的 response 对象，该对象包含来自主机的 HTTP 响应。

由于这个连接使用 SSL 进行安全连接，因此您可能注意到，它比非安全的 HTTP 请求要多花费一些时间。

```
print response.toString()
```

这个语句将 response 对象打印到 TestMaker 的输出窗口。

HTTPS_Connect 代理给出了一个通过 SSL 连接与 Web 主机进行交互的非常简单的示例

10.1.5 SSL 上的 SOAP

SOAP 规范特意避免描述请求和响应的安全机制。早期基于 SOAP 的 Web 使能应用程序使用 SSL 来加密 SOAP 请求和响应文档。本章的后面我们将看到为什么 SSL 并不很适合于保证基于 SOAP 的 Web 使能应用程序的安全。但是现在我们将介绍如何在一个 SOAP 环境里使用 SSL。

SOAP 实现运行在应用服务器和 Web 服务器技术之上。因此 SOAP 请求可能建立在 SSL 加密的连接之上。TestMaker 框架构建了一些智能测试代理，这些代理能够使用下面的测试代理脚本中的技术，与使用了 SSL 上的 SOAP 协议的 Web 使能应用程序的主机进行通信：

```
# Import tells TestMaker where to find Tool objects
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler, SOAPProtocol, SOAPBody, SOAPHeader

from com.pushtotest.tool.response import Response
from java.lang import Long

# Set the default values defining the location of the host
host = "examples.pushtotest.com"
port = 92
path = "axis/servlet/AxisServlet"
endpoint = host + ":" + str( port ) + "/" + path

# Create a SOAP object to communicate with the host
protocol = ProtocolHandler.getProtocol("soap")

# over an SSL encrypted connection
protocol.setScheme( "https" )

body = SOAPBody()
protocol.setBody(body)

# Set the endpoint values
protocol.setHost( host )
protocol.setPath( path )
protocol.setPort( port )

body.setTarget( "responder_rpc" )
body.setMethod( "Respond" )

body.addParameter( "wordcount", Long, 150, None )
body.addParameter( "delay", Long, 100, None )

print "Sending request to server..."
response = protocol.connect()

print "Here is the response:"
print response

print
print "Agent done."
```

除了 setScheme()方法，这个代理与前面几章中的代理相同。setScheme()方法告知被协议变量引用的 SOAPProtocol 对象，使用一个 SSL 连接与主机进行通信。

10.1.6 .NET 护照授权

Microsoft .NET 护照是一个简单的登录系统，它允许开发人员将他们的用户授权延期到.NET 护照服务器上(第 9 章介绍了用于创建 Web 使能应用程序的 Microsoft .NET 框架)。为了提供这个服务，Microsoft 使用密钥和配置文件的一个组合来确保用户的身份。

早期的.NET 应用使用 SSL 加密来保证应用和护照服务器之间的 SOAP 请求和响应的安全。本章的后面我们将看到为什么 SSL 不适合于保证基于.NET 的 Web 使能应用程序的安全。现在我们将给出护照提供的安全技术的简单概况。

从实现.NET 护照单个登录的 Web 站点的观点来看，.NET 护照管理器(.NET Passport Manager)是主要的服务接口。护照管理器通常是一个可以在 ASP 中使用的 Microsoft COM 服务器。但是，.NET 框架通过 PassportIdentity 类在托管代码中实现了大多数相同的功能。

在授权期间，.NET 护照管理器负责提供到.NET 护照服务器的一个链接，其中利用了查询字符串中的数据，.NET 护照服务器能够执行适当的验证，并将用户返回到建议的站点。.NET 护照管理器使用输入参数、注册设置和存储在.NET 护照服务器中的关于站点的数据的组合，以此来告知.NET 护照服务器需要根据站点的需求完成授权。当授权被建立时，用户浏览器被重定向到站点上，并设置 cookies 以包含用户的.NET 护照票和后续请求的简档信息。关于护照的详细情况，请访问 http://msdn.microsoft.com/webservices/和 http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/MSDN-FILES/027/001/644/msdncompositedoc.xml&frame=true。

10.1.7 HTTP 基本授权

HTTP 协议定义了一种基本的授权方法，该方法提供了非常简单的安全机制，以此来保护 HTTP 请求的结果。在 HTTP/HTML 环境中——如第 6 章描述那样——Web 主机在提交请求时，要求 HTTP 头包含用户 id 和密码，但这并不是强制性的。

如果 HTTP 头丢失，服务器将响应“401 未授权错误代码”。除了显示 401 未授权错误代码之外，大多数浏览器还显示一个警告，要求用户输入它们的 id 和密码。然后浏览器提交相同的请求加上额外的 HTTP 头，其中 HTTP 头中包括 id 和密码。

TestMaker 在它的 HTTPProtocolHandler 对象中支持 HTTP 基本授权。下面引用了 TestMaker 中实现的一个智能测试代理，其中 http.setUsernanme()方法和 http.setPassword()方法定义了在 HTTP 头中发送到主机的 id 和密码。

```
from com.pushtotest.tool.protocolhandler import ProtocolHan-
dler, Body
http = ProtocolHandler.getProtocol("http")
http.setUsername("MyName")
http.setPassword("MyPassword")
...
http.connect()
```

HTTP 基本授权应当仅被用于那些安全风险最低的地方。HTTP 基本授权期望在一个请求的 HTTP 头中发现用户 id 和密码。头信息以明文的形式在 Internet 中传送，因此有网络监视器的任何用户都能看到没有加密的 id 和密码信息。

10.2 SOAP 和安全

大多数第一代的 Web 使能应用程序部署在公司防护墙后面的内部集成项目中。但是现在一些公司正将 Web 使能应用程序部署为能够通过 Internet 向商业伙伴、供应商和顾客展示内部系统。在财政、政府和保健部门可以发现 Web 使能应用程序技术的早期采用者，这些地方被攻击的风险非常大，因为在本质上这些地方交换的数据通常是敏感的或价值高的数据。

SSL 在第一代 Web 使能应用程序中对于加密和授权基于 SOAP 的 Web 使能应用程序通信是有效的。这些早期的服务一般仅在两点间移动信息。SSL 在这些环境中做得很好。SOAP 客户包含服务器认证的副本，能够授权服务器并建立一个加密通道。

SSL 中的问题出现在 B2B 事务里。Web 使能应用程序的 SOAP 接口访问高级功能，该高级功能是 B2C 的 Web 使能应用程序不能够访问的。例如，在一个 B2C 应用的站点中不需要客户认证就能够完成低价商品的交易，如录像带或枕头套。在 B2B 环境中，犯错误的风险更大，SOAP 主机更关心授权客户。

当 Web 使能应用程序需要双向授权时，需要使用客户端的 SSL 认证。客户授权服务器，服务器也授权客户。这样管理梦魇随之到来：每个客户和每个服务器都需要一个数字认证。

使用客户端 SSL 的外联网被广泛地部署，以提供授权，但是客户端 SSL 需要的局部化用户管理器对于基于 SOAP 的 Web 使能应用程序提供的松散集成解决方案来说，是不受欢迎的。更糟糕的是，Web 使能应用程序允许组织互相通信而不必同意公用包或实行扩展集成。SOAP 和 WSDL 意味着能够被快速地部署，以支持松散耦合系统。这样，SSL 也要求客户端 SSL 认证分布到每个可能向 Web 使能应用程序发出请求的基于 SOAP 的 Web 使能应用程序。这是必须要解决的问题！

一个老矿山老板曾说过："我们不明白的事情，我们就真的不了解它。"他的意思是，当一个人开始使用新技术工作时，他很少知道怎样提出问题。这个名言也适用于基于 SOAP 的 Web 使能应用程序的安全。Web 使能应用程序在未来会遭受黑客、攻击和错误都是肯定的。作为开发人员、QA 技术员和 IT 管理员，我们今天的任务是在下面的领域中放眼未来：

- 向内查找我们系统中的弱点。当发现这些弱点时，使用监视器和测试来更正它们。
- 向外查找哪种技术可以在以后应用，以解决安全弱点。

例如，通过向内查找，我们可以看到 SSL 没有保护 Web 使能应用程序免受更加传统形式的攻击。世界上的 Web 服务器每天都遭受缓冲区溢出攻击。黑客通过发送比 Web 使能应用程序所需要的还长的参数来进行缓冲区溢出攻击。在路由前验证消息的内容就可以阻止这种攻击。

通过向外检查，一些新的消息级的安全技术可用于提高除 SSL 之处的基于 SOAP 的 Web 使能应用程序的安全。这些 XML 安全技术在 OSI 栈的应用层或消息层起作用。

- **XML 模式** 描述了 XML 文档允许的内容和结构。在安全情况下 XML 模式保证了 SOAP 文档不包含怀有恶意的数据。详情请访问网站 http://www.w3.org/XML/Schema。

- **XML 签名** 验证一个 XML 文档在客户端和服务器提供者间没发生变化。一个 XML 签名包括 X.509 认证，用于链接请求者与请求。一个 XML 签名在签名中使用 X.509 认证，以授权请求者。这包括验证认证的状态，使用 XKMS 可以获得认证的状态。详情请访问网站 http://www.w3.org/Signature/。
- **XML 密钥管理规范(XKMS)** 提供 PKI 服务的 XML 接口，包括密钥发布、验证和撤销。XKMS 支持安全检查服务，即一个 Web 使能应用程序能够授权另一个 X.509 认证。详情请访问网站：http://www.w3.org/2001/XKMS/。
- **SAML** 允许授权和认证信息被安全域的多个部分所共享。使用 SAML，基于 SOAP 的 Web 使能应用程序发出对授权和授权信息的 XML 请求。一个请求可以包含一个断言，该断言由信任的 SAML 授权签名，SAML 授权可以被服务提供者所使用以授权 Web 使能应用程序的使用。访问管理产品行业使用 SAML 来保证它们能共享授权信息。详情请访问网站 http://www.oasis-open.org/committees/security/。我的文章，*SAML Myths and Misunderstandings*，可以在网站上 http://docs.pushtotest.com 找到。
- **自由联盟** 是 Microsoft 强烈支持的标准体，它致力于为服务提供者组研究一个简单的签名标准，.这些服务提供者希望能够联盟他们的登录过程。详情请访问网站 http://www.projectliberty.org/。
- **Web 使能应用程序的互操作性(WS-I)** 是一个标准体，它主要用于定义第二代基于 Web 使能应用程序的平台，包括 Web 使能应用程序安全。详情请访问网站 http://www.ws-i.org/。

这些技术最杰出的地方是它们能够组合使用，以保护 Web 使能应用程序免受未授权的访问。

现在 Web 使能应用程序的第二次浪潮正在兴起，并比已往更需要消息级的安全。第 4 章关于 SOAP 的内容预测了一个未来，即 Web 使能应用程序组将为更大的应用提供功能。这种“联合应用体系结构”扩展了对安全技术和智能测试代理的需要，以验证安全环境中 Web 使能应用程序的可扩展性和性能。关于第二代 Web 使能应用程序安全技术的文章请访问网站 http://docs.pushtotest.com，其中包括一篇关于 *Myths and Misunderstandings Surrounding SAML* 的文章。

10.3 产生认证和密钥库

尽管公钥安全技术存在，并被很好地证明，但是我认识的大多数人甚至不想考虑关于认证的事情。这种行为导致浏览器发布者使用一些他们信任的根认证的随意集合。当您浏览一个没有匹配认证的 Web 站点时，浏览器会弹出一个对话框，询问您是否接受该未知的认证。

创建智能测试代理的 TestMaker 框架是一个使用 JSSE 库提供 SSL 服务的 Java 应用程序。安装 JSSE 时也同时安装了 Verisign 和 Thawte X.509 认证。当 JSSE 连接到没有相关认证匹配的服务器时，将抛出异常。

```
URL exception, javax.net.ssl.SSLException: untrusted server cert chain
```

本节将介绍 U.S. 海军关于 TestMaker 测试代理和安全主机的工作经历。然后，我们以一些指令来结束本章，这些指令是关于利用 Java 的密钥工具实用程序来使用您自己的数字认证的。

海军供给信息系统活动(Navy Supply Information Systems Activity—NAVSISA)经历

我们被邀请辅助测试属于我们基地海军活动的 Web 服务器。他们建议使用 TestMaker。我们的站点不安全，例如，URL 配置是“http”。我们必须驱动的页面是一个简单的由文本字段组成的页面。使用 TestMaker 很容易解决这个问题，我们很成功。

别人得知了我们的成功也向我们寻求帮助。但是这种新的请求是面向安全站点的，如 URL 模式是“https”。我们第一次试图生成如下这些信息：

```
"ElementURL: URL exception, javax.net.ssl.SSLException: untrusted
server cert chain"
```

我们的问题在于服务器使用的是国防部(Department of Defense—DoD)认证，该认证与 TestMaker 不兼容。

我们向 TestMaker 支持组寻求帮助，他们建议了一些信息资源，这些信息资源是关于如何将我们的 DoD 认证链放置到密钥库中，其中密钥库是由 Java 虚拟机(利用“密钥工具(keytool)”)使用。

我们下载了 CA 认证(dodroot.cac)文件，并使用 Java 虚拟机将它导入到“cacerts”文件，但是它无法运行。我们想到密钥工具不使用 DoD 所使用的格式(PKCS#12)。因此将“dodroot.cac”导入到我们的浏览器(Microsoft Internet Explorer 5.5)中，并将其导出到一个 X.509 格式的文件中，该格式文件是密钥工具所能接受的。

接着使用密钥工具的导入命令将 X.509 格式的“dodroot.cac”认证导入到“cacerts”密钥库中。

在这一点上，我们没有遇到“untrusted server certificates”错误信息——但出现了其他问题。

我们当前正在测试的 Web 站点需要由应用本身授权。提供给用户的第一个页面具有两个文本字段，用户需要在这两个文本字段中分别填写他们的用户名和密码。我们不知道将什么内容放置到 URL 中，该 URL 是在 TestMaker 脚本中构造的，用于完成授权。我们使用 HP 中(HP OpenView Internet Services)的一个工具发现我们所需要的关键字是“USERID”和“PASSWORD”。因此我们又成功了。

接下来的一个请求是测试 Web 站点，它也是安全的，但使用服务器来进行认证，而不是应用。例如，服务器被配置为通知浏览器弹出一个对话框，该对话框请求用户的信任状，而不是由应用来请求信任状。

这种授权方法直接使用了 HTTP 协议，取代了 URL 的某些部分。为了进行这种授权，我们需要某种方法来影响 TestMaker 构造 HTTP 头的过程。在文档里，我们发现了答案：HTTPProtocol 对象的 setUsername()和 setPassword()方法能够提供 HTTP 头的密钥/值，该 HTTP 头是我们正使用 TestMaker 构造的 HTTP 头。实际的 TestMaker 语句是：

```
from com.pushtotest.tool.protocolhandler import \
ProtocolHandler, Body
http = ProtocolHandler.getProtocol("http")
http.setHeader("Authorization", \
"Basic dHQwMDAwMDAwMDE6c3dtMTIzNDU=")
...
http.connect()
```

我们用来解决这个问题的方法的某一部分参考了 HTTP 1.1(RFC2068 11.1 节，基本认证模式)的 RFC，您可以通过 http://www.w2c.org 找到这些内容。我们也使用网络分析器/嗅探器来查看当与服务器通信时，客户机实际发送的内容。

单词“Basic”后面的“funny stuff”是用户 id：使用所谓的 base64-encoding 规范来加密的密码。实际上，我们没有这种能力，因此必须到网站上来寻找 base64 的编码程序。

如果我们能够为其他人提供服务，我们会尽全力帮助他们。

Ray Leiter

Raymond_R_Leiter@fmso.navy.mil

Java 密钥工具

用于构建智能测试代理的 TestMaker 框架是一个 Java 应用程序，该应用程序使用 JSSE 库提供 SSL 服务。安装 JSSE 时，也同时安装了 Verisign 和 Thawte X.509 认证。因此测试代理能够直接连接到使用了 Verisign 和 Thawte 认证的服务器。当一个测试代理与一个具有另一种认证类型的服务器建立安全连接时，将抛出异常：

```
URL exception. javax.net.ssl.SSLException: untrusted server cert chain
```

在本节中，我介绍如何利用内置的 Java 密钥工具实用程序来使用新的数字认证、生成密钥、密钥库和信任库。关于密钥工具的详细情况，请访问网站 http://java.sun.com/docs/books/tutorial/security1.2/index.html。

在下面的内容中，我们将利用几个命令来创建服务器和客户端的自签名认证，并将它们存储到密钥库 server.keystore 和 client.keystore 里。这些命令也将服务器认证导出到 server.cer 文件中，然后导入到客户密钥库中，命名为 client.keystore。客户端的认证被导出到文件 client.cer 中，并被导入到服务器的密钥库 server.keystore 中。

通过这些步骤，密钥库 client.keystore 也能被客户用作信任库，从而验证服务器提供的认证，反之亦然。

注意：在下面命令中使用字符“\”表示同一行的继续。“\”字符不是实际命令中的字符。

首先，利用下面的命令，生成服务器密钥库，并存储在文件 server.keystore 中：

```
java_home\bin\keytool -genkey -alias tomcat-sv \
-dname "CN=localhost, OU=X, O=Y, L=Z, S=XY, C=YZ" \
-keyalg RSA -keypass changeit -storepass changeit \
-keystore server.keystore
```

为了将密钥库中的认证导出到外部文件 server.cer 中，使用下面的密钥工具命令：

```
java_home\bin\keytool -export -alias tomcat-sv \
-storepass changeit -file server.cer \
-keystore server.keystore
```

为了在文件 client.keystore 中生成客户密钥库，使用下面的密钥工具命令：

```
java_home\bin\keytool -genkey -alias tomcat-cl \
-dname "CN=Client, OU=X, O=Y, L=Z, S=XY, C=YZ" \
-keyalg RSA -keypass changeit -storepass changeit \
-keystore client.keystore
```

为了将密钥库中的认证导出到外部文件 client.cer 中，使用下面的密钥工具命令：

```
java_home\bin\keytool -export -alias tomcat-cl \
-storepass changeit -file client.cer \
-keystore client.keystore
```

为了将客户的认证导入到服务器的密钥库中，使用下面的密钥工具命令：

```
java_home\bin\keytool -import -v -trustcacerts \
-alias tomcat -file server.cer -keystore client.keystore \
-key-pass changeit -storepass changeit
```

最后，为了将服务器认证导入到客户密钥库，使用这个密钥工具命令：

```
java_home\bin\keytool -import -v -trustcacerts \
-alias tomcat -file client.cer -keystore server.keystore \
-keypass changeit -storepass changeit
```

这些步骤完成了一些密钥工具功能。使用密钥工具命令时要牢记以下几点：

- 服务器认证中指定的服务器名是本机。如果您想使用网络上的符号名或 IP 地址来访问服务器，您必须指出产生密钥时的字符串。对于同一个物理机器，您也能够具有它的多个认证。
- 利用这个脚本运行这个示例是足够的，实际产品部署必须使用一个适当的名称值，从而获取由一个授权 CA 签订的认证。
- 我强烈建议使用产品等级工具来管理客户认证，尤其是在服务器想根据提供的认证对客户授权时。

本章没有介绍公共密钥基础架构或 SSL 的基本知识。我们推荐 Scott Oaks 的 *Java Security* 一书，由 O'Reilly&Associates 出版。详情请访问网站 http://security.oreilly.com/。

10.4 本章小结

本章提供了在安全环境里测试 Web 使能应用程序的技巧和技术。SSL 技术的应用使得商业和组织能够保证客户端和服务器间的通信安全。虽然商业上已经采用了 PKI 作为安全 Web 使能应用程序的非官方标准，但本章还是指出：需要新的 XML 技术来管理 SSL 不提供的认证、访问、和授权。

在第 11 章中，我们将了解在编写完一个智能测试代理，并测试一个系统后，还需要做些什么工作，主要包括解决关于所需要的设备、网络带宽和资源需求等问题，以使得测试产生出有意义的结果。

第11章 从构造到测试Web应用程序框架

在前面的几章中介绍了智能测试代理与 Web 使能应用程序进行通信和命令 Web 使能应用程序所需要的一些技术。本章继续介绍后续内容。本章中的技术和示例将演示如何配置和运行一个并发的和可扩展测试。接着，在第 12 章中将介绍如何分析结果。

11.1 交易办公桌和智能测试代理

电视、电影和报纸上经常会出现股票和债券交易商在计算机和电话系统之前工作的场景。在一个大的面板中显示了关于市场的交易信息，交易商使用键盘来购买或卖出股票，电话中传出紧张交易商大声喊出的指令。当一切工作正常的时候，每个人都会赚到钱。当系统减速或停止时，事业将遭受危急。因此急需智能测试代理！

智能测试代理在检测可扩展性和性能问题，以及监视服务在大体积、高容量且接近于实时系统情况下的运行情况方面，是一种理想的工具。当前的系统正逐渐变得复杂化。例如，在线股票经纪人通常需要能够将买卖指令与市场状况相结合的服务。在能源公司中，出售股份的命令将在油价降低到某个特定值时被触发。为完成这样的功能，经纪人系统必须能够自动从一个系统中收集油价，并指示一个买/卖系统出售股份。那么如何测试这个系统呢？

内嵌在基于现代可互操作的 Web 服务系统的复杂性需要新的测试方法和技术。例如，贸易系统将使用一个测试代理来模拟触发一个 Web 服务动作的复杂市场状况。本章介绍了智能测试代理如何使用多协议和数据源来产生有意义的测试结果。

11.2 可扩展的测试目标

依赖于复杂的互操作系统的当今商业需要一些测试方法学和技术，来回答基本的操作问题。在股票交易中，系统操作员的目标可以利用系统事务的正常运行时间和速度来衡量。虽然在股票交易系统中很多方面需要测试，但在这个示例中，最重要的测试需要回答下面的几个问题：

表 11-1 建立测试的方法和度量

问 题	方 法	度 量
我有正确数目的服务器吗	系统的每个部分都会增加处理事务的开销。在一个贸易系统中，防火墙、路由器、负载平衡器、Web 服务器、应用服务器、数据库服务器和存储系统都增加了处理事务的开销。一个系统的测试揭示了处理事务的最大瓶颈。针对预期数目的用户，解决系统瓶颈问题回答了所需要购买的设备数目问题	度量从客户机处每秒往返的事务数目。然后度量每个主要系统部件处理事务的开销

(续表)

问　题	方　法	度　量
系统如何处理用户负载增加的情况	系统用户具有内置的容量，可以容忍某些慢系统性能。但是，这带来了另一个问题，即慢的系统性能对商业目标起反作用。用户的生产率应当由所考察的用户来度量，以便确定系统性能的提高度，以及通过获知每个用户的管理生产率目标来度量	度量在系统上完成任务需要花费的时间；另外度量完成每个较大任务所需要的一些子任务所花费的时间
足够的活动将最终削弱系统吗	复杂的互操作系统通常会耗尽系统的资源。内存泄漏、未释放的文件句柄，太少的硬盘空间和随机的硬件问题都会影响系统的性能。测试期间在一段时间内模拟大量并发资源的使用情况能够发现系统的资源问题	在 1~2 天的长期测试里度量系统的资源(CPU 使用的时间、内存分配和硬盘的使用情况)

本章中提供的测试代理主要为一个假想的股票交易公司回答这些问题。我们通过查询现有的系统基础架构和确定用户原型来构造这个测试。

11.3　系统基础架构

对于假想的股票贸易公司，我们将优秀的系统性能定义为具有如下特征：

- 服务在一个用户和 100 个并发用户下工作是一样的。假定用户是混合群体。包括交易商、管理员、分析师和网络管理员。
- 所有的用户都具备丰富的经验，尽管某些用户可能仅使用一个功能，而其他用户可能使用所有的功能。
- Web 服务在用户任何时候上线时都可用。

系统组件的选择在很大程度上决定了服务所能够达到的目标。前几章中介绍了创建和部署 Web 使能应用程序的框架，即 Flapjacks 体系结构。如图 11-1 所示，第 2 章介绍的在股票贸易公司里的 Flapjacks 体系结构部署了很多小服务器，这些小服务器通过负载平衡器来访问，它们为强大的数据库服务器提供了前端。

股票贸易公司系统把业务逻辑的内核放到应用服务器层。用户的请求通过负载平衡器和 Web 服务器进行服务。贸易应用软件从数据库服务器访问用户账号和交易历史。系统通过对外部 MarketFacts 服务的一个 SOAP 连接来访问当前的市场状况。

股票贸易公司使用 Flapjacks 体系结构来减少停机时间，并随着时间的推移，在改变 Web 和应用服务器数目方面提供了灵活性。另外，Flapjacks 体系结构是可测试的。

图 11-2 显示了添加到系统基础架构中的智能测试代理可以记录、度量和在某些情况下驱动系统组件。在股票贸易公司示例中，测试的业务目标是要求代理监视主要组件，模拟用户请求，并检查远程资源的性能和可用性，例如 MarketFacts 服务。

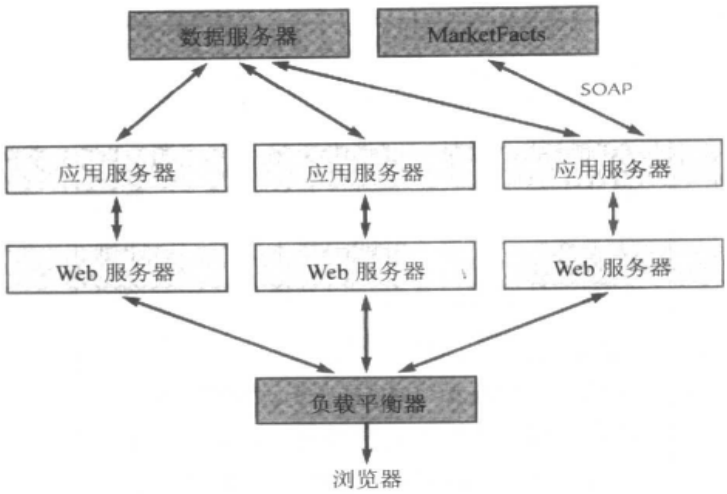

图 11-1 Flapjacks 系统体系结构实现了假想的股票交易公司。Web 服务层作为中间层来处理简单的页面请求。应用服务器处理动态配置请求信息所需的复杂过程

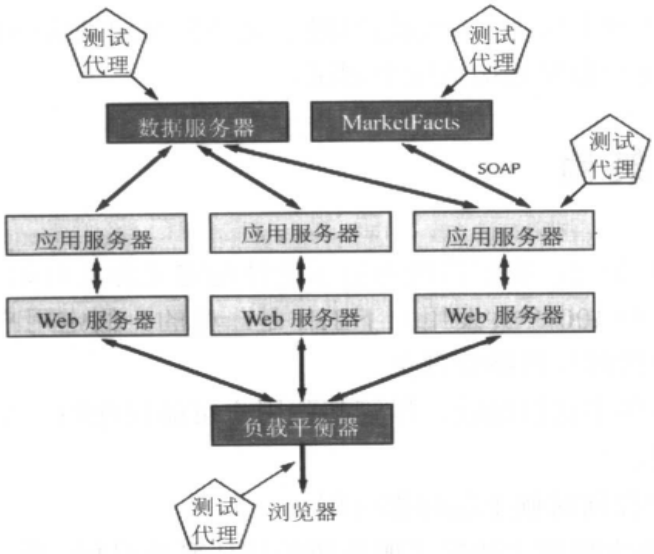

图 11-2 添加到 Flapjacks 基础架构中的智能测试代理用于确定系统瓶颈、组件级性能和可扩展性

每个测试代理监视不同的组件，如下所示：

- 测试代理监视数据服务器，记录数据库服务器应用的 CPU 和硬盘的利用率。
- 测试代理监视应用服务器，记录处理事务所花费的时间。
- 测试代理检查 MarketFacts 服务，确定网络延迟开销、Web 服务的可用性和事务的持续时间。
- 最后，最底层的测试代理模拟用户原型集合的活动。

接着，我们学习如何通过识别一些用户原型来识别系统上典型的用户活动。

11.4 用户原型

股票交易公司是应用用户原型的一个很好的示例。用户原型是一种强大的技术，用于模拟多面系统中的用户。多面系统需要多个步骤完成一个目标。关于用户原型的详情，请参阅第 1 章。

用户原型是一种很好的技术，可以消除许多开发人员在考虑如何使用他们的软件时所经历的恐惧感。很多开发人员考虑到测试必须覆盖有代表性的用户团体。其他的开发人员相信高质量的软件需要测试，以便满足由产品管理员、项目市场人员和组领导所定义的 Web 使能应用程序的最初设计目标。那些不能适应特殊环境的测试方法通常都满足不了目标需要。

通常的测试一般要进行大量的假设，包括用户集合将如何使用 Web 使能应用程序和他们完成通常目标所采用的步骤。系统并不是这样简单地被使用的。事实上，每个用户都有其自己使用系统的目标和方法以达到其目标。

普通测试的矫正方法是开发和应用用户原型。原型是基于使用系统的真正人员的典型用户。构建一个用户原型的最好方式是以单用户开始。只选择一个用户，观察他如何使用 Web 使能应用程序，了解其目标，观察他为了达到目标所采取的步骤。您越理解个体用户的需要，您就越能创造有价值的测试。

很多开发人员的心中都具有原型用户方法。他们命名用户原型并描述其背景和习惯。他们给定原型的深度，因此开发组的其余人员能更好地理解该原型。因此，开发组能更好地理解测试是因为他们考虑到了用户原型。他们也能在描述 Web 使能应用程序中的问题时使用原型。对于股票贸易公司，考虑如下这些用户原型：

- Mira——股票交易商，在办公桌前工作，通过电话定购，或通过系统终端购买或出售股票。Mira 是一个有强迫症的人：在学习了如何打高尔夫以后，她参加了一个 3 周的免费度假，在六个州中跟随 Tiger Woods 学习高尔夫课程。Mira 结过婚，但现在离婚了，没有孩子。她每天都喝 Starbucks 咖啡。她不能忍受的时候就发怒和挖苦。
- Simon——股票交易商实习生。毕业于名牌大学的商业学院。这是他走出校门后的第一份工作。Simon 在划船组，取得过相当好的成绩，他有一个新女朋友。Simon 在办公室有规律的工作，并研究一个由州投资的股票的分析认证，这要花费他两年的时间。
- Doree——股票操作管理员。她已经在股票市场领域工作 10 多年了。她比其他的股票交易商工作得更出色，但仍然抽出时间训练新的股票交易商。她两年前结婚了，她的丈夫是一个有竞争力的公司的销售经理。夏季期间，她一般躲在室内，在冬季，她喜欢滑雪、驾驶雪橇和穿着雪靴步行。
- Vicki——系统管理员。CEO 儿子的朋友。两年前在公司开始工作，主要负责维护 CEO 的台式计算机系统。从那时开始，Vicki 担任了逐渐复杂的系统管理工作。她声音温和，有时有点羞怯，这是她的父母遗传给她的特点。她的父母是佛教徒，反战的和平主义者。Vicki 最害怕的事情是在办公室中面对 Doree 和 Mira 的暴怒。

这 4 个用户原型没有覆盖所有的系统用户。但是他们能鉴定这 4 个用户原型目标。随着时间的流逝，也可以定义其他的用户原型和更多的测试，用于模拟更多的用户目标和更多的测试代理。

使用通用标记语言(UML)技术的开发人员会发现用户原型与 UML 的 Actor 有些类似。用户原型是 UML 的 Actor 思想的深入。在这种情况下，您能够将 Actor“角色扮演”为一个真实的人。

用户原型方便了智能测试代理的开发。例如，在 Mira 原型上建模一个测试代理主要集中在处理事务的速度上。Mira 需要很快地获取股票信息，了解新的行情，作出定购决定。为了完成这些目标，测试代理需要登录到系统，使用搜索功能发现一支股票，然后进行定购。这个代理在几个步骤之间没有延迟，因为它竭尽全力尽可能快地完成多个事务。该测试代理完成其目标需要几个步骤，并且当系统无法实现 Mira 的速度目标时，测试代理将给出提示。

为了完成 Vicki 的目标所建模的测试代理主要用于访问和监测系统。Vicki 需要知道系统在何时没有正常运行。她需要在 Mira 和 Doree 之前发现系统的运行速度缓慢。该测试代理需要查询系统的性能日志，以了解哪一个事务所花费的时间最长。

理解用户原型是创建智能测试代理的关键。例如 Vicki 的测试代理可能比 Simon 的测试代理更持久。如果一个测试代理企图执行一个使该测试代理出现故障的交易，Vicki 将重试。原型行为使得测试代理具有智能行为。

使用原型来描述一个用户，比制定一个 Web 使能应用程序用户的本质的广泛概括更有效、更精确。原型使得开发模拟每个用户目标的测试代理更加简单。

接着，我们研究进行股票交易示例测试的需求。然后，将构建智能测试代理，并运行这个测试。

11.5 理解测试需求

在软件开发的某一个时段中，测试活动都是由一个人组织和进行的。其后我们发现测试复杂的可互操作的信息系统，尤其是 Web 使能应用程序，需要多个组共同努力和协作。测试不再是一个人的工作了。例如，在股票交易公司示例里，设想 1000 个并发运行的 Simon 测试代理所产生的效果。Simon 在大多数股票交易前进行研究。这些研究导致了对局部数据中心和外部 MarketFacts 服务的网络传输和负载。如果没有足够的计划和协调，这样的测试可能导致 Vicki——股票交易公司网络的管理员——怀疑网络被黑客攻击。另外，MarketFacts 销售管理员可能认为股票交易公司的下个月账单会很大，因为对市场数据的请求数目太多。而事实上这些数据是为了响应一个测试代理的请求。

在一个测试中，协调多个人的工作需要理解他们的需求。我们可以把这些需求描述为一个需求列表。在股票交易公司示例里，需求如表 11-2 所示。

表 12-2 测试股票交易公司系统的需求

需　求	为什么重要	解 决 方 案
测试必须是可配置的	开发人员、QA 分析师和 IT 管理员都想运行他们自己的测试。测试代理必须是可配置的，不需要最初的测试代理创建者的干涉	测试代理使用配置文件来定义测试参数

(续表)

需　求	为什么重要	解 决 方 案
测试必须是可重用的	开发人员、QA 分析师和 IT 管理员能够创建测试代理。一旦创建测试代理后，它就是公司的资产，公司里的任何人都能使用它	测试代理都是在同一个框架里开发的，该框架包括易用的脚本语言、协议处理程序对象库和启动装置。代理被封装为可共享的软件应用，这些软件应用被放到一个源代码版本控制系统中，以便发布使用
测试必须是可维护的	即使测试代理仅包含简单的几行代码，它最终也需要变化、更新和更改，以维护它的可用性	代理被封装为可共享的软件应用，这些软件应用被放到一个源代码版本控制系统中，以便发布使用
测试必须是模块化的	测试代理实现了一些功能，这些功能在一个较大范围的软件自动化开发中十分有用。例如，一个自动化的构建-部署(build-and-deploy)系统将从一个测试代理插件中受益，利用该插件完成构建-部署-测试(build-deploy-test)功能	可以在外部应用中调用测试代理。另外，可以在外部共享测试代理

需求仅要求测试代理是可配置的、可重用的、可维护的和模块化的。实现这些的难点是什么？在 11.6 节中，我们将在构造股票交易公司的测试时，讲解这些难点。

11.6　构造测试

本章的测试代理回答了关于一个假定股票交易公司的可靠性、可扩展性和功能的基本问题。在前面我们已经学习了如何定义测试的目标，在理解用户目标之后如何使用用户原型来建模测试代理的行为，以及如何定义测试环境的需求。现在我们将构造这个测试。

首先，让我们理解测试代理环境的流程。图 11-3 显示了测试的主要部件和它们之间的关系。我们在三个连续的部分中构造这个测试：安装、运行和分析。实际上，分成这几个部分后，使得随后的管理和维护更加容易。每一部分完成一个基本的功能，从而最终完成整个测试。

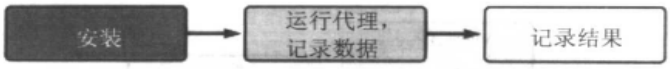

图 11-3　整个测试分成 3 部分。安装部分描述了测试环境、并发运行的测试代理数目和每个基于用户原型的代理的行为。第二部分实际运行代理，并将结果记录到日志文件中。第三部分分析记录的结果并显示结果

虽然图 11-3 给出了从测试的一个部分到下一个部分的流程，但理解每一部分是如何设计的，对于理解测试组件来说将至关重要。股票交易公司测试由以下几个步骤实现：

- 一个主代理调用一些代理模型，以完成驱动系统的实际工作。

- 一些代理模块实现用户原型描述中定义的行为。例如，doree.a 是一个 TestMaker 脚本，它实现了 Doree 用户原型的行为。测试结果存储在特定的数据文件中。
- 一个记录(tally)模块分析特定的结果数据文件内容，并显示结果。

图 11-4 演示了主组件。在股票交易公司系统测试中，具有很少的安装需求，因此这个测试在主组件中实现了安装功能。这是因为这个测试的测试代理期望 Doree、Mira、Simon 和 Vicki 的用户账号已经被建立，并且期望系统能处理交易定购和信息请求，而不必在测试前预先配置。在测试代理运行之前，人们希望系统的状态已经被配置好了。

图 11-4 master 代理封装了整个测试。结果被记录到一个文件中，由记录代理总结和显示

在其他的系统测试里，配置一个测试的系统要求该系统状态是可配置的。这包括几个操作，如建立新的用户账户、为每个新建立的账户设置使用参数、输入测试代理将查询的市场信息。如果系统依赖于某种类型的预配置，则对于记录和配置安装来说，使用测试代理就比较理想。这包括创建尽可能多的初始状态。

除了安装，主组件在多个并发运行的线程中实例化用户原型的行为。每个线程封装用户原型的行为，并发出驱动股票交易系统所需的命令。测试的配置文件通知主组件要创建多少个线程和以及在每个线程中要使用什么类型的用户原型行为。实际的用户原型行为在几个文件中实现，这些文件由主组件调用。例如，mira.a 文件包含 Mira 用户原型行为的实现。Simon、Doree 和 Vicki 的也如此。

查看一个用户原型的实现可以发现其基本的行为功能。图 11-5 显示了 Simon 用户原型。本章前面描述的 Simon 用户原型表明他是一个股票交易实习生，负责进行股票交易。Simon 也对市场趋势感兴趣，因此他也请求 MarketFacts 服务进行股票交易定购。在 Simon 调察一种趋势时，他提醒他的同伴 Mira 和 Doree。Simon 以邮件消息的形式发送他获取的信息。

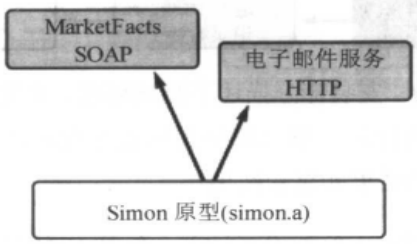

图 11-5 测试代理对 Simon 的建模遵循市场趋势(其方式是利用 MarketFacts 服务的几个 SOAP 接口)，并且通过基于 Web 的邮件服务来发送邮件进行提醒

在本章的后面我们将研究 Simon 代理是如何被编码的。现在，让我们看一下如何定义测试的行为，主要考虑用户原型完成他们的目标所需要的步骤。

在 Mira 的示例里，用户原型描述的用户行为是积极的和快速的。为了捕获 Mira 的行为，测试代理需要执行下面的动作：

- 它要同时处理几个股票交易。
- 它逐个地请求股票价格。如果一个请求超过 2 秒，代理就会取消该请求并进行下一个请求。
- 代理发出一连串的请求，然后暂停一会儿，这就像是 Mira 代理正在接电话，在进行动作之前，需要稍等片刻。
- 它检测 Simon 测试代理发出的邮件消息。如果存在邮件消息，则阅读这些邮件信息。

Simon 用户原型描述的行为甚至比 Mira 更快。Simon 发出很多对 MarketFacts 服务的请求来了解市场情况。下面是 Simon 测试代理所采取的行动：

- 它请求 marketFacts 的 2~5 个报告。在分析一会儿报告的内容后，重复这个过程。
- 它偶尔进行股票交易。
- 它将研究的结果以邮件消息的形式发送给 Mira 和 Doree。

实现了 Doree 行为的测试代理和 Mira 的一样活跃。但是 Doree 要执行轮询任务，她观察并确保交易是否被有效地处理，如下所示：

- Doree 测试代理一遍遍地检查新消息。从代理开始运行直到测试完成，一直会重复进行下去。
- 每隔一小时，代理会向交易商发送新消息，指示市场趋势和公司的股票情况。
- 每隔 10 分钟代理请求一个管理报告，来显示每个经纪人的销售状况。

主组件使用配置文件来获知需要创建的线程总数和需要使用的用户原型的相关数目。例如，假设配置文件调用 100 个并发运行的线程。配置文件定义了 35%的线程使用 Mira 用户原型，20%的线程使用 Simon 用户原型，30%的线程使用 Doree 用户原型和 15%的线程使用 Vicki 用户原型。图 11-6 显示了用户原型的划分状况。

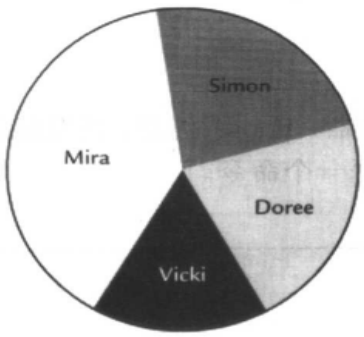

图 11-6　master 组件使用配置文件来确定用户原型行为的百分比，它将在全部的线程中创建该百分比。代理的总数可能增加，但是用户原型的百分比保持恒定

现在我们理解了用户原型的动作和测试组件的整个构造，下面将编写代码来实现这个测试。

11.7 将用户原型实现为代码模块

下面，我们学习如何规划股票交易公司的测试代理，使用 TestMaker 测试代理来实现测试中的用户行为。在第 4 章中已介绍过 TestMaker。我们首先介绍如何将 Doree 用户原型实现为一个智能测试代理。

在查看下面的代码时，请了解两点。首先，关于股票交易测试组件的完整代码可以通过网站 http://thebook.pushtotest.com 下载。其次，为这个测试代理实现的功能仅模拟了实际的调用。

Doree 代理反复检查新的消息。每次 Doree 代理都向其余的交易商发送一个通知消息。下面是实现这两个方法的脚本函数。第一个函数定义的方法用于检查公司的邮件服务器，查找新消息。下面是这个方法的代码，其后是关于该方法的详细解释。本书的所有源代码都可以通过 http://www.pushtotest.com/docs 下载。

```
# ------------------------------------------
# Read email messages

def read_emails():
    protocol2 = ProtocolHandler.getProtocol("mail")

    protocol2.setHost("mail.pushtotest.com")
    protocol2.setUsername("buddy")
    protocol2.setPassword(  "email"  )

    response = protocol2.connect()
    response.setPermission( Folder.READ_WRITE )
    response.setFolder(  "INBOX"  )

    messages = response.getMessages()

    for i in messages:
        print i.getFrom() [0],  i.getSubject()
        i.setFlag( Flags.Flag. DELETED,  1 )

    response.close( 1 )
```

read_emails 方法检查邮件服务器，查找新消息。当发现消息时，该方法下载并删除该消息。接着，介绍 read_emails 方法里的每一个命令。

```
#-----------------------------------------------------
# Read email messages

def read_emails():
    protocol2 = ProtocolHandler.getProtocol("mail")
```

首先，定义一个新方法 read_emails，在后面相同的代理脚本里将调用它。接着，脚本创建了 Mail-Protocol 对象的一个新实例，我们可以在一个协议变量中引用该实例。可以通过查看 testmaker_home/agents/Mail_agent.a 来获取关于 Mail 协议处理程序的信息。

```
protocol2.setHost("mail.pushtotest.com")
protocol2.setUsername("buddy")
protocol2.setPassword(  "email"  )

response = protocol2.connect()
```

这些命令告知 MailProtocol 对象在哪里找到所要使用的邮件主机和账号。PushToTest 在 mail.pushtotest.com 上创建了伙伴邮箱账号，这使得您能够运行测试代理。请不要滥用这个邮箱账号。

```
response.setPermission( Folder.READ_WRITE )
```

setPermission 方法指出我们的测试代理想使用的邮件主机。默认情况下，Folder.READ_ONLY 通知邮件主机我们将阅读邮件消息，但是不会从主机中删除它们。Folder.READ_WRITE 通知邮件主机，期望获得消息删除命令。

```
response.setFolder(  "INBOX" )

messages = response.getMessages()
```

POP3 协议默认情况下使用一个名为 INBOX 的文件夹。IMAP 协议支持多个文件夹。

然后代码循环处理每个消息。getForm、getSubject 和 writeTo 方法返回 from、subject 和 message 体。

```
for i in messages:
    print i.getFrom() [0],  i.getSubject()
```

也可以在 writeTo 方法中使用 message 体，该方法将一个 outputStream 对象作为其输入参数。setFlag 方法支持代理脚本标记邮件消息，以便从邮件主机中删除这些邮件消息。在这个循环中设置标记，但是直到 close()方法执行时才实际起作用。如果 setPermission 方法没有被设置成 Folder.READ_WRITE，则该标记将不起作用。

```
i.setFlag( Flags.Flag. DELETED,  1 )
```

在我们循环处理完所有的消息后，脚本将断开与主机的连接。close 方法使用了一个布尔参数。当参数被设置为 True 或 1 时，close 命令指示邮件主机删除任何被标记的消息。False 或 0 参数通知邮件主机忽视删除消息。

```
response.close( 1 )
```

下面，我们看看 Doree 代理是如何向其余的交易商发送通知消息的。下面给出的是完整的方法，其后是它的详细解释。

```
# ------------------------------------------------------
# Post a news alert
```

```
def post_news_alert():
    http_ph = ProtocolHandler.getProtocol("http")

    http_ph.setHost( "examples.pushtotest.com" )
    http_ph.setPath( "responder/htmlresponder" )

    http_ph.setType( HTTPProtocol.POST )

    body = ProtocolHandler.getBody( "http" )

    body.addParameter( "login", "myname" )
    body.addParameter( "passwd", "secret" )

    http_ph.setBody( body )

    response = http_ph.connect( 0 )
```

post_news_alert 方法使用一个 HTTP 表单发送(an HTTP form post)来发送消息。接着我们查看组成 Doree 代理的每个命令。

```
def post_news_alert():
```

首先，定义一个新方法 post_news_alert，在后面相同的代理脚本里将调用它。接着，脚本创建 HTTPProtocol 对象的一个新实例，我们可以在一个协议变量中引用该实例。关于 HTTP 协议处理程序的信息，请查看 testmaker_home/agents/signin_agent.a。

```
http_ph = ProtocolHandler.getProtocol("http")

http_ph.setHost( "examples.pushtotest.com" )

http_ph.setPath( "responder/htmlresponder" )
```

脚本指示 HTTP 协议处理程序在哪里寻找主机和 servlet，以便发送消息。

```
http_ph.setType( HTTPProtocol.POST )
```

接着，脚本通知 HTTP 协议处理程序，对主机的请求将使用 HTTP 发送(post)命令的形式。

```
body = ProtocolHandler.getBody( "http" )

# send the sign-in values
body.addParameter( "login", "myname" )
body.addParameter( "passwd", "secret" )
```

新的体对象包含了 HTTP 邮件参数。我们将第一个参数设置为具有 Doree 登录身份，第二个参数设置为具有她的密码。

```
http_ph.setBody( body )
```

接着，脚本通知 HTTP 协议处理程序使用新的体对象。

```
response = http_ph.connect( 0 )
```

最后，脚本使用特殊形式的 connect()方法，该方法通知 HTTP 协议处理程序不用接受从主机接收的任何重定向命令。通常对一个新消息发送的响应是将用户浏览器重定向到“message posted”页面。因为测试目标的原因，在这个测试里我们不需要加载重定向页面。

Doree 代理在功能上基本完备了。post_news_alert 和 read_emails 函数用于完成测试代理的动作。接着，我们将介绍如何编码 Doree 代理的其余部分，以使得这些函数能够工作。

这里是 Doree 代理的主要部分。下面给出完整的代码，其后是该代码的详细解释。

```
# ------------------------------------------------
# Doree

def Doree( agent_num,  local_name ):

    global running,  log, up_count,  record,  tick,  errortick

    random.seed( agent_num )

    up_count += 1

    pulse = Date().time       # record the start time

    sleeptime = 0

    news_end = Date().time +  ( 10 *  ( 60 * 1000 ) )

    while running:

        status = "ok"
        datapulse = Date().time

        try:
            read_emails()

            if Date().time > news_end:
                post_news_alert ( )
                news_end = Date() .time + \
                ( 10  *  ( 60  *  1000 ) )

        except java.lang. Exception,  ex:
            print  "Error thrown:"
            print  " ",ex
            print
            status = "fail"
            errortick += 1
        except Exception,  ex:
            print  "Error thrown:"
            print  " ",ex
```

```
            print
            status = "fail"
            errortick += 1

        responsetime = Date().time - datapulse

        # Log the results to a comma-delimited file with
        # these fields:
        # response time,  status,  time index,
        # machine name,  thread id number

        if record:
            totalresponsetime = Date().time - pulse
            tick += 1

            try:
                result = str( responsetime )  +  ","  + \
                status +  ","  + str( Date().time )  +  ","  \
                + local_name + "," +str( agent_num )
                print  "result:",result
                log.log( result )
            except Exception,  ex:
                print  "Error while logging results:  ",ex

        # This test agent thread sleeps after each request
        # to make the test more closely modeled to real
        # production environments where requests are
        # rarely uniform.

        time.sleep( random.randrange( 0,  5 )  )

    # We're done so reduce the thread count and exit
    up_count -= 1

# ------------------------------------------------
# Instantiate a thread

thread.start_new_thread( Doree,  ( i,  "Doree"+localname )  )
```

Doree 方法实现了 Doree 代理的主要部分。接着，我们查看 post_news_alert 方法里的每一个命令。

```
def Doree( agent_num,  local_name ):

    global running,  log,  up_count,  record,  tick,  errortick
```

该脚本将 Doree 方法定义为 Doree 代理的主要部分。Doree 方法期望接收一个代理编号标识符和一个 local_name，以便在记录测试结果时使用。另外，master.a 脚本用于实例化 Doree 代理，它设置了 Doree 脚本需要的几个全局变量。如下所示：

- running——一个标记，当主组件想要 Doree 代理运行时，将其设置为 true。
- log——log 对象，Doree 使用该对象记录结果数据。

- up_count——一个简单的整数值，用于指示正在运行的 Doree 代理的副本数目。
- tick——一个临时变量，用于跟踪成功事务的数目。
- errortick——一个临时变量，用于跟踪在错误条件下结束的事务数目。

```
random.seed(agent_num)
```

这个代理使用了一个随机数生成器，以确定每个事务后的睡眠时间。我们使用 agent_num 值作为一个种子。

```
up_count  +=  1
```

主组件使用多个线程来创建多个并发运行的 Doree 代理的副本。up_count 全局变量跟踪正在运行的代理线程数目。在关闭正在运行的测试时该全局变量是很有用的，在后面我们将了解到这一点。

```
pulse = Date().time
sleeptime = 0
news_end = Date().time +  (  10 *  (  60 * 1000  )  )
```

接着，脚本记录开始时间。这个信息在后面记录测试结果时会用到。news_end 值被设置为当前时间加上 10 分钟(1000 毫秒＝1 秒)

```
while running:
```

master.a 组件由 *running* 变量控制。当全局变量 *running* 被设置为真时，代理续续发送请求。当代理决定结束测试时，running 变量被设置为假，所有的代理都将结束其活动。

```
status  =  "ok"
datapulse = Date() .time
```

这个脚本使用 datapulse 变量来记录处理事务的时间。

```
try:
    read_emails()

    if Date().time > news_end:
        post_news_alert()
        news_end = Date().time +  (  10'(60'1000)  )
```

try 命令封装了对 read_emails 和 post_news_alert 的调用，并在 except 块中捕获错误。当 10 分钟过去后，news_end 变量将给出指示。当 10 分钟结束时，脚本使用 post_news_alert 方法将一个消息发送给其他交易商。

```
except java.lang. Exception,  ex:
   print "Error thrown:"
   print  "    ", ex
   print
   status = "fail"
   errortick += 1
except Exception,  ex:
   print "Error thrown:"
```

```
        print "    ",ex
        print
        status = "fail"
        errortick += 1
```

except 命令捕获处理一个事务时所发生的 Python 和 Java 错误。在这个测试代理示例中，将以相同的方式处理这两种类型的错误。errortick 值被递增 1，然后代理继续运行。

```
responsetime = Date().time - datapulse

if record:
    totalresponsetime = Date().time - pulse
    tick += 1

    try:
       result = str( responsetime )  + "," + \
       status + "," +str( Date().time )  + "," \
       + local_name + "," +str( agent_num )

       print  "result:",result
       log.log( result )
    except Exception,  ex:
       print  "Error while logging results:  ",ex
```

当一个新测试开始时，主组件实例化用户原型线程。为消除在 Java 虚拟机上创建新线程所产生的影响，主组件避免一次实例化所有的线程。当所有的线程被实例化后，主组件将 record 全局变量设置为真，以指示所有的代理开始记录事务的结果。

脚本将结果记录到一个文件中，文件中的内容被逗号分成以下几个域：

- response——完成每个事务需要的时间，单位为毫秒。在 Doree 示例中，一个事务包括每 10 分钟运行一次的 read_emails 方法和 post_news_alert 方法。
- Status——当为真时，表示成功的事务；为假时，表示在处理事务的时候发生了错误。
- time index——当事务开始时，当前的系统时间(以 1904 年开始的时间，单位为毫秒)。
- machine name——本地机器的名称。
- thread id number——这个代理的线程号。

```
time.sleep( random.randrange( 0,  5  )  )
```

Doree 测试代理在每个请求后睡眠，如本章的前面在我们定义用户原型的时候解释的那样。

```
thread.start_new_thread( Doree,  ( i,  "Doree"+localname )  )
```

master.a 组件通过使用 exec()命令运行 doree.a 脚本文件，来实例化 Doree 代理。脚本文件创建 Doree()方法，然后实例化包含 Doree()方法的新线程。

本节介绍了如何将 Doree 用户原型实现为一个智能测试代理。关于 Simon 和 Mira 代理的详细信息请查看 simon.a 和 mira.a 文件，通过 http://thebook.pushtotest.com 站点可以下载这两个文件。

接着，我们介绍如何构造主组件，以描述测试并实例化用户原型代理。

11.8 实现主组件

在前面的小节中，我们阐述了在脚本里实现用户原型行为所需要的步骤。在脚本注释里，许多引用都是针对主组件。主组件是领导者，它筹备、运行和关闭测试。您可以在 master.a 文件里发现股票交易公司测试的主组件，该文件可以通过 http://thebook.pushtotest.com 下载。

本节介绍如何构造主组件。主组件的实现由下面三个部分组成：

- 安装——运行 property.a 脚本来初始化全局变量和创建 log 对象。
- 运行——实例化测试代理。
- 清除——停止测试代理并运行 tally.a 脚本来记录结果。

11.8.1 安装

下面是主组件的整个安装部分的代码。脚本的后面是详细的解释。

```
# Stock Trading Test Suite

# Import TestMaker TOOL and Java objects

from java.lang import Thread
import java
import time
from java.util import Date
from com.pushtotest.tool.logger import simplelogger
import sys
import thread

# Main body of agent

exec open( scriptpath + "properties.a"  ).read()

running = 1
up_count =  0
record =  0
tick = 0
errortick = 0
first_time = Date().time

logtarget = resultspath + "results_" + localname + ".txt"

try:
   log = simplelogger.getInstance()
   log.setFileName( logtarget )
except java.lang. Exception,  ex:
   print "Could not start logging. Shutting down agent."
   sys.exit()
```

主组件的第一部分标识了脚本中使用的对象，初始化了几个全局变量，还初始化了一个新的 log 对象，用于报告代理结果。

```
from java.lang import Thread
import java
import time
from java.util import Date
from com.pushtotest.tool.logger import simplelogger
import sys
import thread
```

Import 命令标识了脚本使用的对象。TestMaker 使用 Python 的一个称为 Jython 的版本。Jython 是使用 Java 实现的 Python 语言。这个方法的优点是，Jython 脚本能使用 Python 对象和 Java 对象。标识 time、sys 和 thread 的 Import 命令指向 Python 对象。这个脚本也使用了 java.lang.Thread 和 java.util.Data 对象。

```
exec open( scriptpath + "properties.a" ).read()
```

exec 命令运行另一个文件中的脚本。在这个示例中，exec 命令运行 properties.a 脚本文件中的脚本命令。properties.a 设置了几个用户配置参数，这些参数定义了测试的范围。

```
running = 1
up_count = 0
record =  0
tick = 0
errortick = 0
```

这些脚本命令建立以下几个全局变量：

- running——通知代理继续运行。
- up_count——计算正在启动和运行的测试线程数目。
- record——通知正在运行的代理何时开始记录结果数据。
- tick——跟踪已完成的事务总数目。
- errortick——跟踪错误数。

```
first_time = Date().time
```

这个脚本记录了当前时间，以供后面分析结果的时候使用。

```
logtarget = resultspath + "results_" + localname + ".txt"
try:
    log = simplelogger.getInstance()
    log.setFileName( logtarget )
except java.lang. Exception,  ex:
    print "Could not start logging. Shutting down agent."
    sys.exit()
```

最后，脚本实例化一个新 log 对象，代理使用该对象纪录它们的结果。log 对象由 TestMaker 的 TOOL 包提供。Log 对象提供一个简单的通用方式，测试代理使用该方式将结果记录到一个通用的日志文件中。

11.8.2 运行

主组件的安装部分建立了对象、全局变量，并实例化了 log 对象。运行部分实例化测试代理，开始测试过程。接着，我们列出主组件的运行部分。然后解释这个脚本。对于完整的 master.a 脚本，请查看本书附带的代码示例目录。

```
# Calculate how many of each type of agent to run based on the
# percentage value set in Properties.a

total = 0

for a in agents:
    total += a[1]

if total>100:
    print "Warning: The total number of agents specified"
    print "          in the Properties.a file is greater"
    print "          than 100%."

# Instantiate the agents as concurrently running threads for
at in agents:

    agenttype = at[0]
    agent_count =int  (  ( float( at[1]  )  / float( total )  )

* agentcount )
    for i in range( agent_count ):
        exec open( scriptpath + agenttype + ".a"  ).read()
        time.sleep( startupdelay )

print "Threads started, we're testing!"

record  =  1

start_time = Date().time
end_time = start_time + timetorun
while Date().time < end_time:
    time.sleep( 1 )

record = 0   # Tells running agents to stop logging results
recording_time = Date().time - start_time
```

这一部分实例化了几个测试代理线程，等待它们来测试系统，然后记录开始时间和结束时间。下面是带有详细解释的相同代码。

```
total = 0
for a in agents:
    total += a[1]
if total>100:
    print "Warning: The total number of agents specified"
    print "          in the Properties.a file is greater"
    print "          than 100%."
```

第一部分根据 properties.a 文件中设置的百分比值，计算要运行的每一种类型代理的数目。agents 是一个列表对象，它包含一个元组集合。例如，agents 可以被设置为：

```
["doree",  50]
["mira",  20]
["simon",  30]
```

每个元组的第一个值利用测试代理所使用的用户原型的名字来标识，第二个值表示在测试时这种类型的测试代理所占的百分比。例如，[“doree”，50]表示 50%的并发运行代理将使用 doree 用户原型行为。

```
for at in agents:
    agenttype = at[0]
    agent_count =int  (  ( float( at[1]  )  \
    / float( total )  )  * agentcount )
    for i in range( agent_count ):
        exec open( scriptpath + agenttype + ".a"  ).read()
```

这个脚本接着实例化混合的用户代理。代码循环处理代理列表中的元组。exec 函数调用 doree.a、mira.a 和 simon.a 脚本文件中的一个函数定义。

```
time.sleep( startupdelay )
```

实例化每个代理线程之后，脚本使用 sleep( )方法给予线程一段时间，以进行一个客户连接，时间结束后，才实例化下一个线程。如果没有这种延迟，线程可能会在连接到主机之前超时。

```
record =  1
```

所有的线程都被实例化之后，改变 record 全局变量的值，通知代理开始将结果记录到 log 对象中。

```
start_time = Date().time
end_time = start_time + timetorun
while Date().time < end_time:
    time.sleep( 1 )
```

主组件然后等待测试周期完成。timetorun 值是在 properties.a 文件中设置的。

```
record =  0
```

测试周期完成。设置 record 全局变量，通知正在运行的代理停止记录结果。

```
recording_time = Date().time - start_time
```

最后，record_time 变量被设置为该测试所使用的时间，单位为毫秒。

11.8.3 清除

主组件的运行部分实例化了一些测试代理线程，然后等待测试周期完成。清除部分将关闭 log 对象，并运行 tally.a 脚本，以确定测试结果。

```
log.closeLogFile()

running = 0
```

```
duration = Date().time - start_time

exec open( scriptpath + "tally.a"  ).read()
```

主组件是领导者，它筹备、运行和关闭测试。主组件使用 properties.a 脚本中的全局变量集。11.8.4 节将介绍 properties.a 脚本。

11.8.4 用于测试配置的属性文件

股票交易系统测试的目标是重用和维护。使用属性文件来存储配置和安装信息，使得达到这些目标变得很容易。属性文件是简单的文本文件，您可以用任何文本编辑器来修改它们。属性文件使得操作测试的用户不必了解太多测试代理代码的内部细节。

properties.a 文件建立了以下几个参数：

- agentcount——并发运行的线程总数。计数值可以为任意正整数。但是如下所述，较大的设置可能影响测试结果。
- agents——一个列表对象，定义了一个代理线程所使用的用户原型，和使用这个原型的线程占 agentcount 值的百分比。例如，agents=[doree，25]告诉主组件在所创建的所有代理线程的 25%中使用 Doree 用户原型。如果 agentcount 等于 60，那么 15 个代理线程将是 Doree 代理线程(60 个代理的 25%是 15 个代理)。
- timetorun——测试周期的时间。这个值的单位是毫秒(例如，1000 毫秒等于 1 秒)。
- localname——用于标识运行测试的机器的名称。
- resultspath——日志文件位置的路径。
- startupdelay——实例化代理线程之间所需要的睡眠时间。该值随所用的处理器速度的变化而变化。

股票交易测试系统中的属性文件实际上是一个可运行的 TestMaker 脚本。主组件运行属性文件来创建变量，供主组件和测试代理组件使用。主组件使用 exec 命令运行属性文件。

11.9 实现日志组件

主组件完成三个功能：安装测试、实例化代理线程和将结果记录到日志文件中。接着，我们查看主组件创建日志文件的方法。每个开发人员、QA 分析师和 IT 管理员需要考虑他们要记录的内容，因为古老的公理“Garbage in garbage out”在测试代理环境里仍旧不变。在股票交易测试中，我们需要回答下面的问题：

- 我有正确的服务器数目吗？
- 系统如何处理负载增加的情况？
- 足够的活动将最终削弱系统吗？

如果将这些问题牢记在心，则有必要记录操作的类型、一个指示事务是否成功完成或失败的标记，以及每个事务的运行周期。这样我们可以看到，当平均事务因更多的代理线程并发运行而需要花费更长的时间时，吞吐量所发生的变化。我们也可以确定 TPS 率和每个事务的出错率(EPT)。

下面考虑代理线程如何记录结果。存在几种方式，包括：

- 在测试过程中记录我们希望报告的统计值，但不记录单独的事务。这种方法很快，并且只需要很少的资源(内存或硬盘空间)。但是它不允许我们后来回溯和重新评估结果。
- 将每个事务记录到一个 log 对象中，在内存中保存 log 记录项，直到测试完成。在测试结束时，将内存中的数据记录到日志文件中。如果测试运行了几天——有一些测试会一直这么做——则这种方法就太耗内存了。
- 将每个事务记录到一个日志服务中，该日志服务将多台机器上并发运行的多个代理线程的日志记录项收集到一个通用日志中。这种方法在使用多台机器运行测试代理时是最好的。
- 将每个事务记录到一个文件中。这种方法使得以后重新观察事务数据变得更加容易。

还有其他记录结果数据的策略。例如，由于多个代理线程并发运行，每个代理可以将日志记录到它自己的文件里，并且所有代理线程都可以将日志记录到一个文件里。

对于股票交易系统，将每个事务记录到一个文件里以供后面分析，将使得问题简单化，并且提高编码效率。为此，我们将代理线程构建为能够将它们的事务结果记录到一个同步的 log 对象中，而该对象将记录的结果存入到一个文件中。TestMaker 的 TOOL 包含一个简单的 logger 对象，它提供这样的同步日志功能。

为了新建一个简单的 logger 对象，使用下面的代码：

```
try:
    log = simplelogger.getInstance()
    log.setFileName( logtarget )
except java.lang. Exception,  ex:
    print "Could not start logging.  Stopping test."
    sys.exit()
```

然后为了记录一个结果，代理线程使用如下代码：

```
log.log( result )
```

前面的代码使用的结果值是一个简单的字符串，它能够包含任何字符串值。

11.10 避免测试可扩展性问题

主组件从属性文件中读取要创建的代理线程的类型和数目。这些线程提高了主机平台运行并发线程的能力。如果主机平台具有一个处理器，则操作系统让线程分时使用处理器。即使主机平台有多个处理器，最终线程的数目也将多于处理器的数目，操作系统将不得不划分处理能力，以处理并发的线程。

在使用智能测试代理测试服务器的可扩展性时，测试代理本身也可能存在可扩展性问题。最终并发运行的代理线程数目将超出操作系统共享一个处理器的能力。每个代理线程获得越来越少的处理器时间。当代理线程没有足够的时间来处理服务器的响应时，从代理的角度看，好像服务器的响应速度很慢。

或许，代理线程遇到可扩展性问题的最重要指示是，查看代理线程处理事务的最小、最大和平均时间。考虑表 11-3 中的测试结果。

表 11-3　测试结果示例

代 理 类 型	事　　务	时间(毫秒)
Mira	Sold shares	3275
Mira	Sold shares	3890
Mira	Sold shares	2501
Mira	Sold shares	4289

这些结果来自 100 个并发运行的代理线程。查看这些结果，最小的毫秒时间数——1 秒等于 1000 毫秒——处理一个事务所使用的时间是 2501 毫秒。最大时间是 4289 毫秒。平均时间是 3488 毫秒。

下面我们将并发运行的代理线程数目增加到 500。表 11-4 显示了每个事务的记录结果。

表 11-4　测试结果表明该测试碰壁了

代 理 类 型	事　　务	时间(毫秒)
Mira	Sold shares	37288
Mira	Sold shares	56701
Mira	Sold shares	18038
Mira	Sold shares	48500

注意：

用于完成一个事务的平均时间变为 40 秒！增加 5 倍代理线程数目后，测试代理速度减慢了 100 倍。这种结果不好。

当配置一个新测试时，您必须注意测试环境的可扩展性。忽视测试环境的可扩展性，测试结果就没有意义、会很糟糕或产生误导。

我所发现的一个成功的测试可扩展技术是，将运行增加测试代理数目的能力构建到测试代理代码中，并用一个简单的图表绘制结果。如图 11-7 所示。

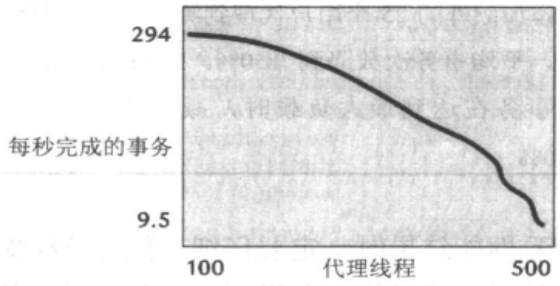

图 11-7　图中的 X 轴表示代理线程的总数，Y 轴表示每秒完成的事务的数目，本图演示了测试环境的可扩展性。它用于估测运行测试所需的测试代理主机的数目

为显示测试环境的可扩展性，该图绘制了处理的事务数相对于并发运行的代理线程数的曲线。这些测试在网络环境中运行，您可以将额外的测试代理主机添加到网络中，以创建所期望的负载级别。该图帮助您估测需要多少个测试代理主机。

另一种解决测试代理中的可扩展性问题的方式是，在代理的代码中加入 yield 命令。yield 通知操作系统代理线程位于一点上，在该点，操作系统可以将控制转向其他的线程。yield 最好是在进行一个大量计算的前后使用，或是在一个循环期间使用，该循环在一个外部过程完成时退出。例如，当代理轮询来自邮件服务器的消息时，代理线程在两个轮询命令之间可以进行 yield 操作。这是我在上幼儿园时的回忆，老师命令沙盒中的所有玩具娃娃共享。

第一次查看结果

虽然第 12 章将介绍如何构造 tally.a 脚本，以及如何分析日志来确定行为知识(actionable knowledge)，但立即查看测试代理结果的诱惑通常是非常大的。因此让我们简单看一下这些结果，然后再学习第 12 章。

本章中构造的股票交易测试，提供了测试系统级别上的可扩展性、性能和功能的一种配置方式。默认设置并发地运行用户原型。测试代理立刻显示出系统的容量、成功的操作和问题，如表 11-5 所示。

表 11-5　查看测试结果所发现的现象及得出的结论

测 试 内 容	发现的现象	结　论
我有正确数目的服务器吗	平均事务少于 5 秒。Mira 的事务比其他多出 20%。最长的响应超过 35 秒	平均事务时间是可以接受的。但是，当查看具有最长响应时间的事务时，我们看到为何 Mira 认为系统运行缓慢。从她的角度看，系统真的是变慢了。我们要了解为何 Mira 的事务完成的那么缓慢
系统如何处理用户负载增加的情况	当使用其他的测试代理时，Simon 的事务在最大负载情况下减慢了 300%。但是在最大负载情况下，Mira 的事务仅减慢了 20%	Simon 的事务仅与外部的 MarketFacts 服务进行通信，其方式是使用 SOAP 调用。这表明 MarketFacts 调用不能处理额外的负载
足够的活动将最终削弱系统吗	当添加额外的 25 个用户代理到测试中时，平均事务次数降到 150%。Doree 的事务在达到最大负载时，减慢了 300%	虽然系统能够处理现存的团队，但系统无法满足公司雇佣额外的交易商的需要

我强烈建议您第一次查看性能结果时一定要仔细。我曾经经受不起诱惑，将一个令人兴奋的邮件消息发送给宣布测试结果的项目组。我提醒您避免我过去的错误。

首次查看性能结果的问题在于它仅仅是第一次。通常需要大量的分析和组织工作来确保结果是精确的和有意义的。

在您对第一次查看到的性能结果满意之前，请阅读第 12 章。

11.11 本章小结

本章使用了前面几章中介绍的技术来命令服务利用多种协议(HTTP、HTTPS、SOAP、XML-RPC)构建一个模拟多用户原型的测试。我们经历了所有的困难——定义用户原型、模拟用户行为和理解用户目标——因此测试接近于仿真一个真实的产品环境。其他的工作集中于确定优先级，以测试系统是否满足原型的用户目标，而不仅仅是为了测试单独的功能。

测试处理了配置、测试代理的实例化和结果记录。第 12 章将介绍如何将记录结果转变为行为知识。

第12章 将测试代理结果转换为行为知识

第11章中的股票交易信息系统阐述了为实现可扩展的、可靠的和功能性的Web使能应用程序的方法学、基础架构、软件设计和协议设计。我们设计了用户原型，编写了多协议的智能测试代理，并制定对应用主机的请求。首先，我们检查正确的功能结果，接着检查主机在并发用户数目增长时所提供服务的能力。所有这些活动提供了一个近似产品的体验，从中我们可以揭示可扩展性问题、并发问题和可靠性问题。它通常也产生大量的日志数据。

查看日志数据能够看到所测试的Web使能应用程序的很多直接问题。日志文件是您能够发现问题的许多地方之一，以及您能够找到优化Web使能应用程序的地方。本章介绍了在运行测试时如何理解和分析Web使能应用程序，以及在测试结束时如何分析结果。本章中的方法能够帮助您演示系统的可扩展性、可靠性和功能性。

第11章利用前面几章中提到的技术在多种协议下(HTTP、HTTPS、SOAP、XML-RPC)操作服务，并且根据一个用户原型集合，创建了一个测试。它介绍了一个股票交易公司的测试目标、用户原型和测试代理。测试的主组件负责处理配置、测试代理线程的创建和将结果记录到一个特殊的日志文件中。本章解释了如何将记录的结果转换为行为知识(actionable knowledge)。

12.1 分析测试结果得到的信息

您可能期望回顾一下测试结果，以了解行为知识。在这一点上，亚历山大罗马教皇曾写到："缺乏知识是很危险的事情：酗酒过度或是浅尝辄止都不会有诗才的源泉。"彻底地分析测试结果能够产生行为知识，相反，仅仅查看测试数据的表面会导致可怕的问题，会危害您自己、您的公司和您的项目。因此，在介绍如何分析前面几章介绍的智能测试代理产生的测试结果之前，本节陈述了我们能够从一个测试中希望得到的信息。

结果数据提供了行为知识，但其意义在您的软件过程中的角色可能是暂时的。考虑下面的测试和行为知识如何根据其运行的测试而变化。表12-1描述了其细节。

表12-1 行为知识依赖于其运行的测试而变化

活 动	测 试	测 试 者	行 为 知 识
软件开发人员编写一个新的功能	功能性测试	开发人员	确定：功能正确，新模块完备、可以进行测试
新软件构建的提交	可扩展性和并发性测试	QA技术员	辨别优化的可能性，以提高性能和减少资源需求(CPU、硬盘、内存、数据库)
产品服务器的升级	初次公开测试	IT管理员	确定数据中心基础架构何时能够服务预测的用户等级

在每一种情形中，可以使用相同的智能测试代理进行测试，但分析结果日志时，却发现了不同的行为知识。例如，当一个 QA 分析师查看服务器上的日志文件，以进行可扩展性和并发性测试时，分析师将查看那些能够指示线程何时变为死锁状态(因为该线程正在等待来自其他线程的资源)的日志记录项。查询相同的结果日志的开发人员将会对模型的功能比较满意。因此，在分析结果开始时要理解测试的目标，并了解如何能够将该目标转换为结果。

下面将介绍几个测试目标，以及这些目标将如何被转换为行为结果。

12.1.1 目标：我们的新 Web 站点需要处理 50 个并发用户的峰值负载

想象一个公司对它的 Web 站点进行重新设计，增加了几个自定义功能。每个功能由 Java servlet 驱动。目标标识了并发用户的预测总数。并发的定义将在本章的后里给出。目前，并发意味着两个或两个以上的用户同时请求一个功能。

一种将目标转换为行为结果的技术是反过来查看目标。例如，当系统不能处理 50 个并发的用户时，我们如何知道这些信息？想象在多个时间周期中，并发运行一个智能测试代理的多个副本的情景。每个测试周期增加并发运行的代理数目。在测试代理运行的时候，系统资源(CPU 时间、硬盘空间、内存)被使用，Web 使能应用程序的整体性能将降低。日志结果将显示，当更多的并发代理运行时，事务的总数将降低。

绘制结果图表使得我们能够设置峰值负载下的可接受的性能标准。例如，在 100 个并发测试代理情况下完成的事务总数可能小于当 50 个并发测试代理运行时的 3 倍。通过绘制增长的并发测试代理数目的事务图，使得我们能够在 50~100 个并发测试代理之间选择一个数目，在该情况下系统的吞吐量仍然是可接受的。

12.1.2 目标：Web 站点注册页需要万无一失

想象一个公司促销一本新小说。公司的 Web 站点提供了一个 Web 页，让预期的用户注册以接收何时将出版该书的通知。简单的 HTML 表单可以让预期的用户填写他们的联系信息，包括他们的 email 地址。该 HTML 表单可以利用 Microsoft 的一个 ASP.NET 对象。当用户发出他或她的联系信息时，该 ASP.NET 对象需要将信息记录到数据库中，并将用户重定向到一个下载页。

这让我想起有一次我等待一位女士的电话，我邀请了她出去吃晚餐。我等了好久，直到我想到可能电话坏了。令我气愤的是，当我拿起电话筒时发现电话仍然处于工作状态。这就阻止了她把电话打进来。与此相似的是测试一个 HTML 表单。如果您不单击一个浏览器接口上的提交按钮，您就不知道服务器是否正在工作。只有单击按钮后，服务器才为您做实际的工作，让您获取资源。

一种将目标转换为行为结果的技术是理解目标的过程。考虑到获知 HTML 表单和 ASP.NET 对象工作无误的仅有方式是使用他们。每次它们被使用并且执行正确时，就表明我们达到了目标。因此您要保持测试多久，以达到“毫无差错”的目标？对测试目标的理解能够被转换为成功与失败的比率。

当 HTML 表单和 ASP.NET 对象的成功测试比率超过某一个数值时，该服务就达到了目标。例如，24 小时以内，如果目标成功测试的比率超过 95%，就达到了目标。查询日志结果来获得这个比率是相当简单的。

12.1.3 目标：顾客的月末报告请求不能减慢订购服务

一个通用的系统体系结构实践，将一个由多个应用服务器组成的负载平衡组配置在数据库服务器的前端。假设应用服务器提供两种类型的功能：一种功能使用了很多数据库查询，产生一个每月销售报告给销售商，第二种功能使用数据库添加命令，向数据库中输入新的定单。在一个 Web 环境里，这两种类型的功能可能被同时使用。

将目标转换成行为结果的一种技术是查询目标的本质。当目标为多个并发活动时，则行为结果提供了调整应用的反馈。这种调整显示了在活动类型比率变化时的系统性能。

在这个示例里，目标显示出系统减慢了速度，这是因为到月底时，客户增加了请求数据库广泛查询的报告。在这种情况下，通过使用两种测试代理的组合，可以将目标转换为行为结果：一种代理请求月报告，第二种代理下订单。测试一个具有 100 个代理的系统，并且不停改变测试代理的组合，可以显示出系统性能在不同代理组合情况下的比率。例如，使用 60 个代理请求月报告、40 个代理下订单，系统的性能是 2 倍于 80 个代理请求月报告、20 个代理下订单的情形。代理类型组合的变化和它对整个系统性能的影响使得我们能够采取一系列的动作，其方式是优化数据库和利用更多的设备提高计算能力。

12.1.4 目标总结

这里提供的示例和目标是为了向您介绍根据目标来确定动作过程的方式，以从测试结果中获取行为知识。这里多次提供了对日志结果的简单统计。虽然这些统计可能看起来很好，但是您所追求的真正目标是从测试结果中获取行为知识。

12.2 五大问题模式

浏览原始的日志结果数据通常感觉像看寒冷的冬季里闪耀的星星。观看星星越久，越多的模式就会出现。在测试 Web 使能应用程序的可扩展性、性能和可靠性的时候，有五种模式可用于识别问题和解决问题。

12.2.1 资源问题

虽然在这个领域可能没有新的软件开发技术，但今天的 Web 使能应用程序软件建立在“即时 (just-in-time)”体系结构上。一个应用在接收到请求时，将立即响应该请求。运行在主机上的 Web 使能应用程序将一直等待，直到给定的资源(CPU 带宽、硬盘空间、存储器、网络带宽)可用。这是导致应用延迟的主要原因。

在任何时候，主机服务器必须能够提供应用所需的资源，包括来自 Web 服务主机的资源。日志结果记录了当主机和 Web 服务为应用提供必需的资源时所产生的延迟。

运行 Web 使能应用程序的主机提供的资源形式包括：存储器、硬盘空间、处理器带宽和网络互连。这些年以来，出现了两种策略，可以用于理解服务器端应用程序的资源分配。独立运行的远程代理程序监视资源，并向您提供一个远程可编程接口，用于获取资源信息。这个空间的领导者是系统网络监视器协议(System Network Monitor Protocol—SNMP)标准。Java 开发人员通过 Java 管理扩展(JMX)库访问 SNMP 代理数据。关于这两者的详细信息请访问

http://java.sun.com/jmx。管理控制台(例如 HP OpenView)，提供收集、比较和显示 SNMP 代理数据的仪表板。

第二个理解资源分配的策略是在 Web 使能应用程序里构建资源监视。这可以通过给软件开发人员提供一个 API 集来完成，该 API 集可以提供从主机上获得的和由应用程序使用的资源的实时数据。开发人员将 Web 使能应用程序编写为：当处理每个事务时，将资源的使用情况写入到结果日志中。

如我们在第 11 章中提供的测试代理中发现的一样，我们选择了后面的策略用于理解资源分配。测试代理将事务数据记录到一个日志文件中。例如，代理也使用已内置的资源报告 API 来保存当前的内存大小、上一次 Java 虚拟机进行垃圾回收的时间和可用的硬盘空间量。大多数测试工具，包括第 5 章里介绍的，具有能够了解当前的资源使用情况和可用状况的脚本的方法。

在测试代理中编写代码来检查资源分配情况是一种很好的策略，因为编写测试代理的开发人员知道什么样的资源信息是重要的。通常使用一个管理控制台和 SNMP 代理来记录系统资源具有产生无关日志结果的趋势。

虽然在一个结果日志里报告的任何不希望的延迟可能看上去像因为缺少资源而产生的，但是延迟常常由其他四大类型的问题产生。

12.2.2 并发问题

将多个专用处理器添加到主板上以后，当软驱启动时一台计算机的视频显示会冻结的日子就一去不复返了。甚至在单处理器的台式机或服务器上，主板也包含了用于处理硬盘、声音、视频和网络操作的专用处理器。通过精心设计，现代的计算机成为多任务机器。在服务器端或客户端上构建了事件驱动应用程序，用于处理多个并发任务。

并发是当多于一个用户操作 Web 使能应用程序时所采用的一种度量。如果 Web 服务能够处理多个用户的操作功能，并且当处理每个用户的操作时，能够同时发出一些请求，而速度下降得很少，那么这个 Web 使能应用程序的并发性是很优秀的。

并发度量有两种记录方式。当一个应用程序运行在多台负载平衡的机器上时，对组合的结果日志的简单分析会将运行在负载平衡器之后的机器上的应用程序的并发性显示为一个单元。第二，当每台机器处理对应用程序功能的并发请求时，结果日志显示了操作系统和应用程序如何处理线程和上下文交换。

在两种方法里，我通过确定当一个事务的启动时间是在第二个事务的启动时间之后，但处于第二个事务的结束之前时，来度量并发性。一个测试代理脚本能够分析日志结果，并记录并发事务的数目。

今天的数据中心世界里，很少有人理解什么会影响系统的性能，但是几乎每一个人都是 Web 使能应用程序的用户，“并发”是造成糟糕事情的原因，这已经成为了一个流行的观点。与流行的观点相反，“并发”并不是一个坏事物。在所有的时间里，几乎没有应用程序需要所有的主机资源。遗憾的是，在服务器应用中，并发已经有了一个很坏的名声，因为它会减慢应用的性能。实际上，度量并发是度量系统共享资源的效率的一种极佳方式。

识别并发问题的典型模式是在一段时间内度量并发。在这段时间里，测试增加了并发运行的事务的数目。为了获得一个并发问题的指示器，查看事务开始和结束期间的 TPS 率。如果您发现当并发的事务数目增加时，完成的事务数目却在减少，那么您有理由怀疑出现了一个并发问题。

并发问题通常由系统瓶颈导致。多个请求堆积在一起，等待一个同步方法，该同步方法正在为前面的接收请求服务。分析结果日志能够发现非并发运行的事务和并发运行的事务所耗时的比率。

理解并发问题的模式在解决可扩展性问题和性能问题上是很重要的。但是有时并发问题会掩盖组件问题。

12.2.3 组件问题

这种问题模式的出现是因为他们重新出现在结果日志中。系统处理的活动越多，引发的问题就越多。另一方面，组件问题模式是非重复性发生的或是很少发生，以至于不重复发生，但通过分析结果日志，可以使得该模式显而易见。另外，在结果日志里，组件问题以错误的形式出现，而大多数结果日志记录项都是成功的事务。当组件失败时，它很少失败。对于组件问题，我们需要不同的策略来进行结果日志分析。

对于开发人员、QA 分析师和 IT 管理员来说，解决组件问题的最紧迫事情是确定哪个组件失败和什么样的动作、使用、负载等情形导致了失败。例如，考虑一个专用协作外连网(private collaborative extranet)服务，该服务为一个大型企业销售组织提供共享文档、交谈和参与 BBS 线程消息的能力。外连网服务的主机在 Inclusion Technologies 上，它使用了 XML 的 single sign on 机制，在 IBM 的 developerWorks 上有对该机制的描述，网址为 http://www-106.ibm.com/developerworks/webservices/library/ws-single/。在登录之后，售货员阅读简短的材料，并把问题发布到一个组讨论列表中。售货员也可以通过他们的邮件客户端定阅以参与组讨论。当消息被发布给讨论列表时，Web 用户在一个线性列表中查看该消息，邮件订阅者在其邮件账户中接收已发送消息的副本。对邮件消息的回复也将被回送给讨论组。

协作外连网系统日志表明，一个售货员大约每隔 3 天就无法收到一天信息量的邮件消息。问题是间歇性的，靠它自己解决，只会导致失败。

解决方案是创建和运行一个智能测试代理，模拟售货员的行为。代理登录、发布消息、接收和回复邮件消息。当系统失败时，代理标记产生问题的时间和步骤。标识造成该问题的背景向软件开发组表明，线程没有处理由邮件服务器抛出的异常。第二天用户再次登录时，一个新的线程将被实例化，以代替出问题的线程，邮件传递问题被神奇地解决了。理解 Web 使能应用程序的哪一部分出了问题，使得开发人员能够构建并部署对代码的修复。

解决组件问题的关键是找到有故障的组件。测试代理有助于找到该组件并观察错误。

12.2.4 竞争问题

Web 使能应用程序之间或一个应用程序中少量线程之间的竞争，会导致系统决策哪一个线程在什么时间获得了焦点。当一种类型的线程能够以绝对优势获取焦点时，将发生竞争问题。例如，在 11 章的股票交易公司示例里，系统并发地响应很多请求。设想当系统优先选择数据库查询任务时，会发生什么事情。来自作股票市场研究的人员的众多请求将造成股票交易商的请求速度减慢。

针对一个信息系统，运行多个并发的智能测试代理，将提供系统中竞争问题的惟一视图。测试代理可以改变并发运行的测试代理类型的组合，分析对性能和可扩展性的影响。例如，考虑表 12-2 中的测试结果。

表 11-2 来自多个并发测试代理的测试结果

测 试 代 理	并发运行的数量	平均事务时间
Mira	40	16 秒
Simon	30	22 秒
Doree	30	35 秒

现在假设我们调整并发运行的测试代理的组合，使得更多的 Simon 代理运行。表 12-3 显示了测试结果。

表 11-3 更改测试代理的组合后发生的变化

测 试 代 理	并发运行的数量	平均事务时间
Mira	20	50 秒
Simon	60	20 秒
Doree	20	35 秒

注意：我们将 Simon 代理的数量增加了一倍，Simon 的平均事务时间降低了。同时我们只运行了一半数目的 Mira 代理，但是 Mira 的平均事务时间减少为 50 秒。虽然很多问题都会导致这种性能上的不一致，但经验告诉我们需要查找竞争问题：查看系统的内存使用情况，查看数据库资源的日志，并查看应用服务器的设置。

12.2.5 崩溃的恢复问题

当台式机或膝上型电脑崩溃的时候，我们的第一反应就是重新启动机器，以解决崩溃所带来的残留问题。这意味着服务器需要运行很长的一段时间，并处理异常情形，如应用程序的崩溃。因此通常不选择重启服务器。取而代之的是，我们必须在 Web 使能应用程序中构建处理崩溃的高质量的代码。

智能测试代理可以用于检查错误崩溃恢复代码。测试代理能够造成崩溃，并驱动系统处理额外的并发请求。例如，考虑一个测试情形，第 11 章中的 Simon 测试代理发出 12 个研究报告请求。系统要查询数据库 12 次。如果有一个或多个查询抛出了异常，服务器必须处理异常并释放任何已使用的资源。如果处理异常的代码没有释放线程的内存、打开文件句柄和数据库连接，则测试代理是了解发生了什么情况的惟一地方。

总结五大问题模式，我们发现通常的模式在分析结果日志时出现。模式提供了检测问题和产生行为知识的方式，软件开发人员、QA 分析师和 IT 管理员可以利用行为知识解决 Web 使能应用程序中的问题。

12.3 结果分析中的主要因素

前面的小节介绍了标识结果日志中行为知识的顶层模式。这些模式显示了需要查询什么内容，下面的小节介绍如何度量您所发现的内容。度量趋向于下面四个组：

- **并发** 并发是当超过一个以上的用户操作 Web 使能应用程序时所采用的度量。当一个 Web 服务能够处理同时使用功能和发出请求的大量用户时，则该 Web 服务的并发性是优秀的。很多时候，并发问题是通过负载平衡的设备解决的。
- **延迟** 延迟是 Web 使能应用程序完成处理一个请求所花费的时间的度量。延迟具有多种形式，如从浏览器到服务器的网络传输延迟和 Web 服务完成处理请求的软件延迟。
- **可用性** 可用性是一个 Web 服务可用于发出请求的时间的度量。很多“高可用性”计算机工业软件发行商和硬件制造商都要求 99.9999%的可用性。作为可用性的示例，想象运行在一台服务器上的一个 Web 服务，该服务器每周需要两个小时的停机时间进行维护。计算可用性的公式为：1—(停机的小时数/总共的小时数)。由于每周有 168 个小时内此，每周 2 小时的停机时间的可用性为 98.8095%。
- **性能** 性能是两个错误之间的时间的度量。这是错误之间时间的一个简单的平均。例如，一个应用在上午 10:30、11：00 和 11：30 出错，性能度量就是 30 分钟。

这四种类型的标准使得我们能够在软件开发人员、QA 分析师和 IT 管理员之间谈论智能，以保证良好的系统性能。这些标准成为关于可扩展性、性能和可靠性的整体报告的组成部分。经验表明，最后的报告通常集中于两个方面：吞吐量和可靠性。吞吐量由每秒事务数来度量，并回答了问题：“我购买了足够的服务器吗？”可靠性由 SPI 度量，并回答了问题：“系统允许用户实现他们的目标吗？”

并发、延迟、可靠性和性能的度量通常根据系统的吞吐量进行计算。这种度量显示信息系统能够处理多少 TPS。

TPS 实际上是一种简单的度量。TPS 度量计算成功的事务数，然后除以完成测试所使用的时间(单位为秒)。事务定义为完成一个业务功能的所有相关请求所花费的全部时间。例如，在第 11 章的股票交易公司示例中，Mira 代理脚本登录到系统，请求股票报价并下订单。这些步骤合起来称为一个事务。

本书支持一种面向用户目标的测试方法学，从用户的角度来观察测试。因此本书着重关注来自 Web 使能应用程序的客户端的 TPS 事务是不足为奇的。这不意味着 TPS 完全由客户端来度量。当调试子系统的可扩展性问题时，通过让一个测试代理直接与后端数据库通信，以在数据库子系统中度量 TPS，是比较合适的。比较客户端 TPS 与子系统 TPS 将显示出系统中的延迟源。

虽然 TPS 能够报告吞吐量，但是另一种方法能够完成用户目标。在 Web 使能应用程序中，量化用户的成就是一件很困难的任务。所需要的是一种新的方法学，用于确定“可接受的 Web 使能应用程序的可扩展性和性能，以及描述 SPI 形式的度量结果”的标准。第 2 章介绍了 SPI。它是一种标准，具有多种用途，包括以下几个方面：

- 鼓励软件开发人员优化代码，以让用户更好地使用程序。
- 帮助 IT 管理员设计和部署容量合适的数据中心。
- 提供管理尺度来判断用户的满意度。
- 辅助 QA 分析师了解是否新的软件中出现前面已经解决过的问题。

度量并发、延迟、可扩展性和性能的 TPS 和 SPI 方法提供了一套标准，软件开发人员、QA 分析师和 IT 管理员可以利用该套标准来管理系统的可靠性。下面介绍这些度量方法如何提供误导的结果，以及如何避免产生这些错误。

12.4 结果数据产生误导的几种情形

多年以来，在分析结果日志和实现可扩展性与性能的测试过程中，我发现存在几种情形，在这些情形中，结果数据看起来是正确的，但是结论却是错误的和误导的。

12.4.1 节点问题

本书中介绍的面向用户目标的测试方法学使用一些智能测试代理来驱动 Web 使能应用程序。您或许会问："我在哪里运行这些测试代理呢？" TestMaker 提供一个方便的环境，可以在本机上运行这些测试代理。很多时候，需要配置测试，使之超过本机的功能，以运行所需数目的并发测试代理。本书建议下面两种策略：

1．配置多台机器来运行 TestMaker。使用远程控制工具——例如 ssh，它是 Linux 上的安全 shell 终端——对 TestMaker 进行远程控制。为了能够从一个命令行 shell 中运行测试代理，TestMaker 具有 Linux 和 Windows 系统中使用的 shell 脚本。在测试代理完成他们的测试后，手工将结果日志复制到单台机器上，以进行分析。

2．使用 PushToTest TestNetwork，它是一个商业产品，用于补充 TestMaker，其方式是提供下面几种能力：

- 运行大规模数量的超过单机所能承载的代理。例如，在一个 2GHz 的奔腾系统上，TestMaker 最多能够运行 500 个并发测试代理。利用 TestNetwork，运行 10 000 个或更多的并发测试代理也是可能的。
- 在多台远程机器上运行测试代理。TestNetwork 把远程系统转换为远程运行测试代理的 TestNodes，并将结果报告给中心 TestMaker 控制台。
- 保持测试代理持续运行一段时间。TestNetwork 的控制台/TestNode 体系结构将测试代理转换为袖珍服务器，该袖珍服务器能够自动处理操作。

根据我的经验，当测试节点(每台机器是一个节点)所在的操作环境很糟糕时，结果数据可能会多种多样。在一个 Web 使能应用程序测试中，我注意到：来自一个测试节点的性能结果比其他测试节点的慢 20%。由于该测试节点允许备份功能，因此导致所有的应用——包括测试代理——在备份期间运行缓慢。为避免类似的问题发生，PushToTest 出售了一种完全配置的 TestNetwork 测试节点设备。详细情况请访问 PushToTest 的网站。

12.4.2 隐藏的错误

2Wire(2Wire 主导了 DSL 配置技术，是世界上最大的 ISP 提供商)的经验表明，测试代理需要深入观察系统的响应。第 15 章介绍了 2Wire 的一种可扩展性测试的案例研究。例如，2Wire 针对 2Wire 数据中心利用 PushToTest 来运行一个可扩展的测试，确定他们是否准备就绪，以处理用户请求。2Wire 设备使用修订的 XML-RPC 协议与数据中心进行通信。虽然看起来像数据中心主机在提供 XML-RPC 响应，但是在 XML 响应中，响应的主要部分包含了错误编码。

经验表明，测试代理不仅仅将一个响应作为一个成功的事务。测试代理需要验证该响应。错误可能出现在一个响应的如下三级中：

- 传输级错误。这级错误的典型是与主机的连接问题(“connection unavailable”，“http fault”)、错误有效载荷的大小异常和不支持的协议错误。其他的问题可能为：“无法获得足够的服务器连接。”
- SOAP 响应异常。SOAP 头包含了一个异常。例如，响应表明请求的第一个元素不符合所期望的输入，因为收到了一个 long 类型的值，而不是期望的 integer 类型的值。
- 消息体异常。消息体指示出一个异常。例如，具有消息体的一个响应元素的响应包含：“MS SQL COM bridge exception: 82101.”

至少 HTTP 和 SOAP 协议提供了错误报告。隐藏的错误甚至会出现在测试邮件协议(SMTP、POP3、IMAP)里，在这里不存在错误报告的标准。例如，当收到来自 POP3 或 IMAP 主机的邮件消息时，测试代理如何知道该消息是否是来自邮件服务器的一个“返回消息(bounce)”呢，因为系统并不知道该接收器？POP3 和 IMAP 没有定义错误协议，因此测试代理查看消息体，试图检测一个“返回消息”状态。

12.4.3 Dileep 的困惑

Elsevier Science 对学习“几种应用服务器(包括 IBM WebSphere、 SunONE Application Server 和 BEA WebLogic Server)上基于 SOAP 的 Web 服务的 SOAP 可扩展性和性能标准”很感兴趣。Sun Microsystems 的 SunONE 组很友好地提供了他们的实验室、设备和工程师，以进行研究。

Dileep Kumar 是一名 Sun 软件工程师，他对测试提供了帮助。在使用 Segue Silk 方面，Dileep 得到了很好的训练。Segue Silk 是一个商业 Web 使能应用程序测试工具。Dileep 使用 Segue Silk 模拟对一个 Web 服务的 SOAP 请求，该 Web 服务运行于一台具有 SunONE 应用服务器的多 CPU SPARC 服务器上。

他编写了一个 Silk 脚本，该脚本向服务器发出一个伪 SOAP 请求。这个 Silk 脚本实际上不使用 SOAP 堆栈；取而代之的是，它制定一个 HTTP Post 命令，并将该命令发送到正在运行的 Web 服务。他测量发出这个 HTTP 请求到接收一个响应所花费的时间。令他惊讶的是，该事务时间比使用 TestMaker 发现的结果快 40%。

他忽略了 TestMaker 结果包括通过客户端的 SOAP 堆栈和 XML 分析器编组请求所使用的时间。这个经历是一个很好的教训，它教导我们在检测结果的时候，首先要确保比较相似的项。

12.4.4 消除反复

很多时候，结果分析要转变成一个消除反复(diminishing returns)的练习。每一次仔细地查看结果日志，对系统的可靠性和可扩展性都会有一些小的收益。最好的测试分析能够容易地捕获一种影响，以继续排序、过滤和支撑结果数据，因为他们觉得如果他们再多花费一点时间，额外的可扩展性和可靠性知识是能够被发现的。为了避免这个问题，我建议您首先要理解测试的目标，然后再分析数据。

12.5 再论股票交易示例

根据所有关于将结果日志转换为行为知识的讨论，您可能对第 11 章的股票交易公司示例感到奇怪。Mira、Simon 和 Doree 都会满意吗？是的，他们会感到满意，事实证明如此。但是，直到我们记录结果日志，得知系统的容量(像 TPS 中度量的一样)之后，这一点才得到了证明。

实现 Tally 代理

在第 11 章中，marster.a 代理脚本通过配置几个测试代理、运行这些代理和将结果记录到文件来进行股票交易示例的测试。master.a 代理脚本的最后操作是运行 tally.a 脚本。tally.a 分析结果日志并确定测试的 TPS 标准。本节介绍如何构建 tally 组件。

首先，我们检查整个 tally.a 脚本。然后提供一个详细的脚本解释。本书中提供的所有代码都可以通过 http://www.pushtotest.com/docs 站点下载。

```
# Script name: Tally.a
# Author:  fcohen@pushtotest.com

# exec open( scriptpath + "Properties.a"  ).read()

# Set-up variables

tallyman = "Come Mister tally man;  tally me bananas."
logtarget = resultspath + "results_" + localname + ".txt"

# First pass through the data
print
print "Tally the results:"
print "Find the minimum, maximum,  average transaction time,
error  index"
print

mark1 = 1
response_total_time = 0
response_count = 0
resp_min = 0
resp_max =  0
error_count  =  0

start_index = 0
end_index = 0

try:
    print "Analyzing data in",  logtarget
    rfile = open( logtarget,  "r"  )
except IOError,  e:
    print "Error while opening the file:"
    print e
    print
```

```
        sys.exit()

    result = rfile.readline()

    while result  != " "
            .
        # The file is in comma delimited format
        params = result.split(",")

        response_time =int( params[0]  )
        status = params[1]
        time_index = params[2]

        if status ==  "ok":
            response_total_time += response_time
            response_count += 1

            if markl:
                markl = 0
                resp_min = response_time
                resp_max = response_time

                start_index = time_index
                end_index = time_index

            if response_time > resp_max:
                resp_max = response_time

            if response_time < resp_min:
                resp_min = response_time

            if time_index > end_index:
                end_index = time_index

            if time_index < start_index:
                start_index = time_index

        else:
             error_count  +=  1

        result = rfile.readline()

    rfile.close()

    print
    print "Agents:  ",  agentcount
    print
    print "Minimum transaction time:  %d" % resp_min
    prmnt "Maximum transaction time:  %d" % resp_max
    prmnt "Average transaction time:  %.2f" %  ( float(
    response_total_time )  / float( response_count )  )
    print  "Error count:",  error_count
```

```
# Calculates the duration in milliseconds that the test took
to run
test_duration = long( end_index ) - long( start index )

print
print "Total test time:  %d" % test_duration
print "Number of completed transactions:  %d" %  (
response_count )
print "Transactions Per Second:  %.2f" %  ( float(
response_count )\
/  ( test_duration / 1000 )  )
print
print
```

tally 组件的第一个部分标识了脚本中使用的对象并初始化了变量。下面是带有详细解释的相同代码。

```
tallyman = "Come Mister tally man;  tally me bananas."
logtarget = resultspath + "results_" + localname + ".txt"
markl = 1
response_total_time = 0
response_count = 0
resp_min = 0
resp_max =  0
error_count =  0

start_index = 0
end_index = 0
```

这些变量协助脚本找到最小、最大和平均事务时间，以及一个错误索引。

```
try:
    print "Analyzing data in",  logtarget
    rfile = open( logtarget,  "r" )
except IOError,  e:
    print "Error while opening the file:"
    print e
    print
    sys.exit()
```

这个脚本打开了一个日志文件，该日志文件是由主组件创建的，并由单独的测试代理线程的数据填充。

```
result = rfile.readline()
while result  != " ":
```

这个脚本逐行分析结果日志文件中的内容。当到达文件的末端时，readline()方法返回一个空值。

```
params = result.split(",")

response_time = int( params[0])
status = params[1]
time_index = params[2]
```

结果日志文件中的数据由逗号分隔。split()方法找到每一个值，并将值的列表返回到 params 变量中。

```
if status == "ok":
```

这个分析中的每秒事务数标准仅仅计算了成功的事务。

```
response_total_time += response_time
response_count += 1
```

response_total_time 和 response_count 值记录了响应，以便日后在脚本中计算平均响应时间。

```
if mark1:
    mark1 = 0
    resp_min = response_time
    resp_max = response_time

    start_index = time_index
    end_index = time_index
```

日志中的第一个结果成为最小响应时间和最大响应时间。

```
if response_time > resp_max:
    resp_max = response_time

if response_time < resp_min:

if time_index > end_index:
    end_index = time_index

if time_index < start_index:
    start_index = time_index
```

当脚本分析结果日志的每一行时，存储了最小和最大响应时间，以备后用。脚本也记录实际的启动和结束时间。

```
else:
    error_count  +=  1
```

尽管 TPS 值仅仅包括成功的事务，但是 tally 脚本也报告错误的事务数。

```
print
print "Agents:  ",  agentcount
print
print "Minimum transaction time:  %d" % resp_min
print "Maximum transaction time:  %d" % resp_max
print "Average transaction time:  %.2f" %  ( float(
response_total_time )  / float( response_count )  )
print  "Error count:",  error_count

# Calculates the duration in milliseconds that the test took
to run
test_duration = long( end_index )  - long  start_index )
```

```
print
print "Total test time:  %d" % test_duration
print "Number of completed transactions:  %d" %  (
response_count )
print "Transactions Per Second:  %.2f" %  ( float(
response_count )  /  ( test_duration / 1000 )  )
```

正如我们所看到的一样，tally.a 是一个简单的脚本，用于计算结果日志中的 TPS 标准。tally 脚本将结果显示到 TestMaker 输出窗口。

12.6 本章小结

本章包括了我们在前面几章中介绍的内容。在第 11 章中我们创建了一些智能测试代理来模拟原型用户。这些测试代理驱动 Web 服务上的近似产品环境负载，从而使得该系统生成完全的原始日志数据。我们编写代理编码以将结果数据分析为行为数据。TPS 报告向我们显示了当该系统位于产品环境中时，从该系统中所能够期望得到的东西。

第 13 章　高容量数据中心的并发性和可扩展性

2Wire 是一个革新 DSL 系统的制造商。2Wire HomePortal 住宅网关和 OfficePortal 宽带路由器都是电话行业的第一个真正智能的、客户可安装的多服务设备。2Wire 使用修订的 XML-RPC 协议，使得 HomePortal 网关设备能够与数据中心通信，接收它的编程、更新和内容。在这个环境里，2Wire 需要回答一个重要的问题：

我们如何为上百万个 HomePortal 服务，以及如何高效地升级 CMS？

本章描述了 2Wire 如何回答这个问题。本章描述了 2Wire 组件管理系统(CMS)的 Web 使能应用程序环境、并发性和数据中心准备就绪测试的目标、所使用的测试方法学，以及如何编译和分析结果。本章对本书前面介绍的概念和方法学都进行了测试。

13.1　简介

2Wire(http://www.2wire.com)提供了领导 Internet 服务提供商的技术，包含 SBC 和 Verizon。2Wire HomePortal 网关设备通过与 2Wire CMS 数据中心通信，为最终用户的家庭或商业提供了自配置的 DSL 网络参数。在我的仓库里有一个漂亮的 HomePortal 盒子。我的本地电话公司——SBC——花费了 50 美元购买 2Wire HomePortal 网关设备。在这个设备里面有一个 DSL 调制解调器、一个 NAT 和 DHCP 路由器、一个防火墙和一个提供了增值服务的智能处理器，包括一个内容过滤服务。在过去我配置这种设备的经历并不是一件令人开心的事情。我花费了很多时间改变配置的设置，以让设备运行起来。当收到 2Wire HomePortal 设备时，我把它插好，启动配置软件，输入一个简单的键码——16 个数字——设备就自动进行配置。非常方便！

2Wire 制造了这种革新的 DSL 系统。2Wire 系统运行在一个 Web 基础架构之上——该基础架构包括 TCP/IP 路由网络、HTTP 协议和 Flapjacks 样式的负载平衡应用服务器——使用了 XML-RPC 协议的一个修订版本。HomePortal 设备包括一个自定义的处理程序，该处理程序实现了智能业务工作流行为，用于实现自动配置、更新和其他的管理设备功能。虽然 2Wire 在当前只运行了一个数据中心，但他们的计划是为他们的用户(主要的电话公司)提供一个蓝图，以构建他们自己的数据中心。

2Wire 通过开源组织了解了 TestMaker，并通过 PushToTest(http://www.pushtotest.com)来查看 TestMaker 是否适合他们的测试需要。2Wire 需要向他们的客户证明，他们准备好了大量的配置。PushToTest 测试了 2Wire 数据中心是否准备就绪、测试了 2Wire 的 CMS 系统以检测并发问题，并传递一个基于 TestMaker 的自定义测试环境。

13.2　2Wire 组件管理系统

ISP 时刻在寻找新的方法以自动化他们的服务。服务自动化是增加收入、减少支持开销和开放新的增值机会以服务用户需求的关键所在。2Wire 是提供 ISP 的领头公司，它提供自动部署的端对端的解决方案，并支持商业和家庭 Internet 网关的解决方案。

2Wire CMS 解决方案组合了一个主机数据中心与智能用户室内设备(CPE)的网关设备。因此 2Wire 系统是自动化商业和家庭对 Internet 访问的最先进的方法。

对于每一个 ISP，自动化不再是一个可选项。他们需要高效的解决方案来自动化基本配置和管理任务，包括 DHCP 和 NAT 路由配置，以及 DNS 服务配置和安全配置。另外，理解 Internet 网络配置和管理的客户越来越希望能够方便地使用网络，并减少维护工作，尤其是那些需要使用电话寻求技术支持的客户。

2Wire 为 ISP 提供了以下的商业好处。

- **减少开销**——CMS 解决方案提供的网关门户设备能够利用从一个 CMS 数据中心接收的信息进行自我配置。客户键入一个键码，系统就会处理 NAT、DHCP 和 DNS 设置的所有配置。配置是完全自动化的。
- **增加收入**——CMS 网关门户设备是可以被动态编程的，以提供增值服务。例如，一个增值内容屏蔽服务提供的保护使得错误的数据不会呈现给计算机用户。对于提供增值的解决方案来说，CMS 是一个完全可扩展的平台。
- **增加客户的满意度**——CMS 解决方案提供友好的、易用的基于 Web 的接口，最终用户能够使用该接口配置和更改选项。这些补充了现存的电话支持选项，其中电话支持选项的典型特征是受挫和长时间等待。CMS 解决方案是用户友好的。

为了完成这些目标，2Wire 设计了 CMS 系统并集成了已被证明了的 Internet 技术、已建立的数据中心体系结构和能够从多个源处获得的技术。图 13-1 显示了 CMS 解决方案的体系结构。

图 13-1　2Wire CMS 系统为 2Wire HomePortal 设备提供了一个 Web 浏览器接口。该设备使用一种修订的 XML/RPC 机制将命令和信息请求发送到数据中心

2Wire 的 CMS 设计实现了一种可靠的、可扩展的和性能良好的系统。CMS 基础架构为系统操作员提供了友好的终端用户接口和强大的管理功能。例如，CMS 能够将一个升级“推”到 HomePortal 设备处或所有的设备处。

CMS 提供了一个可插用的接口，能够随时添加新的增值服务。该设计倾向于方便当今的 Web 基础架构能够在未来被用于提供新的增值功能。

CMS 系统允许 2Wire 用户处理大型的部署。理解 CMS 解决方案如何完成大型的部署是为 ISP 提供可扩展的解决方案的 2Wire 策略的关键。2Wire 致力于提高系统的可靠性、可扩展性和可扩展性，这需要对系统有深刻的了解。

13.2.1 了解在网络上发生的事情

为了理解 2Wire 系统的体系结构，我将首先介绍 HomePortal 网关设备配置。图 13-2 在用户浏览器中显示了网关设备的一个用户接口。用户键入 ISP 提供的 20 位键码。网关设备使用该键码来识别数据中心的 URL。

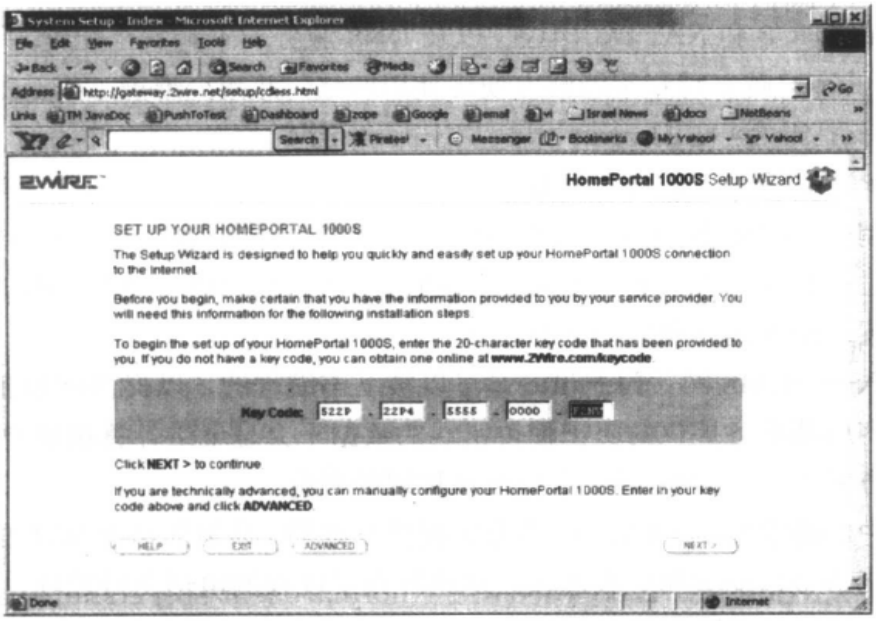

图 13-2 HomePortal 配置开始于一个 20 位数字的键码。网关设备提供了这个 Web 浏览器用户接口，用于输入键码。然后设备连接到 CMS 数据中心，以接收它的设置，包括 TCP/IP、 DNS、 NAT 和 DHCP 值

当接收到新的键码时，网关设备与 2Wire 数据中心通过设备到 2Wire 数据中心的 DSL 连接上的 HTTPS 和 TCP/IP 协议进行通信。网关设备执行一个“bootstrap”命令码。CMS 数据中心响应设备的 IP 地址、DNS 信息和 NAT 与 DHCP 配置信息，以及大量的其他设置。所有这些通信都发生在网关设备和 CMS 数据中心之间，对最终用户是不可见的。

对应用来说，2Wire 协议是有效的和安全的。有效性通过使用一个多元套接字管理器(MUX)来实现。安全性通过在网络层使用 SSL 和在应用层使用基于符号的所有权授权技术实现。2Wire 选择这些方法是因为客户端和服务器端都需要互相通信。Web 使能应用程序通常都担心开放环境下的互操作性和安全性。2Wire 知道网关设备仅与 CMS 数据中心通信。因此它仅需要关心他自己的互操作性。

从网关设备到 CMS 数据中心的每一次通信都要使用三步或更多步来授权网关设备，传递一个命令请求和它的支持数据，并接收响应和数据。对于每一步来说，这都潜在地需要多次

TCP 连接。 Internet 已经饱受 HTTP/1.0 协议的影响，在该协议中，每一步都需要打开一个 TCP 连接，用于获取每一个 URI，这将花费传输报文的时间和往返的时间，然后关闭每一个连接。对于小的 HTTP 请求，这些连接会降低性能，因为 TCP 的启动很慢，而往返需要打开和关闭每一个 TCP 连接。

万维网联盟(W3C)开发了 HTTP/1.1 持续连接和流水线来减少网络信息流量和由于打开和关闭 TCP 连接而带来的 TCP 开销的数量。但是，对于 2Wire 体系结构来说，HTTP/1.1 也有它自己的缺点。请考虑 HTTP/1.1 流水线不能支持同时的在线对象，以及在没有关闭 TCP 连接时允许放弃 HTTP 事务的情况。虽然 W3C 工作于它自己的连接多路复用器(MUX)上，但该工作从来不会让开发人员取消，因为它的 API 很复杂。详细情况请访问 http://www.w3.org/Protocols/MUX。

2Wire 工程师选择使用连接多路复用器(MUX)来建模 Caucho 的 Hmux(详细情况请访问 http://www.caucho.com)。HMux 使用简单的套接字连接来进行多个同时进行的全双工会话。图 13-3 演示了通过一个简单的开放套接字连接进行的一个典型的基于 MUX 的会话——当然，使用 MUX 的要点是几个这样的会话可以同时进行。

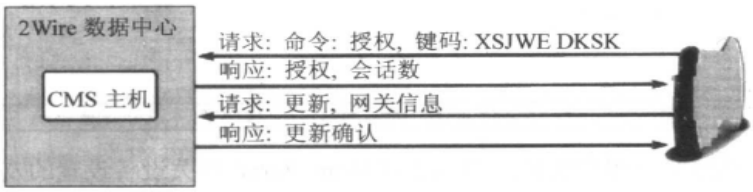

图 13-3　为了更有效地使用网络套接字，2Wire CMS 协议使用了一个 MUX 层。网关箱打开一个套接字，网关和服务器都通过打开的套接字发出请求

2Wire 协议是 XML-RPC 协议的变种，它使用 TCP 套接字协议移动 XML 数据，以提供有效的网络连接。网关设备打开到 CMS 数据中心的一个套接字连接。在请求和响应命令处理的过程中，套接字连接一直保持连接状态。和通常的客户/主机通信相反，MUX 连接允许网关设备和 CMS 数据中心主机同时向对方发出请求。XML 编码的数据有效载荷伴随着每一个命令。例如，在图 13-2 中，会话的第一部分授权到 CMS 数据中心的网关设备。XML 编码的有效载荷包含了设备键码。CMS 数据中心授权网关设备，并将授权的会话标识符返回给网关设备。

2Wire 系统使用命令协议来支持网关设备请求更新、接受它的初始设置，以及提供注册信息。该协议是可扩展的，因此 2Wire 可能增加额外的命令，而不用重新部署网关设备。

在网关设备和 CMS 数据中心之间的每一个命令都提供了所需要的交互，以透明地配置和维护 HomePortal 网关设备。这极大地为最终用户隐藏了复杂性。所有的最终用户看到的是基于浏览器的用户界面，如图 13-4 所示。您可能感到奇怪，作为技术设备的消费者，为什么每一种高技术设备没有 2Wire 级的易用性和配置呢？一句话：太复杂了。

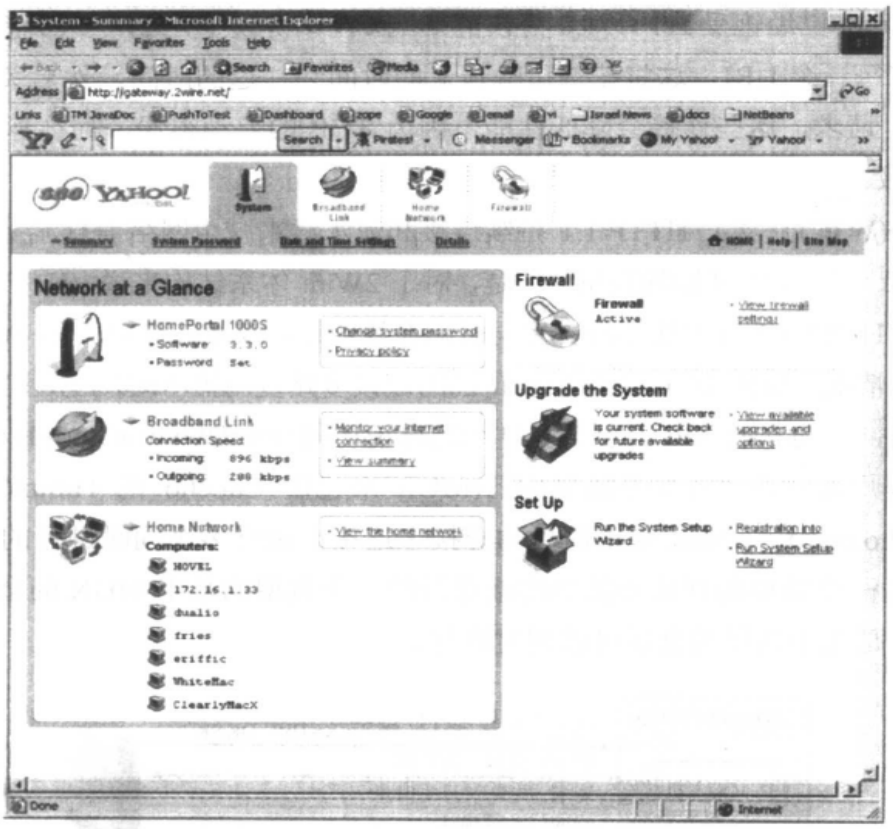

图 13-4 从 2Wire 系统最终用户的角度显示了到 HomePortal 网关设备设置的友好 Web 浏览器界面

13.2.2 复杂环境中的测试

下面考虑测试 2Wire 系统的并发问题和确定系统的可扩展性要考虑什么问题。系统有几个潜在的问题：

- 在 HomePortal 网关设备和 CMS 数据中心之间的通信协议使用定制的授权机制，通过 MUX 连接，使用一种修改的 XML-RPC 协议和一种定制的命令/响应协议。另外，系统使用 SSL 来加密 HomePortal 网关设备和 CMS 数据中心之间的通信。没有现存的商业测试工具可以用来检查这个系统。
- 由于网关设备应当只支持一个用户操作它的基于浏览器的配置接口，因此检查网关设备的功能就足矣。但是不存在可编程的方式来通知新的功能将何时被添加到设备中，设备的动态更新本质上意味着功能在没有任何警告的情况下可能会变化。
- 2Wire 数据中心使用冗余的负载平衡路由器来共享支持的网关设备。负载平衡器需要被测试，以确定它们正平等地共享可用服务器的负载。
- CMS 主机提供了支持大量网关设备所需的事务工作流。在 MUX 连接断开的时候，任何网关设备的工作流命令需要短暂的持续会话，该会话由 CMS 主机管理。糟糕实现的代码和 bug 潜在地将并发问题引入到 CMS 主机中，仅当具有足够大的负载，使得主机耗尽资源时，才能够发现这种并发问题。
- CMS 主机使用关系数据库来提供网关设备信息的持续存储。需要测试数据库的效率和容量。

虽然一些商业测试工具能够显示 CMS 性能的一些方面，但不存在单个工具能够显示所有的方面。没有单个工具能够回答系统的可扩展性和性能的所有问题。2Wire 发现 TestMaker 有能力仿真它们系统的 3 个部分，如图 13-5 所示。

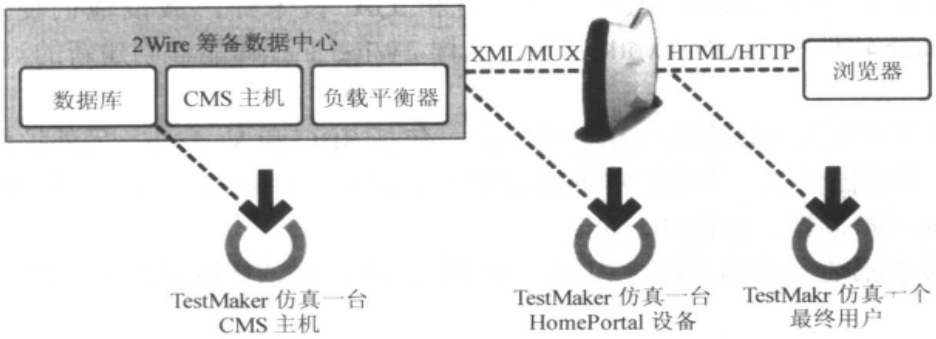

图 13-5　TestMaker 提供 HomePortal 网关设备、CMS 主机甚至是最终用户的仿真

TestMaker 在 2Wire 系统里适合于三种不同的测试。TestMaker 仿真浏览器后端最终用户的行为。TestMaker 使用本地 HTTP 协议驱动 HomePortal 网关设备，以测试网关设备的功能。第二，TestMaker 通过使用本地 CMS 主机命令协议与 CMS 主机交互，从而仿真网关设备。最后，TestMaker 通过使用 JDBC 驱动程序来驱动数据库系统，从而仿真 CMS 主机。

另外，2Wire 发现了它的软件开发、QA 和 IT 组使用 TestMaker 测试环境所带来的好处。2Wire 预想这些角色：

- **软件开发组**——编写 Java 模块，这些模块可以被插入到 TestMaker 框架中，从而实现 2Wire 网关功能。开发组编写能够调用 Java 模块的测试代理，对新的 2Wire CMS 功能进行功能测试。
- **质量保证组**——编写测试代理脚本来执行新的 2Wire CMS 功能的性能和可扩展性测试。这些测试消除了性能瓶颈，并显示对哪些方面进行优化能够产生更大的吞吐量、资源利用率和稳定性。
- **操作组**——IT 管理员将来自开发和 QA 分析的测试代理用作新的 2Wire CMS 构建的交付证明(proof of delivery)检查，也作为一个服务质量监视器。

虽然 2Wire 存在上面列出的几个潜在问题，但是它们最大的优点是 CMS 主机能够服务大量的网关设备。因此，2Wire 与 PushToTest 共同提供了一个定制的基于 TestMaker 开源项目的测试环境，并共同执行一个独立的 CMS 系统性能和可扩展性的审核：

- 我们编写一个可扩展的研究建议，以定义后端 HomePortal/CMS 支持系统的性能和扩展性的标准。我们采访了软件开发人员、QA 技术员和 IT 管理员，以理解每个组的目标和所关心的事情。结果测试标准定义了四种用户原型——原型化的 HomePortal 用户——我们从中建模智能测试代理。这个标准定义了一个代理的组合，以仿真系统中的用户需求。
- 我们创建、部署和分析一个智能测试代理系统，以确定最大的并发用户数和以每秒事务数度量的吞吐量，以及确定这些如何与一个服务器所支持的 HomePortal 网关设备数相关联。
- 我们编写了一些仿真 HomePortal 单元所需要的 XML-RPC 协议处理程序。PushToTest 然后使用 TestNetwork 来运行上千个并发的代理，以仿真真实环境，从而确定系统的功

能、可扩展性和性能，其中 TestNetwork 是 TestMaker 的商业版本，用于完成分布式测试和分析环境。关于 TestNetwork 的详细情况请访问 http://www.pushtotest.com/ppt/TestNetwork/index.html。

- 我们培训了一些 2Wire 软件开发人员和 QA 技术师，教授他们使用 PushToTest TestMaker 软件和 TestNetwork 软件，这样 2Wire 工程师能够创建新的测试并维护现有的测试。

2Wire 将项目产生的测试数据交给它的客户，包括 SBC 和 Verizon。另外扩展的 TestMaker 和 TestNetwork 环境提供了一个易扩展的测试平台，用于在基础架构、CMS 软件和部署发生变化时检查系统的可扩展性、性能和功能。

下面本章接着描述测试的方法、如何开发性能标准、测试环境和测试结果。

13.3 测试方法

2Wire 设计的 CMS 解决方案在不断增长的大型配置里是可靠的和可扩展的。虽然单独的 2Wire 系统组件是被 Web 基础架构证明的且现在正在使用的标准，但是 2Wire 仍需要开发和使用新的测试方法学、技术和工具来理解整个系统的性能。为了提出 2Wire CMS 解决方案设计的高级本质，2Wire 开发了两种测试方法学：

- **面向用户目标的测试**——2Wire 不仅测试单独的系统功能，还为原型用户开发用户原型。智能测试代理为用户原型编码。测试代理可以并发运行，并记录结果供以后分析。测试针对用户目标来度量系统的性能。
- **真实世界的设备仿真**——测试环境实现由 CMS 数据中心和 HomePortal 网关设备使用的本地协议。接着，测试驱动整个系统，像真实的网关设备一样。另外，模拟的设备可以在数据中心内部或远程位置运行。

2Wire 测试方法用两种方式来度量系统的可扩展性和性能：

- **吞吐量**——度量一个 CMS 服务器有效处理每个网关设备请求的能力。当所有的请求在预定的时间内都被处理时，则系统具有很好的垂直可扩展性。
- **可扩展性**——度量 CMS 主机数据中心响应不断增长的网关设备的能力。当客户设备数目增加，而系统的性能不受影响时，则系统具有很好的水平可扩展性。

通过设计，CMS 解决方案提供了一种轮询风格(polling-style)的系统，其中 CMS 对请求的响应由网关设备发出。系统使用 Internet 协议(TCP/IP 连接之上的 XML-RPC)供网关设备与 CMS 主机交互。这些协议用于三个增长的复杂事务类型。表 13-1 归纳了这些类型。

表 13-1 根据有效载荷的大小对事务进行分类

类 型	描 述	平均有效载荷大小
类型 1	对于请求，需要无状态地、简单地响应的服务。例如，更新 DNS 服务	1 200 字节
类型 2	需要多协议多步骤的有状态事务进行数据库访问的服务。例如，HomePortal 网关设备周期性地向 CMS 主机发出更新请求	4 500 字节
类型 3	需要多步骤事务的服务，其中这些事务需要大量的数据传输。例如：CMS 为网关设备提供的自动升级能力	75 000 字节

通过设计，2Wire 网关设备具有一种普通的行为，包括周期性地与 CMS 主机进行通信、常规用户请求帮助和偶尔的软件升级。对于测试项目的目标，我们假设 2Wire HomePortal 网关设备对操作产生的请求的数目如表 13-2 所示。

表 13-2　CMS 操作和计划的频率

操　　作	计划的频率
操作#1，HomePortal 设备检查 CMS 主机，以进行更新	每天 1 次
操作#2，HomePortal 设备接收初始设置和更新编程	启动时 1 次，然后每年一次
操作#3，HomePortal 用户请求关于设置的信息	每月 1 次
操作#4，HomePortal 接收新的编程	每年 3 次
操作#5，HomePortal 将自己注册到 CMS 主机	每年 1 次

通过实现智能测试代理中的每一种操作(Heartbeat、Bootstrap 等)，2Wire 可扩展性和性能测试仿真了真实世界的 CMS 主机数据中心，以服务大量的 HomePortal 网关设备。混合的测试代理向 CMS 测试主机数据中心并发地发出请求。例如，当利用 5 千个模拟网关设备测试一个 CMS 解决方案时，并发运行的测试代理进行了表 13-3 中列出的混合操作。

表 13-3　计划的操作的混合

操　　作	混合百分比	来自测试代理的操作
操作#1	96.960%	4 848
操作#2	0.177%	9
操作#3	2.155%	108
操作#4	0.531%	27
操作#5	0.177%	9

智能测试代理方法提供了一种方便的技术来改变操作的混合百分比。另外，具有大量的网关设备的测试可以通过增加更多的并发运行的测试代理来完成。

虽然 TestMaker 测试代理创建了接近真实世界的使用，但也需要在开始测试前运行另外几种安装操作。图 13-6 显示了所要采用的步骤的顺序。在安装阶段，一种特殊配置的测试脚本通过连接到 CMS 数据库，创建了未注册网关设备的一个限定的数据文件。在 Live 测试阶段，代理使用本地的 CMS 协议来驱动事务。分析阶段计算 Live 测试阶段记录在统计报告中的结果。

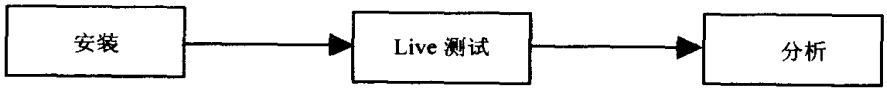

图 13-6　构造一个测试的步骤

测试结果被存储在日志文件中，并被分析以获得事务的统计，包括成功操作的 TPS 和一个加权的 SPI，SPI 中包括导致错误的操作。第 2 章中首先介绍了 SPI，它是一种加权的度量，用于衡量单个请求中的慢性能和每个测试运行期间的错误。

下面介绍一个测试是如何使用这种方法学进行的。

13.4 测试环境需要考虑的事项

由于扩展的 TestMaker 允许使用本地 CMS 协议，因此 2Wire 在进行可扩展性和性能测试时，具有很大的灵活性。例如，测试代理可以在数据中心内运行，以进行功能测试，或是在数据中心外部运行以进行可扩展性测试。图 13-7 显示了一种配置，其中测试代理在数据中心外部运行，以测试整个系统在仿真 HomePortal 网关设备数目增长的情况下的可扩展性。

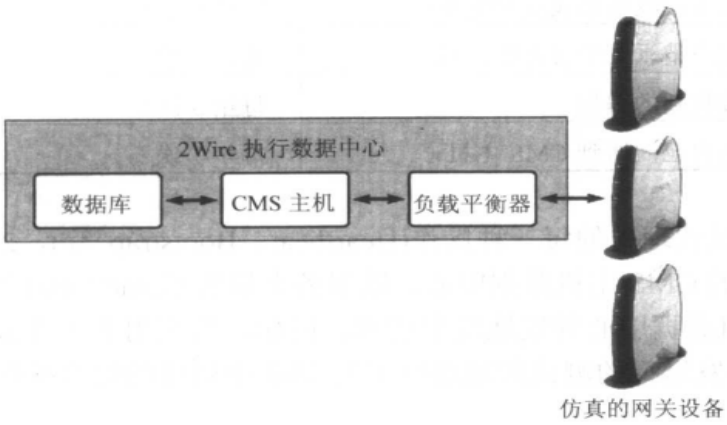

图 13-7 通过增加仿真的 HomePortal 网关设备的数量，可扩展性测试配置检查 CMS 数据中心的吞吐量

TestNetwork 环境允许 2Wire 开发接近真实条件下的系统负载，以产生尽可能有意义的结果。TestNetwork 环境使用 TestMaker 作为中心控制台，来调整 TestNode 集合上的测试的启动。因为网关设备仿真开销的原因，每个 TestNode 最多能够仿真 150 个网关设备。TestNode 将它们的日志结果数据返回到 TestMaker，用于分析。表 13-4 显示了更详细的测试环境的设备、工具和设置。

表 13-4 测试环境的设备、工具和设置

组　件	描　述
智能测试代理	代理实现了原型用户的行为。例如，通过使用本地 CMS 协议，代理可以仿真 HomePortal 网关设备。运行多个并发代理以仿真多个网关设备
TestNetwork TestNodes	每个 TestNode 是一个便宜的基于 Linux 的 RedHat 系统，带有一个 1.4MHz 的 Intel 处理器、256M 字节的内存和 IDE 硬盘机制。TestNode 运行 Sun 的 Java 虚拟机(JVM)和 TestNetwork 软件，包括 HomePortal 网关设备本地协议仿真
2Wire 产品数据中心、筹备服务器(staging server)	2Wire 操作一个产品数据中心，它具有筹备服务器，在将新软件安装到产品服务器之前，筹备服务器用于测试新的 CMS 软件。筹备服务器和产品服务器的带宽和容量是一样的
网络连接	测试代理与数据中心通过 100Mbit 的交换以太网连接进行互操作。TestNode 设备在公司的网络上运行，并被隔离成小的子网
防火墙	TestNode 和 CMS 主机之间的 TCP/IP 连接通过一个防火墙设备

这样配置测试环境后，2Wire 用实验室的设置进行可扩展性和性能测试，并推断到接近真实世界的结果。开发组采用下面的步骤来进行测试。

在各个测试之间，TestNetwork/Gateway 仿真和应用服务器/数据库需要重新启动，以减少内存泄漏，并释放可能影响测试结果的网络互连。

1. 产生每个 TestNode 的网关识别数字。一个专用目的的测试脚本查询 CMS 数据库，获取序列号。根据所使用的 TestNode 的数目，脚本将 ID 数字分成相等的几组。脚本将结果 ID 数字文件复制到 TestNode 中。

2. 确定每个 TestNode 的最大吞吐量。专用的测试脚本在本地 TestNode 上运行不断增长的并发测试代理线程。脚本通过评估线程的吞吐量来确定并发测试代理线程的最大效率。这时，脚本确定每个 TestNode 能够处理 75 个并发运行的测试代理线程。

3. 确定单个 CMS 主机的最大吞吐量。完成并发测试代理的几次运行，最初在每个 TestNode 上运行 50 个代理，随后的每次运行将在每个 TestNode 上增加运行 10 个代理。在实例化代理所花费的启动时间后，每次运行将花费 3 分钟。

4. 分析结果以找出具有最少异常数的最大 TPS 级。代理脚本自动输出一个由逗号分隔的结果日志。tally_agent 脚本然后分析日志的 TPS 值。

5. 再次运行步骤 3、4，但是这次运行测试 15 分钟。这次将发现内存泄漏、未释放的资源和数据库资源使用情况(CPU、内存、硬盘)。后来我们运行同样的测试 8 小时。然后分析短时间运行和长时间运行的区别。

使用测试环境和这些步骤，2Wire 通过增加测试代理和调整每次测试的时间，能够改变测试的内容。

13.5 测试环境的约束

本章到目前为止，TestMaker 测试环境好像是 2Wire 用于检查它自己的技术，用于检查可扩展性、性能和功能。TestMaker 是一个好的开始，但并非全部。本节描述了测试您自己的环境时需要注意的某些方面。

2Wire 提供给用户的所有测试结果都包括一个通用的否认声明：您的测试可能不同(your tests may vary)。设备、网络互连和软件的变化都将极大地影响测试结果。因此，2Wire 为用户提供 PushToTest 服务来验证客户的测试目标、方法学、测试配置和结果。另外，2Wire 通过有规律的维护和更新，不断努力提高软件的性能和可靠性。每个新的 2Wire 软件的出现都可能影响可扩展性和性能的结果。

在实验室设置中的测试可能无法有效地外推到真实的产品环境，因为在真实的产品环境中，网络延迟、TCP/IP 连接和路由问题都可能使系统不同于测试环境中的系统。

2Wire 在测试 CMS 中的目标包括度量最大的并发 HomePortal 网关设备到 CMS 数据中心的事务数。该测试不包括所有的网关设备命令。例如，网关设备提供了未经测试的动态 DNS 服务和 FTP 服务。另外，测试执行响应数据的一些小检查，用于验证有效性和完整性。如果响应包含一个 succeeded 响应代码，则那就是所有的测试代理需要在成功的响应中查找的东西。更深层的测试也需要检查响应数据，以验证有效性。

除了端对端的测试之外，2Wire 的 IT 管理员想进行网络基础架构段的组件隔离测试。例如，他们想越过负载平衡器，直接对 CMS 主机和数据库服务器运行一个测试。他们寻找基础架构的组件的一个更详的统计视图。遗憾的是，我们没有足够的时间来进行组件隔离测试。

在将 TestMaker 环境从基于 Windows 的平台移到基于 Linux 的平台时，我遇到了另一个问题，该问题也是从一种 Java 虚拟机移到另一个主要版本时所出现的问题。我发现 URL 和字符串编码的默认编码格式在不同平台中是不同的。

13.6 本章小结

使用本书介绍的 TestMaker 工具和方法学，2Wire 能够更好地理解它服务客户的能力。通过经验获取的知识使得 2Wire 仅需要购买它所需的网络和服务器设施，因此节约了大量的费用，并使得 2Wire 能够验证它们的数据中心是否准备好为大量的 2Wire HomePortal 网关设备服务。结果定制测试和监视环境检查 2Wire 系统的可扩展性、可靠性和可用性。新的测试环境由于方便了开发、QA 和操作工作的协作，在运转的头一个月就收回了成本。

第14章 为SOAP的可扩展性进行正确选择

软件开发人员目前生活的时代提供了软件开发工具、应用服务器和互连的众多选择。您所作的每个选择都会影响您所完成的应用的可扩展性和可靠性，尤其是在创建 Web 服务时。例如，您将在本章后面学到的我在研究 SOAP 编码类型时，发现通过选择一种 SOAP 编码类型，可以获得 30 倍的性能提高。通过理解 SOAP 编码类型、Web 服务开发工具、应用服务器和平台对性能的影响，我们的选择大大提高了系统的性能。

本章介绍了一个调查的结果，该调查显示了每一种选择如何立即影响可扩展性和可靠性。本章讨论了“让开发工具为我们制定关于可扩展性和性能的选择”的影响。然后提供了一些指导、工具和测试代理，以在您自己的环境中进行测试。

14.1 SOAP 流行的原因

以我的经验，当一些独立的技术革新出现交叉时，世人将享受产品、服务和技术的变化所带来的快乐。例如，电灯泡需要电的生成和灯丝纤维技术。在 Web 服务情况下，企业信息技术管理员们多年来一直欣喜他们买到了大量的计算机、服务器、路由器和其他的网络设施。由于世界经济的衰退、糟糕的股票市场、世界经济的不确定性，以及不得不对付恐怖主义和 SARS，这些信息管理员只能依靠他们现有的 Web 基础架构，通过新的软件项目来提高其团队的生产力。

与此同时，很多软件开发人员意识到他们很喜欢 XML。例如，XML 比使用 Windows 注册表和基于文本的属性文件来存储和描述应用数据要好一些。软件开发人员发现了 XML 的优点并努力在他们的应用中使用 XML。

作为一个软件开发人员，我开始注意那些期望接收一个包含 XML 编码数据的值的 API。例如，在创建 Sun Microsystems 的一个门户系统时，我发现用于创建新用户账号的 servlet 接收的域组成了 XML 文档中的用户联系信息(email、地址和电话号码)。优于一次将一个值传递给一个方法的是，该方法获取一个 XML 值，该 XML 值中包含了几个值。使用 XML 来实现应用接口对于开发人员来说是清晰的。另外，这些 XML 描述的接口能够跨平台和编程语言工作。使用 XML，所有事情看起来都像一个接口。

这些交叉技术使得 Web 服务更为流行。同时，软件开发人员又一次开始试验软件体系结构，尤其是应用事务逻辑和表示代码(presentation code)的位置。表示代码处理窗口、鼠标、键盘和其他用户接口。事务逻辑是定义了一个应用的行为和操作的一些指令。

第一代软件体系结构在一个单系统上创建了表示和事务逻辑。在第二代体系结构中，客户/服务器体系结构使用了大型的、集中控制的数据中心，在 20 世纪 60 年代，当大型机占领信息世界时，这种结构是非常流行的。在客户/服务器体系结构里，桌面系统是一个哑终端，仅仅需要显示由服务器提供的数据。早期的 Internet 模拟了客户/服务器体系结构，浏览器对一个

服务器发出简单的请求。随着浏览器在功能上的改进——applets、JavaScript、ActiveX 和 DHTML 的引入——某些系统将事务逻辑包含在台式机一端。但是，功能的主要部分仍保留在服务器上。

在“网格计算”时代，应用将事务逻辑模块驻留在台式机上或服务器上。这些模型使用 UDDI 和 P2P 技术发现对方。事务逻辑模型的多个副本也可以运行在一个由数据中心组成的网格上，从而允许失败、动态路由和功能说明。所有这些体系结构都运行在 Web 环境中，并能够提供 Web 服务。

因此即使基于 SOAP 的 Web 服务被其他类型的 Web 服务技术所取代，使用 XML 编码数据的远程过程调用仍然会存在很长一段时间。

14.2 SOAP 编码类型

SOAP 使用 XML 来编组数据，这些数据被传输到软件应用程序。大多数时候，SOAP 在软件对象之间移动数据，但是 SOAP 规范趋向于对旧的遗留系统和现代面向对象系统更有效。因此，SOAP 定义了超过一个以上的数据编码方法，以将软件程序中的数据转换为 XML 格式或是转换回来。SOAP 编码的数据被打包在一个消息体中，并发送给主机。主机将 XML 格式的数据解码为一个软件对象。

由于 SOAP 的引入，3 种 SOAP 编码方式越来越流行，并由一些软件厂商和技术提供者实现：

- **SOAP RPC 编码**，也称为第 5 部分编码，由 SOAP 1.1 规范定义，后来的 SOAP 1.2 将其定义为 RPC 编码和规范。
- **SOAP 远程过程调用文字编码**(SOAP RPC 文字)使用 RPC 方法进行调用，但是使用 XML 的 do-it-yourself(自己作)方法来编组数据。
- **SOAP 文档类型的编码**，也称为消息类型或文档文字编码。

也存在一些其他编码类型。但是软件开发人员没有广泛地采用它们，大多数是因为它们的发起人不同意这种标准。例如，早在 Web 服务诞生之初，Microsoft 提出直接 Internet 信息交换(Direct Internet Message Exchange，DIME)来编码二进制文件数据，而其他的计算机公司采用具有附件的 SOAP。SOAP RPC 编码、RPC 文字和文档类型的 SOAP 编码已经成为一个软件开发人员能够指望的编码类型。

某些开发人员没有意识到这些编码类型的存在，是因为他们使用的开发 Web 服务的工具已经为他们实现了这些编码类型。例如，BEA WebLogic Workshop 提供了快速和有效的 JWS 接口实现。JWS 实现了一套 API 和一些文件的一个标准描述，JWS 引擎使用这些文件自动配置服务器上的 Web 服务。JWS 为您自动创建 Web 服务部署描述。因此您仅需要定义公共 Java 方法，JWS 发布 SOAP 代理来访问这些方法。这使得开发更加容易，但是也可以说有很多工作要做。我们将在本章的后面详细讨论这个问题。

在讨论 SOAP 编码类型对性能的影响之前，您需要理解这些类型之间的不同点。图 14-1 显示了一个 SOAP RPC 编码调用的整个堆栈。

SOAP RPC 给开发人员提供了最简单的编码类型。开发人员调用远程对象，传递所需的任何参数。SOAP 堆栈将参数串行化为 XML，使用传输(如 HTTP 和 SMTP)将数据转移到目的地，接收响应，将响应反串行化为对象，然后将结果返回到调用方法。SOAP RPC 处理了所有的编码和解码，甚至是对于非常复杂的数据类型，并自动绑定到远程对象。

现在，想象您是一个开发人员，已经具有某些 XML 格式的数据。SOAP RPC 也允许 XML 数据的文字编码作为单个域，该域被串行化并被发送到 Web 服务主机。因为只有一个参数——XML 树——SOAP 堆栈只需要串行化一个值。SOAP 堆栈仍然处理传输问题，以获得对远程对象的请求。堆栈将请求绑定到远程对象并处理响应。

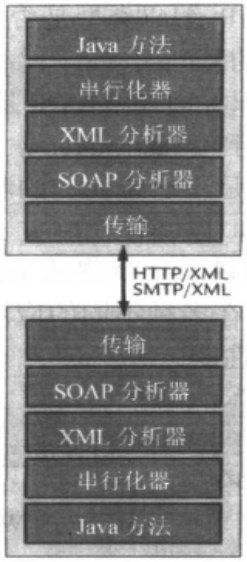

图 14-1　Java 方法通过使用 SOAP 堆栈和 SOAP RPC 编码来调用 Web 服务

最后，在 SOAP 文档类型的调用中，SOAP 堆栈将整个 XML 文档发送到服务器，而不需要一个返回值。消息能包含任何 XML 数据，只要该数据适合远程服务。在 SOAP 文档类型的编码中，开发人员处理任何事情，包括确定传输(如 HTTP、MQ、SMTP)、编组和解组 SOAP 封装体，分析请求和响应中的 XML 以发现所需的数据。

图 14-2 显示了 3 种编码系统的比较。

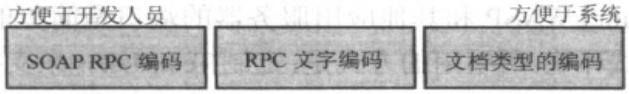

图 14-2　SOAP 编码类型为软件开发人员提供了更大的生产率，但是需要牺牲性能和系统资源的开销

SOAP RPC 编码对于软件开发人员来说是最简单的；但是牺牲了可扩展性和性能。SOAP RPC 文字编码需要软件开发人员处理更多的 XML 分析，但是需要 SOAP 堆栈的较少开销。SOAP 文档文字编码对软件开发人员来说是最困难的，但是需要很少的 SOAP 开销。

为什么软件开发人员会觉得 SOAP RPC 比较容易？使用这种编码类型仅仅需要在您的代码里定义一次公共对象；SOAP 堆栈解组请求参数，将它们放置到对象中，并直接将它们传递到您对象的方法调用中。否则，您需要分析 XML 树来发现您需要的数据元素，然后需要调用公共方法。

这里有一个意见供您分析 XML 数据：由于您更了解 XML 树中的数据，您的代码将更有效地分析数据，比通常的 SOAP 堆栈代码要好。正如我们将看到的一样，当度量 SOAP 编码类型中的可扩展性和性能时，就会发现这种情况。

但是在我讨论这些内容之前，再看一下企业信息系统管理员如何理解 SOAP 编码类型和可扩展性。

14.3 简单对象访问需要简单的测试

Elsevier(www.elsevier.com)是研究科学、技术和医疗工业方面的内容发布(content publisher)的先驱。Elsevier 现在使用了一个内容发布平台，该平台使用 SOAP 来创建应用编程接口。Elsevier 的信息管理员需要知道他们选择的 SOAP 编码类型是否能够扩展和每天处理上百万的事务。他们的决策影响着 Elsevier 将如何投资新的基础架构。随着时间的过去，他们需要知道他们自己的软件的新版本、应用服务器软件的新版本和平台的改变是如何影响可扩展性和性能的。

Elsevier 通过开源组织了解到 TestMaker，并通过 PushToTest(详情请访问 www.pushtotest.com)来查看 TestMaker 是否适合他们的测试需要。Elsever 利用 PushToTest 对 SOAP 堆栈和编码类型进行独立地审计，以回答他们关于系统性能和可扩展性的问题。结果测试环境如图 14-3 中所示。PushToTest 提供了一个测试 Web 服务(TWS)，该服务处理 RPC、RPC 文字和文档类型的 SOAP 消息，并运行在多种不同的应用服务器上。该环境由一组智能测试代理完成，用于检查 TWS 的可扩展性和性能。

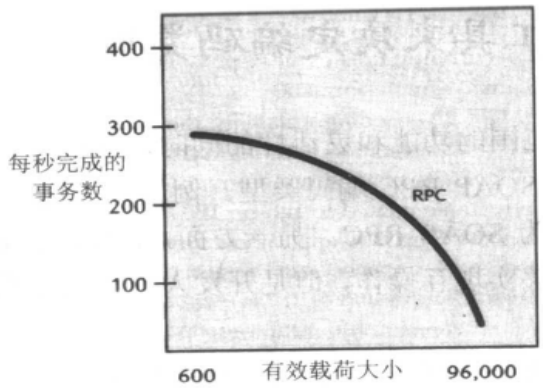

图 14-4　SOAP RPC 编码：可扩展性随着有效载荷大小的增加而变得明显

如图 14-4 所示，当进行请求时，测试代理每秒记录了 294 个事务，其中响应 SOAP 封装度量了 SOAP RPC 编码数据的 600 个字节。随着测试代理增加响应的大小，每秒事务数急剧下降。当进行 SOAP RPC 编码数据的 96 000 个字节的请求时，代理仅度量了每秒 9.5 个事务。

测试环境使用 SOAP 文档类型的编码时，性能有很大的提高，如图 14-5 所示。

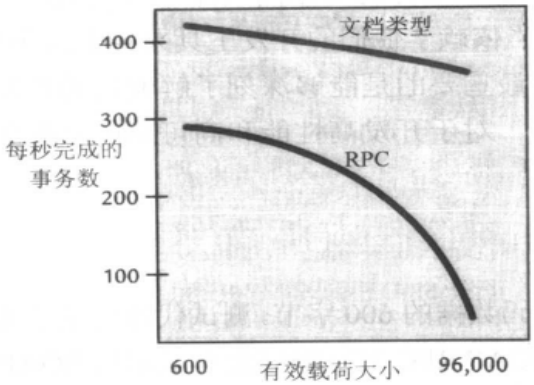

图 14-5　文档类型的编码：随着有效载荷大小的增加，性能保持相对稳定

根据文档编码数据的 600 字节，测试代理度量了 422 个 TPS。回顾对于同样大小的请求，SOAP RPC 编码的请求给予我们 294 个 TPS。另外，当测试代理增加响应大小，我们使用文档类型编码的响应时，TPS 值并没有显著降低。

当测试环境使用 SOAP RPC 文字编码时，我发现了有效的折衷，如图 14-6 所示。

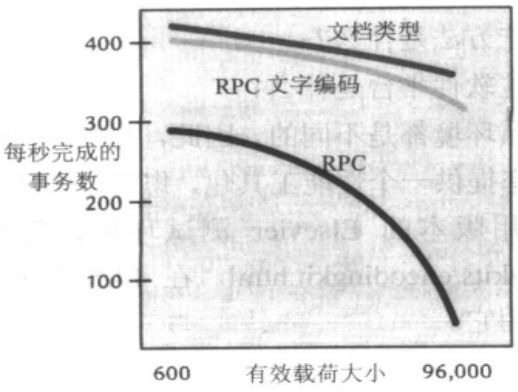

图 14-6　SOAP RPC 文字提供了 SOAP 文档类型编码的良好性能，分析 XML 数据只需做很少的工作

14.4 是否让开发工具来决定编码类型

SOAP 编码类型提供大范围的功能和灵活性的同时，也引入了互操作性问题。Java 平台上的大多数 SOAP 工具默认为 SOAP RPC 编码类型。例如，当使用 IBM WebSphere 应用开发器时，默认的编码类型被设置为 SOAP RPC。另一方面，.NET 开发工具默认情况下实现文档类型的 SOAP 调用。两者都能够实现互操作，但是开发人员需要熟悉不同的编码类型以避免问题的发生。

为了更加方便软件开发人员，这些工具应该考虑到可扩展性。这就是 Microsoft 和 Sun 公司在硅谷软件开发研讨会上讨论关于 J2EE 和.NET 的相关优点时所强调的。详细情况请访问 http://www.sdforum.org。Microsoft 争辩说，作为一个完整解决方案的惟一提供者，它能够更好地服务开发人员。在讨论会的另一方，Sun 觉得开发人员应当选择一些工具，将这些工具集成为一个解决方案。这种自上而下相对于自下而上的争论渗透到两个公司的开发工具中。例如，来自 Sun 和 Microsoft 的代表被要求解释，为什么开发人员应当选择 SOAP RPC 编码，而不是 SOAP 文档类型的编码。Microsoft 的代表给出了某种技术上的回答，但他们认为这个问题是没有意义的，因为开发人员应当依赖于他们的开发工具来决定编码类型。

对软件开发人员来说，最重要的是能够深刻了解他们的开发工具和环境。理解每个工具处理 SOAP 编码类型的方式，对于开发高性能和高可靠的软件项目来说，这是一个非常重要的因素。

是否能逃避现实

根据 SOAP RPC 文字编码数据的 600 字节，测试代理度量了 422 个 TPS。这几乎就是 SOAP 文档类型请求的性能记录。SOAP RPC 文字编码没有显示出 SOAP RPC 编码性能的急剧下降的 TPS 功能。

在这一点上，您可能会提出这样的疑问：

- 这是真的吗？SOAP 的可扩展性很差吗？
- 应用服务器间的性能差别是什么？IBM WebSphere 比 BEA WebLogic 执行得更好吗？
- 创建可靠的 Web 服务所要考虑的最重要的问题是什么？
- 我需要什么设备？
- 部署 Web 服务的最佳方法是什么？
- 创建 Web 服务的最佳软件平台是什么？

根据我的经验，每个产品环境都是不同的。因此，与其针对所有的应用服务器和编码类型来回答您的问题，还不如给您提供一个性能工具包，您可以下载并在您自己的产品环境中使用它。我已经制作了一个通用版本的 Elsevier 测试环境，您可以免费下载使用，网址为 http://www.pushtotest.com/ppt/kits/encodingkit.html。在 14.5 节中，我将介绍这个工具包，并教授您如何在自己的环境中使用它。

14.5 性能工具包

企业信息管理员对 Web 服务基础架构的选择，对于配置良好的性能、高可靠性和可扩展性的系统以运行 Web 服务来说，是非常关键的。我每天会接到软件开发人员、QA 技术员和 IT 管理员的电话，得知他们想要知道：

- 创建可靠的 Web 服务所需要考虑的最重要的问题是什么？
- 我需要什么设备？
- 部署 Web 服务的最好方法是什么？
- 创建 Web 服务的最好的软件平台是什么？

阻止我立刻回答您的问题的几个原因如下：

- 您使用的应用服务器是一个移动目标。流行的商业和开源软件厂商不畏疲劳地提高性能、增加功能，以及提高使用他们产品和技术的技巧。性能和可靠性特性随着他们产品的新版本的变化而变化。
- SOAP 堆栈中的底层代码库也是移动目标。例如，大多数基于 Java 的 Web 服务应用使用流行的 Apache Xerces 库来分析 XML 数据。Xerces 不仅复杂而且速度很慢。更换新的版本或是替换为其他的库都很大地影响着 Web 服务的性能和可扩展性。
- 您的网络设施(负载平衡器、路由器、服务器设备和存储设备)对每个人来说都是惟一的。这使得一对一(apples-apples)的比较是非常困难的。
- 随着软件应用的新构建被应用到您的产品环境中，您自己的软件开发工作将会提高您的软件应用的性能和可扩展性。

因此本节并不是尽量回答您的问题，而是主要介绍如何理解您自己环境中的 Web 服务的可扩展性和性能。

Web 服务性能工具包为信息管理员提供了可靠的和可重复的性能度量工具，以用于度量驱动基于 Web 服务业务的系统。这个工具包可以应用于任何基于 Web 服务的系统。但是，本书中的示例是专门针对 SunONE 应用服务器、BEA WebLogic 服务器和 IBM WebSphere 应用服务器的。首先我给出这个工具包的内容概况，然后介绍怎样将这个工具包应用到应用服务器环境中。

14.5.1 如何获得性能工具包

可以在 PushToTest 网站上免费下载 Web 服务性能工具包。网址为 http://www.pushtotest.com/ptt/kits/index.html。PushToTest 站点提供最新的软件、以前版本的存档和关于这个工具包的 FAQ，以及各种技术支持服务，包括用户和开发人员的邮件支持列表。

这个工具包以 Zip 压缩格式进行发布。解压缩文件，安装这个工具包中的所有内容。您也需要从 PushToTest 站点上下载 TestMaker。第 5 章中曾详细地描述了 TestMaker。这个工具包包含了运行在 TestMaker 中的测试代理和运行在 Java 1.4 或更高版本所运行的地方(包括 Windows、Linux 和 Macintosh OS X)的测试 Web 服务。其他允许 Java 的平台能够运行这个工具包，但是没有具体测试过。

TestMaker 每个月更新一次，以提供 bug 修复和新的特征。在新的版本可用时，PushToTest 网站提供了免费的邮件通知服务，以发送邮件警告消息(PushToTest 不公布或共享邮箱地址)。

14.5.2 安装性能工具包

安装这个工具包很容易。通常，很难安装和配置测试 Web 服务。在性能工具包中您会发现：

- 易读的指令描述了如何创建和测试 SunONE 应用服务器、BEA WebLogic 服务器和 IBM WebSphere 服务器上的 Web 服务。本章由于篇幅限制，仅介绍测试 BEA WebLogic Workshop 上的 Web 服务。
- TWS——一个免费的开源 Web 服务，它模拟了一个数据库驱动的 Web 服务。TWS 响应多种 SOAP 编码类型。
- Performance_Agent(性能代理)——一个免费的智能测试代理，它将决定 TWS 的可扩展性和性能。Performance_Agent 对 TWS 发出并发请求，并以每秒的事务数来度量性能。
- TestMaker——一个免费的开源软件，用于运行 Performance_Agent，以及用于创建您自己的 Web 服务测试代理。第 5 章中有关于 TestMaker 的详细信息。

性能工具包为 ZIP 格式的压缩文档，包括 HTML 文档、测试代理脚本文件和 Java 源代码文件。请使用解压缩软件将文件解压到本地计算机中。在解压缩的文件中，Index.html 文档包含了工具包的安装、配置和 how-to 指导。

PushToTest 提供了大量的支持选项，以帮助您使用这个工具包。这些支持由 PushToTest 组织免费提供。另外，也提供了负费支持协议。详细情况请访问 http://www.pushtotest.com/ptt/support.html。

14.5.3 开始安装

解压这个工具包之后，我建议您花几分钟时间了解一下它的组成。查看一下文档和源代码文件。大多数代码都是自记录(self-document)的。然后按照下面的步骤安装该工具包：

1. 如果您还没有做的话，就下载并安装一种受支持的应用服务器。Sun、BEA 和 IBM 提供了免费试行版本的软件：SunONE 应用服务器、BEA WebLogic 服务器和 IBM WebSphere 应用服务器。本章只介绍使用 BEA WebLogic Workshop。
2. 安装测试 Web 服务。
3. 下载并安装 TestMaker。
4. 运行 PerfCheck 代理来测试 TWS 性能。PerfCheck 是一个使用 TestMaker 编写和运行的智能测试代理。PerfCheck 包含 Tally 脚本并显示 TPS 和其他的 TPS 性能指标。

下面我们看一下这个工具包的组件以及安装、配置和运行该工具包所需要的步骤。

1. 测试 Web 服务

TWS 是一个 Web 服务，它模拟了一个典型的数据库驱动的 Web 服务。TWS 使用 SOAP RPC 编码的请求并返回一个响应。响应包含由请求的一个参数决定长度的乱语单词。对于测试应用服务器平台的性能、可靠性和可扩展性来说，TWS 是一个简单而容易的选择。

这个工具包在多种应用服务器上实现了 TWS。对于 BEA WebLogic，我使用 BEA 的集成开发环境，称为 Workshop。Workshop 的设计者以一种清晰的方式给出了在 SOAP 和 WSDL 规范中发现的多种选项。如图 14-7 所示。

Java 类中的公共方法被自动转化为基于 SOAP 的 Web 服务。例如，图 14-7 显示了 responder_rpc 类的单个公共方法，称为 respond。最右边的属性面板显示了响应的 SOAP 选项。图 14-7 显示了为 respond 方法更改 SOAP 编码类型是非常容易的。这些选项包括：document、rpc 和 default。WebLogic 的默认值选择 SOAP 文档-文字(document-literal)类型的编码。

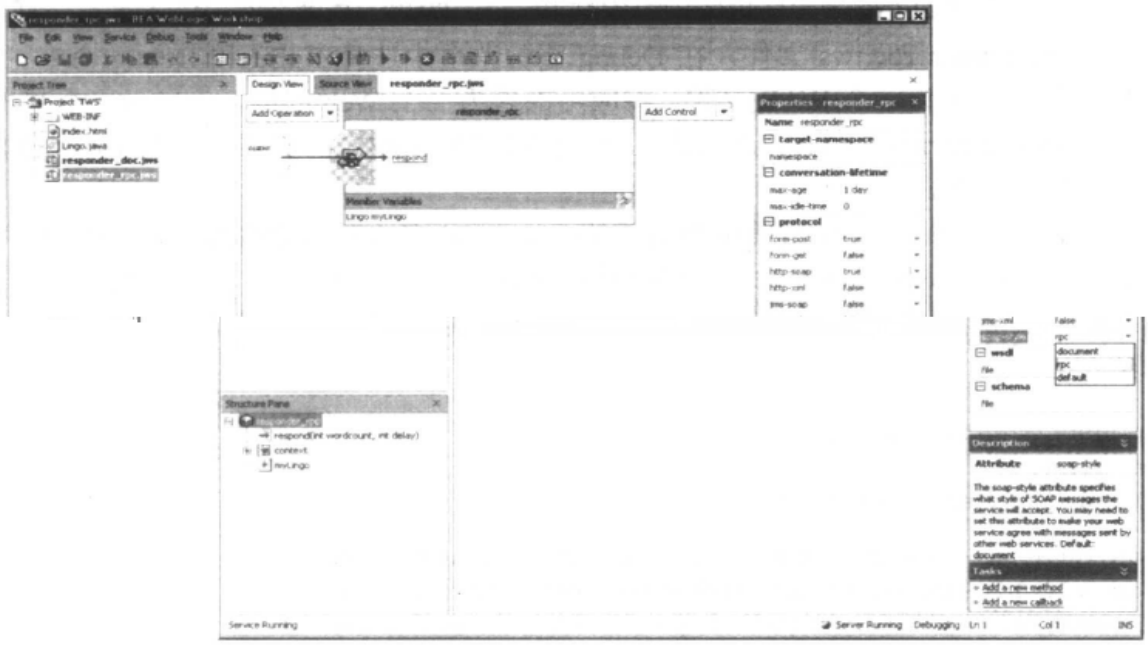

图 14-7　Workshop 使得很多 SOAP 选项很容易被调整，包括设置 SOAP 编码类型为文档类型或 SOAP RPC 类型

在 Design View 选项卡中选择一个 SOAP 编码类型能够改变 Java 代码中的 JWS 头。Source View 选项卡中可以看到这些改变。JWS 为您完成这些工作。例如，Source View 选项卡中 respond 方法默认情况下的声明如下所示：

```
import weblogic.jws.control.JwsContext;

/**
* @jws:protocol form-get="false" soap-style="document"
*/

public string respond(int wordcount,  int delay)  { ...
```

这段代码对于 Workshop 包装部署描述符来说，已经是足够了，部署描述符用于通知 JWS 引擎部署一个 Web 服务，该 Web 服务具有一个 respond 方法，且该方法使用 SOAP 文档类型的编码。

当我们在 Design View 选项卡中将编码类型改变为 rpc 时，Java 代码为：

```
import weblogic.jws.control.JwsContext;

/**
* @jws:protocol form-get="false" soap-style="rpc"
*/

public String respond(int wordcount,  int delay)  { ...
```

测试环境完成时，Elsevier Science 能够理解它的系统的可靠性、可扩展性和功能。这种测试代理技术将被新构造的 Elsevier 软件所使用，且 Web 服务应用服务器将变得可用。正在计划的测试环境也将提供回归(regression)测试、功能测试，以及作为服务证据监视器。

14.7 本章小结

在本章里，我们发现软件开发人员在构建 Web 服务系统时，具有很多种选择：SOAP 编码类型、Web 服务开发工具和应用服务器。本章显示了一个调查的结果，该调查显示了每种选择如何立即影响可扩展性和可靠性。

Elsevier 采用 SOAP 作为他们创建下一代内容聚合系统的标准方式。我们介绍了 Elsevier 如何开发一种新的方法学和测试环境，以检查各种 SOAP 实现(包括一些应用服务器、SOAP 堆栈和实用工具)的可扩展性和性能。

第 15 章　电子邮件环境中的多协议测试

Saint Leo 大学(http://www.saintleo.edu)是世界上第三大在线高等教育提供者和第六个为 U.S 军队提供高等教育的提供者。IT 职员通过搜索开源测试工具发现了 TestMaker，他们使用这些测试工具检查邮件系统在负载下的可扩展性和性能。在几个月中，他们的学生、教员和管理者将返回一个新的任务，并开始使用一个新的邮件系统。他们给自己提出的问题是："我们购买了正确的邮件软件包、网络带宽和服务器吗？"

本章提供了回答这个问题的一个调查结果。讨论了大学对一个邮件环境进行一个多协议的测试所需要的东西，以及测试工具和方法学的选择，详细描述了用于进行一个测试所使用的测试代理代码，并介绍了 Saint Leo 大学是如何分析结果的。

15.1　需要和益处

现在的大学依赖于邮件。大学的信息系统将为学生们提供服务，包括住在校园内的学生，或是住在校外的学生，以及参加卫星学校任务的学生(他们将很少在校园内)。另外，教员使用同样的邮件系统来安排教学任务、将任务提供给学生，以及与管理者和其他教员通信。

IT 管理员正在作一件非常困难的平衡工作。维护邮件系统的目标是使得带宽、服务器和路由器设备高效，而不会把系统逼迫到边界条件。在边界条件下，硬盘耗尽空间，用户不能访问系统，因为网络连接超载，所以系统管理员不能查看这些问题。IT 管理员需要平衡"对效率的需求"和"对可获得性与系统准备就绪为用户提供服务的需求"。

测试邮件系统为大学提供两个好处。它在学生、教员和职员遇到问题之前，揭示当前配置中的问题。通过提供更好地理解邮件系统如何处理用户负载，大学可以避免购买一些不必要的服务器和路由器，以及与更好的 Internet 服务提供商联系，从而节省金钱。

像其他的机构一样，预算由大学的效率需求来驱动。因此，IT 管理员寻找开源工具来节省经费预算，而不必花钱购买商业软件测试工具。他们被 TestMaker 所吸引，因为它的多协议支持。TestMaker 测试代理驱动邮件服务，像真实的用户使用邮件服务的本地协议一样。例如，一个学生使用 MicrosoftOutlook Express 来驱动邮件系统使用 SMTP、 POP3 和 IMAP 协议。与此同时，一个学生使用浏览器的 HTTP 协议检查邮件消息。TestMaker 测试代理使用本地 HTTP、SMTP、 POP3 和 IMAP 协议，如同学生所做的那样。

IT 职员已经使用 TestMaker 来创建他们自己的测试代理；但是他们在进行测试时，通常时间很短，一般在几周之内，某些工程师很喜欢在任务之间休息一下。Saint Leo 大学询问 PushToTest，要求它提供顾问服务，以编写测试以及对新的邮件系统进行一个可扩展性调查。

15.2　限定项目的范围

Saint Leo 大学提供在线大学课程，因此这些在线系统的更新是其运作的一部分。在它的最大的革新中，Saint Leo 大学改变其基础架构，以为学生和职员提供更多的内容，其方式是增加

一个 Web 端口。大学需要将多个位置处的多数据系统和技术集成为一个无缝的在线系统，以管理它的在线应用和表格，并为校友和学生开发在线讨论组。该项目包括升级邮件的能力。对于此，Saint Leo 大学选择 Stalker 软件的 CommuniGate Pro 消息服务器。

CommuniGate Pro 是一个建立在大学校园之间的消息服务器。该软件为用户提供一个 Web 接口和一些标准的 SMTP/POP3/IMAP 接口，用于检查和发送邮件消息。软件安装在便宜的基于 Intel 的服务器上，并提供低维护的管理来使其运行。虽然软件被很好地设计，但人们却很少听闻，运行一个高效的邮件系统是对网络带宽、服务器和路由配置的需求。该大学每学期为用户提供 12 000 个邮件账号。首次展示新的邮件系统时，该大学需要知道他们的系统、数据中心、网络带宽和软件配置是否已经准备就绪。

在开始项目之前，PushToTest 首先进行了一个调查，如表 15-1 所示。

表 15-1　关于邮件测试项目范围的调查

问　　题	IT 职员的回答
在任意指定时间内邮件系统上的并发用户的最大数量是多少	1 000
何时您需要使用峰值，什么原因	峰值的使用发生在上午 8：00 与上午 10：00 之间。原因：学生/职员接收邮件
您的用户将用于连接到邮件系统的 Internet 连接的速度和本质是什么	我们当前具有一个 3MB 的管道，用于 Internet 连接(这将在不久的将来升级)。用户将在校园内或校园外使用 Outlook、Netscape 和一个 WEB > 接口连接到邮件系统
您将提供 IMAP、SMTP 和 POP3 服务吗？还有其他的服务吗	IMAP、POP3、SMTP，以及一个使用 IMAP 的 Web 接口。我们也使用 CommuniGate MAPI 协议将选择的(100-200)教员和职员连接到服务器上的日历服务
在您的邮件系统中使用防广告软件工具吗？如果使用，它们是什么	使用 Real Time Blacklists
在您的邮件系统中使用防病毒软件工具吗？如果使用，它们是什么	是的。Sophos
如果您想描述三种类型的邮件系统用户(原型用户)，他们将是谁？他们将怎样使用邮件系统	用户 1：校外学生。使用 SLU 邮件和 SLU 职员通信。用户 2：校内学生。使用 SLU 邮件和 SLU 职员或亲友通信。用户 3：教员/职员成员。使用 SLU 邮件和同学与其他职员通信。大量的、MicrosoftOffice 附件
使用什么管理策略来管理每个邮件账号？例如，最大存储分配，最大的邮件消息数	一个学生的最大信息存储是 20MB。教员/职员的限制为 40MB。当然对很多用户也赋予他们一些特例。我们将特例限制为 200MB 的存储量。邮件的最大数量没有设置。我们使用 Sophos 插件，限制每小时最多发送/接收 5 000 个消息
您怎样提供新的用户账号	存储限制为学生 20MB、教员 40MB。学生账号以批处理模式每周两次在系统中运行。在增加教员/职员时，他们必须从 DR 部门获得签字
您有多少技术人员来操作邮件系统？他们典型的技能和经验是什么	我们有两个帮助桌面技术(help desk tech)来安装账户。我们有一个 FTE 位置来维护和管理服务器

调查和 IT 职员的回答告诉我，使用 TestMaker 测试大学邮件系统是再恰当不过的了。我也使用 Stalker 软件来获知测试中的输入。它们给出积极的选项，并提出关于项目的测试范围和测试目标方面的建议。如您在 15.3 节中将看到的，调查在定义测试本身方面，也给出了一个很好的开始。

15.3　测试设计和方法学

本章前面一节中给出的调查提供了很多关于如何设计测试和测试方法学的建议。测试目标是获知他们现有的系统配置、网络带宽和服务器配置是否能够服务于大学用户团体。PushToTest 设计了测试，用于模拟一组虚拟用户来加载系统。虚拟用户记录完成每个事务所花费的时间。然后测试分析结果，确定系统在负载下的吞吐量，使用 TPS 进行度量。另外，当测试运行时，IT 职员会首先看到测试对于系统资源(硬盘、CPU 带宽和网络 IO)和网络带宽的影响。

测试设计包括图 15-1 中描述的资源。

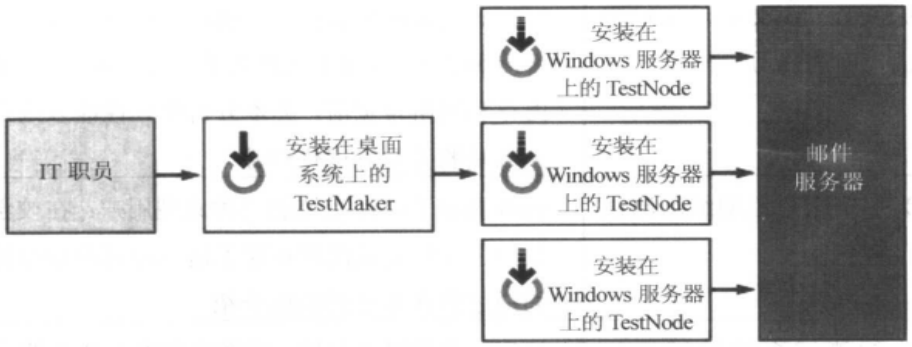

图 15-1　邮件系统测试使用 TestMaker 作为一个中心控制台，以驱动正运行在几个 TestNode 上的测试代理。本测试的 TestNode 为便宜的基于 Intel 的 Windows 服务器，在其上运行 TestNetwork 软件。测试代理模拟虚拟用户来驱动邮件服务器上的负载

在一个分布式测试环境中，测试代理运行在多台测试服务器设备上。每台服务器，称为一个 TestNode，即运行 PushToTest 的 TestNetwork 软件，通过模拟原型用户的行为来产生邮件服务器上的负载。对于邮件服务器来说，来自 TestNode 的请求好似来自真正用户的请求。在本节的后面将介绍关于用户原型设计的细节。

分布式测试环境给大学 IT 职员提供了很大的灵活性。他们可以在实验室设置中运行 TestNode，用于加载测试邮件服务器。他们也可以将 TestNode 分布到他们的网络基础架构中，以得知网络的哪个部分无法正常运行。他们也可以将 TestNode 放置在邮件服务器的不同网络连接上。例如，Saint Leo 大学利用 Time Warner、Verizon 和其他提供商提供的连接，为用户提供到邮件服务器的连接。通过在每个外部网中放置一个 TestNode，该大学可以得知所提供网络之间的延迟差别。

TestMaker 提供了一个控制台，来执行和同步邮件系统的测试。TestMaker 使用客户端软件来控制 TestNode。TestMaker 命令协议使用 HTTP 连接上的 SOAP 消息将指令发送到 TestNode。

这种设计意味着 Saint Leo 大学不必改变它的配置，包括在其防火墙上开放新的端口，就能在该大学网络内部和外部使用分布式的测试环境。

TestMaker 提供了一个图形用户界面，用于启动一个测试、监视测试进程和分析并提供测试结果。Saint Leo 大学在 IT 组成员的台式机上安装 TestMaker，并从那里控制 TestNode。TestMaker 和 TestNetwork 是纯 Java 应用程序，可以运行在 Java 能够运行的任何地方。因此当 IT 职员的任务发生变化时，环境可以很容易地从 Windows 移植到 Linux 或 Mac OS X，或任何能够运行 Java 的地方。

这个测试方法学使用表 15-2 列出的步骤。

表 15-2　邮件系统的测试方法学

步　骤	目　标	行为和度量
1	配置系统，用于一个测试	安装邮件系统，并准备服务于用户。创建 500 个测试用户账号。TestNetwork 软件安装在 TestNode 机器上。TestMaker 软件安装在 IT 职员台式机上
2	标识 TestNode 机器运行测试代理的能力	运行 ramp-up 测试脚本，该脚本在 TestNode 机器上运行增长的智能测试代理。每个代理仿真一个虚拟用户。在本机耗尽 CPU、内存和网络带宽前，脚本分析代理性能以确定用于运行的、最大数量的并发测试代理数
3	为每个用户原型编写测试代理	调查揭示了邮件系统的三种原型用户：在校学生、校外学生和教员。三种测试代理实现了这三种用户原型的行为。用户原型的描述将在本章的后面介绍
4	执行测试系统的试运行 (trial run)	运行一个测试 5 分钟，以确定表中 1~3 步的测试方法学是否正确配置
5	运行测试	相继在邮件系统上运行大量的虚拟用户——500、1 000、1 500、2 000、2 500 和 3 000——跨越所有的 TestNode。运行每个测试 10 分钟
6	运行 Tally 脚本	Tally 脚本分析测试结果日志，确定每秒事务索引和其他统计
7	重复 5 到 6 步三遍	当运行在步骤 5 中的测试的每秒事务索引变化超过 10%时，表明外部施加的压力起作用了，影响了测试结果
8	运行测试	运行测试 1 小时
9	运行 Tally 脚本	对比来自短期和长期测试运行的结果统计
10	搜索测试和系统中的反常	构造一个可能影响测试结果的系统属性列表
11	记录测试和发现的文档	包括测试方法学、测试统计和观察结果

用户原型

智能测试代理技术是一种强大的新方法，用于根据原型用户的目标来测试系统。除了测试

Saint Leo 大学邮件系统的原始功能，测试还建模用户原型，显示了系统性能与用户目标的接近程度。第 2 章提供了关于如何在一个测试的上下文中定义可扩展的、可靠的和性能良好的用户原型的详细讨论。在 Saint Leo 大学邮件系统示例中，调查揭示了 3 种用户原型。根据进一步的调查，PushToTest 能够构造下面的用户原型：

- Marilyn 是一个校外学生，使用邮件系统与大学教员进行通信。Marilyn 是一个本科生，在一个法律办公室做全职职员。Marilyn 使用邮件 Web 接口来检查消息。当新的邮件系统等待时，她立刻阅读并删除该邮件消息。Marilyn 每次给朋友发送邮件消息时都会检查消息。
- Muriel 是一个校内学生，他使用邮件系统和指导教员进行通信。Muriel 是一个硕士生，兼职助教。他使用基于 Windows 的膝上型电脑，其上安装有 Microsoft Outlook Express。Muriel 天生有些急躁。在等待新邮件时，她会立刻下载并阅读该邮件。当从教员处接收到邮件时，Muriel 立即发送回复消息。
- Betsy 是一个享有终身职位的教授，她使用 SLU 邮件来和学生、教员进行通信。Besty 使用 SLU SMTP/POP3 接口来检查邮件。她喜欢以组的方式处理邮件。她通常等待有五个消息后才会阅读邮箱中的消息。Besty 把学生分成 10 个组。她不时地使用 Microsoft Office 附件(Excel、 Word 和 Zip 格式)来给每个组发送消息，以便确保他们能够得到最新的作业。

这些用户原型阐述了 Saint Leo 大学邮件系统中 3 种原型用户的行为。他们覆盖了校内学生、校外学生和很少在学校的仅参加卫星校园会议的学生。另外，Betsy 原型拥有教员的行为，使用邮件系统来安排教学任务，给学生提供任务，与管理员和教员通信。当然，也有空间让更多的用户原型来测试系统，包括 IT 教员的原型。本章的后面将介绍这些用户代理，它们模拟了这些用户原型。

15.4 安装和配置

测试环境的安装需要以下步骤，它们由 Saint Leo 大学的 IT 职员成员执行：

- 在一台桌面计算机系统上安装 TestMaker。TestMaker 使用一个 ZIP 格式的压缩文件。解压缩该文件，创建一个 TestMaker 目录，包含运行智能测试代理所用到的所有文件、脚本、实用程序和库。TestMaker 需要 Java 1.4 版，或更高的版本。
- 在 TestNode 机器上安装 TestNetwork。TestNetwork 使用一个 ZIP 格式的压缩文件。像 TestMaker 一样，TestNetwork 包含操作 TestNode 的所有文件、脚本、实用程序和库。默认情况下，TestNetwork 被配置为一个基于 SOAP 的 Web 服务，需要通过端口 8090 访问它。TetNetwork 安装指令显示了如何配置一个不同的端口号。TestNetwork 需要 Java 1.4 版。
- 将测试代理脚本安装到运行 TestMaker 的机器上。测试代理脚本包括表 15-3 中描述的文件。

表 15-3 测试代理文件

文 件 名	描 述
EmailTestScenario	包含所有将被发送到 TestNode 的文件，用于执行邮件测试，包括下面几个文件： properties.a——配置和设置测试。例如，控制基于用户原型行为的测试代理的相对比例 master.a——一些主要的入口，分别指向安装、运行和记录测试。 archetypes——包含实现用户原型行为的脚本的目录，实现方式是通过使用 TestMaker 的 TOOL 库来驱动使用本地 SMTP、 POP3、IMAP 和 Web 协议的邮件系统
nodes.conf	一个属性文件，定义了测试中要使用的 TestNode
testmail.a	整个测试的主入口。这个脚本执行 TestNode 的测试代理，传递所有需要的参数，并将记录的结果日志数据返回给 TestMaker 机器

- 配置邮件系统，使测试代理具有一组邮件账号。
- 设置 properties.a 文件，以配置想要的测试代理原型的组合、到 POP3 和 SMTP 服务器的 URL、到 Web 主机的 URL 和虚拟用户的数量。

执行完这些安装和配置步骤后，测试就已经准备就绪。在我深入介绍到测试结果之前，本章的 15.4.1 节将介绍如何从用户原型描述转移到测试代理脚本。

15.4.1 从原型到测试代理脚本

用户原型描述了一个系统的原型用户。每个系统都有很多原型，但是每个原型都是某种类型行为的完全描述。例如，我的舅舅 Norman 是 20 世纪 60 年代的文理学士的原型。他是美国人，出生在纽约，后来在罗马成为空军飞行员。他没有孩子，最后在 64 岁结婚，有着非凡的维护世界友谊的天赋。他的生活方式、举止、幽默感和品位代表了 20 世纪 60 年代的文理学士。由于成百上千的人具有相似的背景，在向您描述完他之后，我不必再描述那个时代或社会背景的任何人了。一旦描述了，原型就描述了单个人的惟一行为，但是代表了一组人。

本章早期的调查显示了 3 种用户原型，分别为 Marilyn、 Muriel 和 Betsy。最终我们将增加更多的用户原型，这将增加测试的广度，以包含更多的面向用户目标的测试。对每个新的原型我们会问自己，新的原型真的是一个现有的原型吗？例如，假设我们开发新的原型，名称为 Simon，他是一名大一的学生，住在校园里的宿舍，他通过邮件把学习任务提交给老师。除非有更多的 Simon，否则我就告诉您 Simon 实际上是具有发送邮件能力的 Muriel。相对于创建新的用户原型，我们宁愿扩展 Muriel，以包括 Muriel 将邮件发送给她的老师的行为。用户原型代表了远离模仿行为的原型行为。

从用户原型转移到实现了原型行为的测试代理，需要在 TestMaker 环境中作某些工作。第 4 章和第 5 章提供了智能测试代理的生命周期(从创建一直到撤销)的很好描述。将这些技术应用到 Saint Leo 大学邮件系统测试是相当简单的。

15.4.2 创建 Muriel

在本章的前面我们定义 Muriel 为在校园内使用邮件系统的学生。Muriel 使用邮件系统与导

师进行通信。她是高年级的学生，通过成为兼职助教有能力成为教师。她快速地检查邮箱消息，并在消息到达时立刻下载和响应邮件消息。Muriel 在回答接收到来自教师的消息时，都是特别小心的。

将 Muriel 原型转换为智能测试代理需要完成如下 4 种行为：

- Muriel 使用邮件客户，她使用 SMTP 和 POP3 协议检查新邮件。
- 在等待新邮件时，她立刻下载和阅读邮件。
- Muriel 给她从 SLU 教员处接收到的任何消息发送回复消息。从测试的目标来看，Muriel 考虑从 test1 到 test50 的教员账号中接收的邮件。
- Muriel 发送没有附件的邮件消息，发送间隔为 30 秒~60 秒之间的随机时间间隔。

下面的代码实现了模拟 Muriel 的智能测试代理的基本逻辑。下面给出了测试代理脚本的相关片断，其后有相应的解释。

```
while running:

        status  =  "ok"
        datapulse = Date().time

        try:
            read_emails( agent_num )

            chattercount += 1
            if chattercount > random.randrange( 1,  3  ):
            chattercount = 0
            send_email( agent_hum,  \
               random.randrange( 51,  100 )  )
```

这段代码表明：只要测试正在运行，测试代理将从这个测试代理标识的测试账户处读取邮件消息。测试代理会随机组成并将一个邮件消息发送到其中一个学生。

在代理读取邮件消息时，如果消息来自教员，意味着来自 test1 到 test50，代理就立刻回复。为了实现这个测试代理逻辑，测试代理包含了一个读取邮件消息的功能。下面是它的全部功能，其后是这个代理的移动部分的详细描述。

```
def read_emails( agentnum ):

    global mailhost

    replyaccount = 0
    fromadr  =  ""

    protocol2 = ProtocolHandler.getProtocol("mail")
    pr0t0col2.setHost( mailhost )

    account = "test" +str( agentnum )
    Protocol2.setUsername( account )
    protocol2.setPassword( "passw0rd" )

    response = protocol2.connect()

    response.setPermission( Folder. READ_WRITE )
    response.setFolder(  "INBOX"  )
```

```
        messages = response.getMessages()
        replyaccount = 0

        for i in messages:
            if ( i.getFrom()  != None ):
                fromadr = str(i.getFrom() [0]  )
                replyaccount = 0

                if fromadr.lower().find("test")  == 0:
                    pos = fromadr.lower().find("@")
                    if pos  != -1:
                account = fromadr[ 4  : pos ]

                if int( account )  < 50:
                    replyaccount =int( account)

        i.setFlag( Flags.Flag. DELETED,  1 )

    response.close( 1 )

    if replyaccount > 0:
        send_email( agentnum,  replyaccount )
```

这个测试代理实现了 Muriel 回复来自教员的邮件消息的需求。下面介绍一下这个函数的工作细节。

```
    def read_emails( agentnum ):

        global mailhost

         replyaccount = 0
        fromadr  =  ""
```

首先我们定义这个函数的名称为 read_emails，它有一个简单的参数。agentnum 是代理线程数目的整数值。代理数组成邮件账号的名称，用于检查或发送消息。例如，当 agentnum 等于 8 的，read_emails 就检查 test8@universitymail.com 邮件账号。

```
    protocol2 = ProtocolHandler.getProtocol("mail")
    protocol2.setHost( mailhost )

    account = "test" + str( agentnum )
    protocol2.setUsername( account )
    protocol2.setPassword(  "passwOrd"  )
```

这个测试代理脚本创建 MailProtocol 对象的一个新实例，我们将在 protocol2 变量里引用它。第 8 章提供了关于 MailProtocol 的详细信息。这里的附加命令通知 MailProtocol 对象在哪里找到邮件主机以及要使用哪个账号。

```
    response = protocol2.connect()
```

connect()方法打开到邮件主机的连接、授权测试代理和返回一个 response 对象，该对象用于后续的邮件操作。

```
    response.setPermission( Folder.READ_WRITE )
```

setPermission 方法指示我们测试代理上的邮件主机。默认情况下，Folder.READ_ONLY 通知邮件主机我们将读取邮件消息，但是不从主机上删除它们。Folder.READ_WRITE 通知邮件主机我们期望消息删除命令。

```
response.setFolder( "INBOX" )
```

默认情况下，POP3 协议使用一个名为 INBOX 的文件夹。IMAP 协议支持多个文件夹。

```
messages = response.getMessages()

for i in messages:
    if ( i.getFrom() != None ):
        fromadr = str(i.getFrom() [0] )
```

循环处理每一个消息。getFrom、getSubject 和 writeTo 方法返回消息的发送者(from)、主题(subject)和消息体(message body)。

```
replyaccount = 0

if fromadr.lower().find("test") == 0:
    pos = fromadr.lower().find("@")
    if pos != -1:
        account = fromadr[ 4 : pos ]
```

这个测试代理脚本解码邮件消息的接收者(to)地址的值。测试代理用户账号使用同样的格式，test1@universitymailcom、test2@universitymailcom 等等。这段代码根据邮件发送者的地址发现账号数。

```
if int( account ) <= 50:
    replyaccount =int( account )
    send_email(agentnum, replyaccount)
```

Muriel 立即回复接收到的来自教员的邮件消息。从测试的目标来看，邮件账号小于 test50 的都被认为是来自教员的账号。当邮件消息来自教员时(账号数≤50)，那么 Muriel 就发送回复消息。Send_email 是定义的另外一个函数，带有两个参数，当前代理数目和目标邮件账号数目，用于发送一个邮件消息。

我们在测试中要采用的额外步骤是让 Muriel 读取消息的内容，以验证通信。消息体来自 response 对象的 writeTo 方法，例如，打印 i.writeTo(outputStream)。

```
i.setFlag( Flags.Flag.DELETED, 1 )
```

setFlag 方法支持代理脚本标记邮件消息，邮件主机根据该标记删除该消息。在一个循环中设置这个标记，但直到 close()方法执行时才实际生效。如果 setPermission 方法没有被设置成 Folder.READ_WRITE，则这个标记将不起作用。

```
response.close( 1 )
```

关闭到主机的连接。close 方法有一个简单的 Boolean 参数。当该参数被设置为 true 或者 1 时，close 命令指示邮件主机删除任何已标记的消息。当参数为 false 或者 0 时，close 命令通知邮件主机忽视已删除的消息。

这个测试代理脚本实现了 Muriel 的行为。Muriel 测试代理考虑来自教员的 test1 到 test50 账号的邮件。测试代理回复来自这些账号的消息。测试代理有规律地将消息发送给其他学生。这些账号大于 test51。最后，测试代理通过使用本地 SMTP 和 POP3 协议，模拟使用 Microsoft Outlook Express 工具，像 Outlook 发送和读取邮件消息一样。

下面，我们看看如何创建模拟 Marilyn 的智能测试代理。

15.4.3 创建 Marilyn

如我们在本章前面所看到的，Marilyn 是一名校外学生，她使用邮件系统与大学教员进行通信。Marilyn 使用 Web 浏览器来读取和发送邮件消息。这就给我们提供了使用 TestMaker 的 Recorder 向导来帮助编写测试代理脚本的机会。

使用 Recorder

Recorder 是 TestMaker 中一个有用且方便的实用工具，它监视您使用浏览器，并为您自动编写一个测试代理脚本。Marilyn 使用一个 Web 浏览器来发送和接收邮件消息。我们使用 Recorder 来记录 Marilyn 读取和发送邮件消息的行为。然后，我们将记录的脚本修改为整个测试的一部分。图 15-2 显示了在 TestMaker 中启动 Recorder。

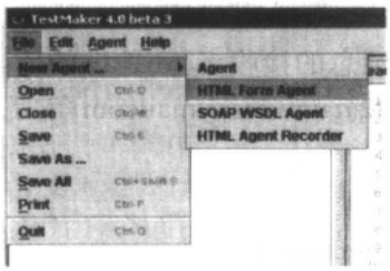

图 15-2 在 TestMaker 中，通过选择 File 下拉菜单中的 New Agent 命令，然后选择 HTML Agent Recorder，来启动 Recorder

激活 Recorder 后，显示一个如图 15-3 所示的对话框。该对话框请求您命名所记录的测试代理脚本。

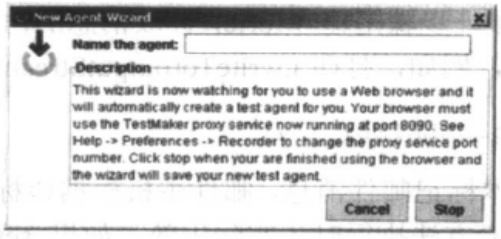

图 15-3 通过询问记录的测试代理的名称来启动 Recorder

TestMaker 将 Recorder 实现为一种特殊类型的代理服务器。您的浏览器通过 Recorder 代理服务器发出请求。该代理将浏览器请求传递给 Web 主机，它将请求记录为一系列的测试代理脚本命令。

Muriel 通过登录到邮件系统，来开始每一个会话。图 15-4 显示了 Web Mail 的登录页面。

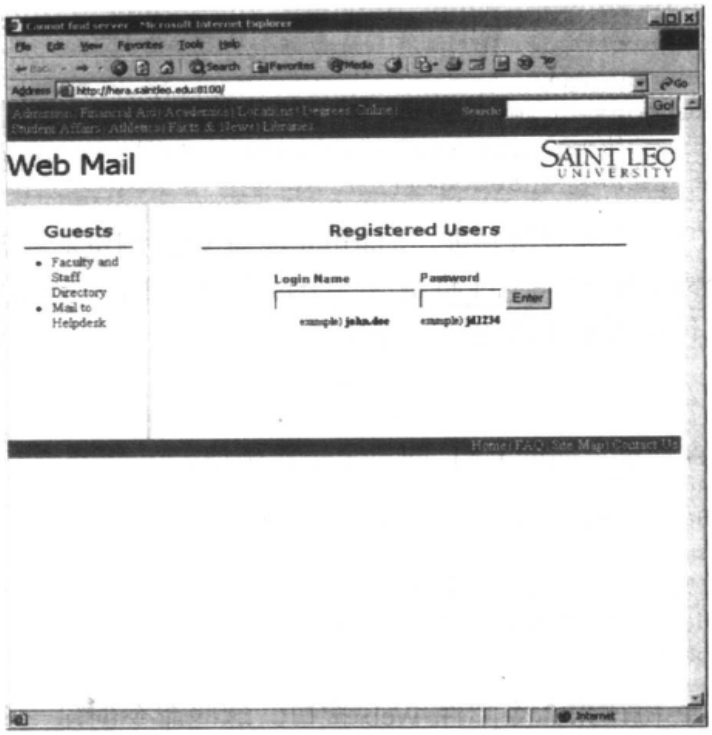

图 15-4　Saint Leo 大学 Web 邮件系统的所有会话开始于一个登录页面。在此用户输入登录名称和密码。所有后续的页面都使用每个页面 URL 中的会话标识符来标识用户

在后台，Recorder 监视登录页面，并编写下面的测试代理脚本。首先我将给出该测试代理脚本的片断，然后会提供详细的解释。

```
loc =  '''http://hera.saintleo.edu:8100/'''

http = ProtocolHandler.getProtocol("http")
http.setUrl( loc )
http.setType( HTTPProtocol.POST )
body = ProtocolHandler.getBody(  'http'  )
http.setBody( body)

body.addParameter('''Username''',  '''test'''  + \
str(agentnum)  )
body.addParameter('''Password''',  '''apassword''')
body.addParameter('''login''',  '''Enter''')

response = http.connect()
if response.getResponseCode()<>301:
    raise Exception,  "Sign-in response code:  "+ \
    str(response.getResponseCode()  )

# Follow redirect to mailbox menu page

loc = response.getParameterValue(  "Location" )
pos = loc.find("Session")
pos2 = loc.find(  "/", pos + 9 )
sessionid = loc[ pos + 8  : pos2 ]
```

```
http.setUrl( loc )
http.setType( HTTPProtocol.GET )
response = http.connect()
if response.getResponseCode()<>200:
    raise Exception,  "Follow response code:  "+ \
    str(response.getResponseCode()  )
```

这个测试代理脚本片断实现登录功能，Marilyn 使用 Web 浏览器执行该功能。只要提供的用户 id 和密码正确，登录页面的 servlet 将响应一个重定向命令，以请求浏览器加载 Hello.wssp 页面。下面是这个测试代理脚本片段的详细解释。

```
loc =  '''http://hera.saintleo.edu:8100/'''
```

新的 loc 变量将包含登录页面的 URL 的一个字符串值。

```
http = ProtocolHandler.getProtocol("http")
http.setUrl( loc )
http.setType( HTTPProtocol.POST )
body = ProtocolHandler.getBody(  'http'  )
http.setBody( body )
```

测试代理使用 HTTP 协议来命令邮件 Web 接口。这些命令创建一个 HTTP 协议处理程序对象，该对象将发出一个到主机的 POST 命令。

```
body.addParameter('''Username''',  '''test'''  + \
str(agentnum)  )
body.addParameter('''Password''',  '''apassword''')
body.addParameter('''login''',  '''Enter''')
```

HTTP 协议体承载 Marilyn 的用户 id 和密码。

```
response = http.connect()
if response.getResponseCode()<>301:
    raise Exception,  "Sign-in response code:  "+ \
    str(response.getResponseCode()  )
```

connect 方法发出到邮件主机的 POST 命令，并创建一个 response 对象。这段脚本执行检查功能，以确保邮件主机将一个 301 重定向命令返回到 Hello.wssp 页面。如果登录功能没有预期实现，则脚本将产生异常并返回。

对于登录页面的响应需要给出具体的说明。响应包括以下 3 种数据：

1. 响应代码——这将成为一个 301 重定向代码。

2. 到下一页面的 URL——到 Hello.wssp 页面或是一个错误页面的 URL。

3. 会话标识符——Saint Leo 大学邮件系统被配置使用 URL 重写来标识一个到主机的用户。每个到 Web 邮箱主机的后续请求必须在请求的 URL 中包含会话标识符。否则，Web Mail 主机将重定向到登录页面的请求。可在浏览器的 cookies 中选择 URL 重写。Web Mail 主机并不是在 cookie 中保存会话标识符，而是在 URL 中保存会话标识符。例如，下面是一个用户登录所使用的 Web Mail URL：

```
http://hera.saintleo.edu:8100/Session/14665y9jgEy1H7IKcrDJxcoAo/Hello.wssp
```

遗憾的是，在重放(playback)期间，Recoder 不支持动态的 URL 重写。如果我们使用记录的脚本，测试代理就会进行登录，然后使用记录的会话标识符。通常系统被配置为在一定时间之后结束一个会话。因此，记录的会话标识符会无效。为了避免这些问题，我向记录的测试代理脚本添加了下面的脚本：

```
loc = response.getParameterValue( "Location" )
pos = loc.find("Session")
pos2 = loc.find( "/", pos + 9 )
sessionid = loci[ pos + 8 : pos2 ]
```

这段代码动态地找到重定向 URL 中的会话标识符。它对到主机的后续请求，存储并使用会话标识符值。

```
http.setUrl( loc )
http.setType( HTTPProtocol.GET )
response = http.connect()
if response.getResponseCode()<>200:
    raise Exception, "Follow response code: "+ \
    str(response.getResponseCode() )
```

测试代理脚本使用同样的 HTTP 协议处理程序对象来获取重定向页面——邮箱菜单页。响应代码为 200 表示成功。在浏览器中，我们发现了图 15-5 中所示的界面。

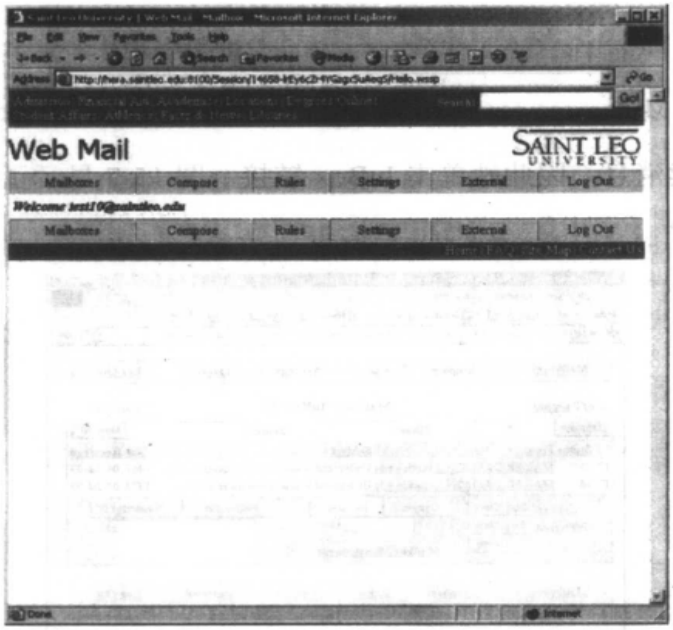

图 15-5　在登录页面之后，Web 邮件界面显示一个页面，该页面让用户选择查看其邮箱、编写新邮件和选择其他选项。此时，Marilyn 正好到邮箱来阅读等待的消息

Marilyn 想查看她的邮箱，因此她单击 Mailboxes 链接。测试代理使用下面的脚本命令来获取 Mailboxes.wssp 页面。

```
loc = "http://" + mailweb + "/Session/" + sessionid\+ "/Mailboxes.wssp"
http.setUrl( loc )
```

```
http.setType( HTTPProtocol.GET )
response = http.connect()
if response.getResponseCode()<>200:
    raise Exception,  "Mailbox summary response code:  "\
    + str(response.getResponseCode()  )
```

单击 Mailboxes 链接，进入 Mailboxes.wssp 页面，如图 15-6 所示。这个页面显示了邮箱的内容。

图 15-6　Web 邮件系统在 Mailboxes.wssp 页面显示了用户邮箱的概况。选择 InBox 链接将显示邮箱的内容

Marilyn 想开始阅读消息，因此她单击 InBox 链接。图 15-7 显示了结果页面，该页面列出了等待邮件消息的链接。

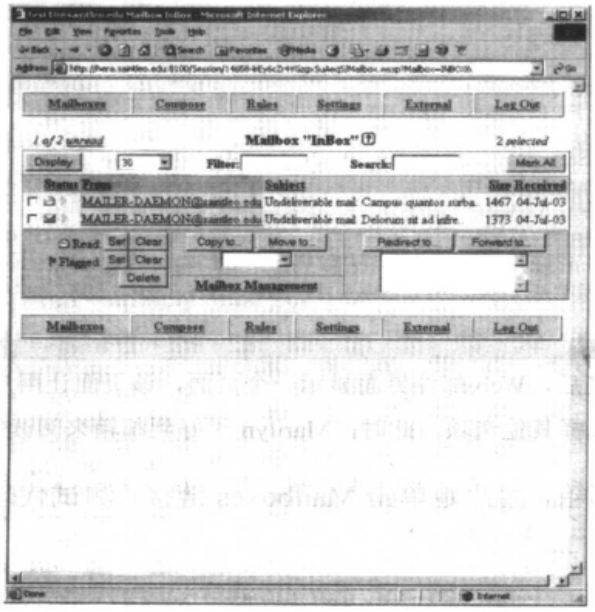

图 15-7　等待邮件消息的链接

执行 Marilyn 下一步动作的测试代理脚本需要做一些额外的工作，以反映这个页面的动态本质。邮件系统显示了一个链接表，这些链接反映了等待邮件消息的列表。测试代理脚本需要分析 html 以发现链接的集合，然后选择一个读取相应邮件消息的链接。首先将为您提供测试代理脚本的片断，然后给出详细的解释。

```
loc = "http://" + mailweb + "/Session/" + sessionid +\
"/Mailbox.wssp?Mailbox=INBOX&"

http.setUrl( loc )
http.setType( HTTPProtocol.GET )
   response = http.connect()
   if response.getResponseCode()<>200:
       raise Exception,  "Email list response code:  "+\
       str(response.getResponseCode()  )

   f1 = "<A HREF=\"Message.wssp?Mailbox=INBOX&MSG="
   f2  =  "\">"

   responselink = ResponseLinkConfig()
   responselink.setParameter(  'beginsearch',  f1 )
   responselink.setParameter(  'endsearch',  f2 )

   search = SimpleSearchLink1()
   search.init( responselink )

   found = search.handle( response )

foundcount = \
found.getParameterValue("simplesearch.foundcount")

   if  ( foundcount == 0 ):
       return

   foundlist = \
   found.getParameterValues("simplesearch.founditems")
   doc = foundlist.get(random.randrange(0,foundcount))

   # Next trim the <a href= and > tags to find the message
number
   msgnum = doc[ len(f1)  :  (len(doc)  - len(f2)  )  ]

   # Get the message
   loc = "http://" + mailweb + "/Session/" + sessionid + "/
Message.wssp?Mailbox=INBOX&MSG=" + msgnum

   http.setUrl( loc )
   http.setType( HTTPProtocol.GET )
   response = http.connect()

   if response.getResponseCode()<>200:
      raise Exception,  "Get message response code:  "+ str(
response.getResponseCode()  )
```

这个测试代理脚本片断加载邮箱概述页面，并随机选择一个等待的邮件消息。下面是这个测试代理脚本片段的详细解释：

```
loc = "http://" + mailweb + "/Session/" + sessionid +\
"/Mailbox.wssp?Mailbox=INBOX&"

http.setUrl( loc )
http.setType( HTTPProtocol.GET )
response = http.connect()
if response.getResponseCode()<>200:
    raise Exception, "Email list response code: "+\
    str( response.getResponseCode() )
```

这个测试代理脚本片断为这个用户获取 Mailbox.wssp 页面的内容。response 变量指向一个包含 Mailbox.wssp 页面的 html 内容的对象。这个页面包含到用户邮箱中的邮件消息的链接。每条消息是到 Message.wssp 页面一个链接，所带有的参数指示消息的编号。例如，下面的 URL 用于编号为 11 的消息。

```
http://hera.saintleo.edu:8100/Session/15091-
hoPMIzFZBueeyLo2zwSy/Message.wssp?Mailbox=INBOX&MSG=111
```

我们可以使用 TestMaker 的分析功能来找到消息编号。

```
f1 = "<A HREF=\"Message.wssp?Mailbox=INBOX&MSG="
f2 = "\">"
```

f1 和 f2 变量定义了搜索参数。f1 是搜索段的开始，f2 是搜索段的终止。

```
responselink = ResponseLinkConfig()
responselink.setParameter( 'beginsearch', f1 )
responselink.setParameter( 'endsearch', f2 )
```

responselink 变量将指向 ResponseLinkConfig 对象，该对象将分析 Mailbox.wssp 页面的 URL，该 URL 以 f1 的内容开始，以 f2 的内容结束。ResponseLinkConfig 是 TestMaker 的工具对象库提供的一个对象。

```
search = SimpleSearchLink()
search.init( responselink )
```

通过使用多个 ResponseLinkConfig 对象，设计允许您一次搜索多个段。对于这个测试，仅需要一段。在我们向主机发出 GET 请求时，这些命令将 ResponseLinkConfig 对象连接到所创建的 response 对象。

```
found = search.handle( response )
```

对 response 对象应用 search.handle 方法，为搜索的字符串分析 html。found 变量现在包含了一个匹配 ResponseLinkConfig 参数的 URL 列表。换句话说，也就是 Mailbox.wssp 页面中的消息链接的一个列表。

```
foundcount = \
found.getParameterValue("simplesearch.foundcount")

   if  ( foundcount == 0 ):
        return
```

found 变量指向一个搜索对象。在其他方法中，它提供了 simplesearch.foundcount 方法来通知我们发现了多少消息链接。如果邮箱中没有邮件消息，测试代理会等待一段时间，然后再重复这个函数。

```
foundlist = \
found.getParameterValues("simplesearch.founditems")
doc = foundlist.get(random.randrange(0,foundcount)  )
```

测试代理脚本随机地从找到的值列表中选取一个消息链接。doc 变量现在包含所选择的邮件消息的 URL。该 URL 看起来如下所示：

```
http://hera.saintleo.edu:8100/Session/15091hoPMIzFZBueeyLo2zwSy/Message.wssp?Ma
ilbox=INBOX&MSG=111
```

由于会话标识符和消息标识符包含动态的值，测试需要动态创建 URL 来获取邮件消息的 Web 页。

```
msgnum = doc[ len(f1)  :  (len(doc)  - len(f2)  )  ]
```

这个脚本使用一些 Python 技巧来查找所选择的链接中的消息编号。这个命令使用 Python 的字符串操作功能来整理<a href = and>标记，以找到消息编号。

通过发出 HTTP GET 命令到 Web 邮件主机，测试代理脚本现在准备获取邮件消息。

```
loc = "http://" + mailweb + "/Session/" + sessionid+\
"/Message.wssp?Mailbox=INBOX&MSG=" + msgnum

http.setUrl( loc )
http.setType( HTTPProtocol.GET )
response = http.connect()

if response.getResponseCode()<>200:
    raise Exception,  "Get message response code:  "+\
    str(response.getResponseCode()  )
```

上面的测试代理片断请求包含所选择的邮件消息的 Web 页。图 15-8 显示了这个命令的浏览器视图。

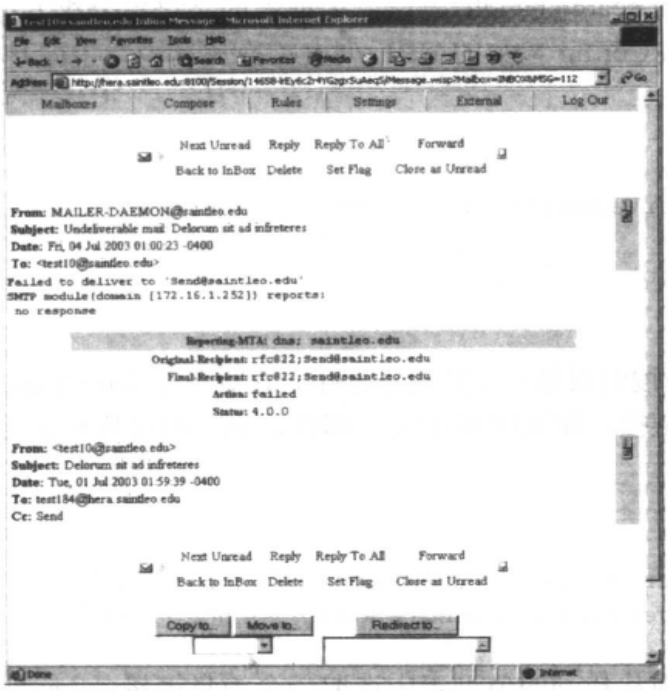

图 15-8 HTTP GET 命令的浏览器视图

当等待新邮件时，Marilyn 立刻读邮件并删除该邮件消息。测试代理脚本的下一步是删除我们刚获取的消息。

```
loc = "http://" + mailweb + "/Session/" + sessionid +\
"/Mailbox.wssp?Mailbox=INBOX&MSG=" + msgnum +\
"&Delete=&"

http.setUrl( loc )
http.setType( HTTPProtocol.GET )
response = http.connect()
if response.getResponseCode()<>200:
    raise Exception, "Delete message response code:"+ \
    str(response.getResponseCode()  )
```

这个测试代理脚本片断发出到 Mailbox.wssp URL 的 HTTP GET 请求，并在其中一个参数中使用删除命令。该脚本动态地创建 URL，以包含会话标识符和消息编号。Recorder 自动地添加脚本命令，以检查请求的响应代码。

创建 Marilyn 代理的最后一步是实现她的行为，即编写邮件消息给其他的学生。Recorder 监视我们在浏览器中单击了 Compose 按钮。这个动作为我们显示了一个 HTML 表单，用于输入一个新消息。

```
loc = "http://" + mailweb + "/Session/" + sessionid +\
    "/Compose.wssp"

    http.setUrl( loc )
    http.setType( HTTPProtocol.GET )
    response = http.connect()
```

```
    if response.getResponseCode()<>200:
        raise Exception,  "Send message response code:" +\
        str(response.getResponseCode()  )
```

这个命令通过包含会话标识符动态地创建了 URL，并获取消息组合页面 Compose.wssp。接着测试代理组合并发送一个新消息。代码片断如下所示，后面有详细的解释。

```
http.setType( HTTPProtocol.POST )
body = ProtocolHandler.getBody(  'http'  )
http.setBody( body )

body.addParameter('''FormCharset''',  '''ISO-8859-1''')
body.addParameter('''Send''',  ''''Send''')
body.addParameter('''Cc''',  '''Send''')
body.addParameter('''Bcc''',  '''Send''')
body.addParameter('''DSN''',  '''Send''')
body.addParameter('''MDN''',  '''Send''')
body.addParameter('''OpenBook''', '''Send''')
body.addParameter('''desiredCharset''',  \
 '''ISO-8859-1''')
body.addParameter('''filled''',  '''1''')
body.addParameter('''To''',  '''test'''  + \
str( random.randrange( 50,  300 ) ) + \
'''@hera.saintleo.edu''')

myLingo = Lingo()
body.addParameter('''Subject''' \
myLingo.getSubject( 4 )  )
body.addParameter('''Body''', myLingo.getMessage())

response = http.connect( 0 )

if response.getResponseCode()<>200:
    raise Exception,  "Post response code: " + \
    str(response.getResponseCode()  )
```

这段测试代理代码片断发送一个 HTTP POST 命令到 Compose.wssp 服务器。Recorder 监视我们接收了 HTML 表单，输入了消息的接收者地址、主题和消息体，然后单击 Send。Recorder 自动编写这个测试代理脚本。我们必须更改 Recorder 脚本，以支持为测试代理账户操作邮件系统和选择一个接收者所需要的动态 URL。

```
http.setType( HTTPProtocol. POST)
body = ProtocolHandler.getBody(  'http'  )
http.setBody( body )
```

首先，脚本设置现有的 HTTPProtocol 对象，向主机发出一个 POST 命令。然后它创建一个新消息体，该消息体将包含消息的参数。

```
body addParameter('''FormCharset''',  '''ISO-8859-1''')
```

```
body addParameter('''Send''', '''Send''')
body addParameter('''Cc''', '''Send''')
body addParameter('''Bcc''', '''Send''')
body addParameter('''DSN''', '''Send''')
body addParameter('''MDN''', '''Send''')
body addParameter('''OpenBook''', '''Send''')
body addParameter('''desiredCharset''', \
'''ISO-8859-1''')
body.addParameter('''filled''', '''1''')

body.addParameter('''To''', '''test''' + \
str( random.randrange( 50, 300 ) ) + \
'''@hera.saintleo.edu''')
```

我更改了 POST 命令的 Recorder 版本，以动态地寻址其中一个测试账号的邮件消息。对于这个测试的目标，Marilyn 希望教员使用的邮件账号的名称在 test1 到 test49 之间。因此这个测试代理脚本随机选择一个从 50 到 300 之间的邮件账号。

```
myLingo = Lingo()
body.addParameter('''Subject''', \
myLingo.getSubject( 4 ) )
body.addParameter('''Body''', myLingo.getMessage() )
```

Recorder 脚本的另一个变化是动态地创建消息主题和消息体。对于此，我使用 TestMaker 工具库中提供的 Lingo 对象。Lingo 返回类似拉丁单词的乱语单词。使用 Lingo 减少了缓存管理器影响测试结果的风险，因为在这个测试中每个消息都包含惟一的内容。

```
response = http.connect( 0 )

if response.getResponseCode()<>200:
    raise Exception, "Post response code: "+ \
    str(response.getResponseCode() )
```

最后，connect 方法发出 POST 命令到 Web 邮件主机，这样一切工作完成。Marilyn 然后回去检查更多的邮件消息。

我们刚刚介绍的这个测试代理脚本实现了 Marilyn 的行为。我们使用 Recorder 创建一个测试代理脚本，该脚本使用了 Saint Leo 大学的 Web Mail 接口。Marilyn 测试代理使用浏览器接口检查邮件并发送消息。她立刻阅读和删除她的邮件账号中的所有邮件消息。在她每次检查消息时，她至少要发送一条邮件消息给她的一个朋友。

15.5 结果分析

Saint Leo 大学邮件测试环境使用分布式的测试机器集，以产生邮件系统的负载和它的支持基础架构。每个测试代理脚本并发运行多次，产生数目不断增长的虚拟用户。每个测试通过分析记录的结果数据来确定邮件系统的吞吐量。图 15-9 显示了这个工作的过程。

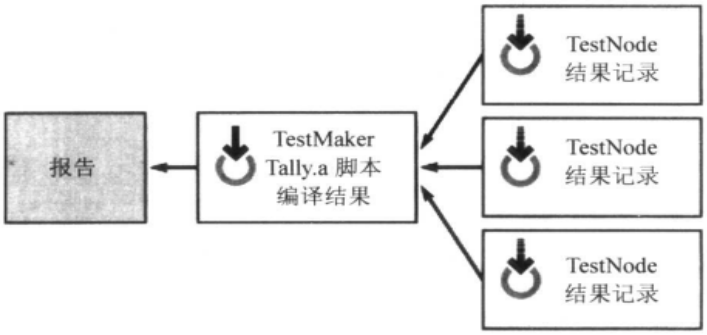

图 15-9　在测试运行时，测试代理脚本将结果数据记录到本地 TestNode 机器上的日志文件中。总结测试时，TestNode 将所记录的数据返回到中心 TestMaker 控制台，用于分析和报告

这些测试代理脚本创建了一个简单的由逗号分隔的数据文件，包含下列域：

- 完成事务的时间
- 当前系统时间
- 测试代理原型
- TestNode 机器标识符
- 事务类型

一旦结果日志被返回到 TestMaker 机器，tally.a 测试脚本就分析每个日志文件，并编译一个统计结果集。

- 每秒平均事务数
- 一个事务持续的最短时间
- 一个事务持续的最长时间
- 每个用户原型完成的事务数
- 错误条件下结束的事务数
- 运行测试的总时间

在每个测试结束时，tally.a 脚本将自动运行。因此测试可以是完全自动化的。

15.6　本章小结

在本章中，通过实现邮件系统功能的一个面向用户目标的测试来满足用户目标，满足了一个重要在线大学的需求。本章讨论了 Saint Leo 大学进行一个邮件环境的多协议测试的需求、测试工具和方法学的选择，详细描述了测试所使用的测试代理代码，并介绍了测试如何将有价值的信息提供给 Saint Leo 大学，以节省时间、精力和资源，同时能够很好地服务它的用户。