重庆邮电大学

硕士学位论文

姓名：王雷雷

导师：杜 江

专业：计算机应用技术

二零一一年五月

分类号 TP393.09 密级 公开

重庆邮电大学硕士学位论文

论文题目 Web 应用性能测试系统的设计与实现

英文题目 Design and Implementation of Web Application Oriented Performance Testing System

硕士研究生 王雷雷

指导教师 杜 江 副教授

学科专业 计算机应用技术

论文提交日期 2011 年 4 月 论文答辩日期 2011 年 5 月 28 日

论文评阅人

答辩委员会主席 曹龙汉 教授 重庆通信学院

2011 年 5 月 28 日

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 重庆邮电大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：王雷雷　　签字日期：2011 年 5 月 27 日

学位论文版权使用授权书

本学位论文作者完全了解 重庆邮电大学 有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 重庆邮电大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：王雷雷　　导师签名：

签字日期：2011 年 5 月 27 日　　签字日期：2011 年 5 月 27 日

摘 要

目前， Web 应用已经深入到人们的日常生活当中，然而现有很多 Web 应用系统的设计开发人员仅仅追求界面的漂亮和功能的正确，很少在系统上线前进行严格的性能测试，导致系统在上线后，一旦访问量过大，造成不可弥补的损失。

本文主要对 Web 应用中的 HTTP 协议以及现有的性能测试方法和测试工具进行了深入的研究和分析，建立了一种联合性能测试方法。该方法结合了应用在客户端的性能测试以及应用在服务器端的性能监控，并依据该方法设计并实现了一个简洁，灵活，易用的性能测试系统。

该系统主要包含了脚本录制功能，测试执行功能，性能监控功能，数据分析功能。脚本录制功能主要由代理服务器和解析器组成，实现了对 HTTP 请求消息的捕获和解析，将这些请求消息保存在 XML 文件中，形成了测试脚本。测试执行功能主要由管理端和测试端组成，管理端可以灵活的制定测试计划和监控策略，传输测试计划和测试脚本，以及启动测试端的测试执行和性能监控功能，测试端主要是负责测试执行以及产生原始测试数据。性能监控功能主要是依据 Web Service 技术灵活地实现了后台监控。数据分析功能主要是利用数据库中的原始测试数据产生被测系统的性能参数，供测试人员定位系统的性能瓶颈。

最后，我们采用该测试系统对重庆爱思网安信息技术有限责任公司的 OA 办公系统进行了性能测试，得到该测试系统在不同并发用户数下的性能参数和监控日志，验证了该测试系统的正确性和有效性。

关键词：性能测试，性能监控，协议分析

Abstract

At present, Web applications have penetrated into people's daily life. However, many existing Web application design and development staff pursue only beautiful interface and correct function, very few lines in the system before the rigorous performance testing, resulting in irreparable damage, once the access is too large.

This paper focuses on the research and analysis of the HTTP protocol in the Web application and existing performance testing methods and tools, establishing a joint performance testing method. The method combines performance testing on the client side and performance monitoring on the server side. We design and implement a simple, flexible, easy-to-use performance testing system.

The system mainly consists of script recording function, testing execution function, performance monitoring function, data analysis function. The script recording function mainly composed of proxy server and resolver, achieving the capture and analysis of HTTP request message and formation of a testing script. The testing execution function mainly composed of management side and testing side. The management side implements flexibility developing test plan and monitoring strategies, transporting the test plan and test script to testing side, starting the testing side and performance monitoring function. The testing client is responsible for testing execution and generating the raw testing data. Performance monitoring is mainly based on Web Service technology to achieve the background monitoring flexibility. The main function of data analysis is to statistics and analysis of raw testing data in the database, then generating the performance parameters.

At last, we adopt the performance testing system to test an OA office system, getting the performance parameters and monitoring log with different number of concurrent users, verifying the correctness and validity of testing system.

Key words: Performance testing, Performance monitoring, Protocol analysis

目录

第一章 绪 论

1.1 选题的背景

伴随着 Internet 的飞速发展，各种各样的网络应用已经呈现在人们的面前，如 Web 应用，VOIP，P2P 等等。任何一种应用如果想要真正为人们所用，必须经过严格的测试，这些测试包括功能测试，性能测试，安全性测试等。然而在这些测试中，功能测试保障了系统的正确性，性能测试保障了系统的稳定性。由此可见，性能测试保证了系统的健壮性，对系统的稳定起到了决定性的作用。

考虑到网络应用门类众多，而且自万维网产生以来，Web 应用已经得到了广泛的发展，特别是这些年，已经在全球范围内形成了一个巨大的信息网络，人们的日常生活和 Web 息息相关，人们用 Web 来浏览新闻，网上购物，收发邮件，网上办公等等[1]。所以说，Web 已经成为人们生活和工作中必不可少的一个部分。所以本文，主要研究 Web 应用的性能测试。

目前，越来越多的信息系统已经移植到互联网上，现在的 Web 系统已经变得越来越庞大和复杂。然而许多 Web 应用的开发人员为了吸引用户的眼球，以及迫于系统短时间发布的压力，只注重对页面效果的设计和功能的正确性，而忽视了应用系统的性能[1]。导致用户在使用过程中感觉访问速度慢，响应时间长，甚至出现系统崩溃的情况。这些都不可避免的造成了公司的损失。如何使用 Web 工程理论开发出高质量，高效率的 Web 应用系统成为一个重要的话题[2]。

Yogesh Deshpande 和 Steve Hansen 在 1998 年就提出了 Web 工程的概念。Web 工程作为一门学科，提倡使用一个过程和系统的方法来开发高质量的基于 Web 的系统。在 Web 工程中，基于 Web 系统的测试，确认和验收是一项非常重要和极富挑战性的工作。Web 测试由功能测试，性能测试，一致性测试，客户端兼容性测试和安全性测试组成[3]。然而，性能测试是 Web 系统测试的重要组成部分，保证了 Web 系统的质量和效率。

国内外的研究机构已经对 Web 应用系统的测试进行了深入的研究，也已经取得了一些成果。国外在应用系统性能测试软件研发方面比较深入。主要以美国和以色列的软件公司为领先，如 Mercury Interactive 公司，Compuware 公司等，并且已经生产出成熟的商用软件，如 LoadRunner 和 QALoad 等。这些性能测试工具支持各种应用协议和多种体系结构测试，而且性能稳定，测试效果突出。但是价格却十分昂贵。

QALoad 是企业范围的负载测试工具，支持范围广，测试内容多，QALoad 的

测试脚本开发是由捕获会话，转换捕获会话到脚本，以及修改和编译脚本一系列的过程组成。一旦脚本编译通过后，使用 QALoad 组织分配把脚本分配到测试环境相应的机器上，驱动多个 play agent 模拟大量用户的并发操作，实施应用的负载测试。

LoadRunner，是一种预测系统行为和性能的工业标准级负载测试工具。使用有限的的硬件资源，通过模拟上千万用户实施并发负载来确认和查找问题。它能够对整个企业架构进行测试，支持广泛的协议，它可以应用于各种系统的性能测试，具有使用最少的硬件资源产生重复并可度量的负载的特点[4]。

国外的许多开源的Web应用性能测试软件也很强大，比如Jmeter，OpenSTA，但是由于其使用起来非常复杂，我们在使用时普遍存在学习成本问题，在使用前需要花费大量的时间和精力来学习和研究测试软件的使用方法，如何配置参数等等。比如OpenSTA是一个功能强大，自定义设置功能完备的软件，但是这些设置大部分需要通过Script来完成，因此在真正使用这个软件之前，必须首先学习好它的脚本编写。国内也积极开展对Web应用系统进行性能测试的研究与开发，但是真正能用的工具还不是很多。因此对Web应用的性能测试进行研究与开发是当务之急。

1.2 选题的意义

当一个 Web 系统完成之后，我们如何知道该系统是否确实可用？当大量用户并发访问该系统时，响应时间是否过长，用户是否满意？如果出现系统崩溃或者响应时间过长等性能问题，我们该如何解决？这些问题是每个 Web 系统在使用过程中都可能面临的问题[5]。

由于国内并没有成熟的性能测试工具，我们只能购买国外强大的性能测试工具，增加了我们的成本，然而国外也有一些开源的性能测试软件，可是使用起来比较复杂。主要问题在于开源软件过分追求功能的强大，Web 性能测试只是其中一部分，使得我们在进行 Web 系统性能测试时，需要反复的配置参数，非常复杂，而且一旦参数配置出错，无法进行正确的性能测试，使得学习成本过高。这种软件必须是专业的测试人员才能完成性能测试的工作。

本文在研究了国内外一些现有的性能测试工具后，针对目前的 Web 性能测试的需求，在充分分析现有的性能测试方法和 HTTP 协议的基础之上，建立了一种联合性能测试方法，该方法强调在执行客户端并发性能测试的过程中，同时进行服务器端的性能监控，以便对系统的性能做出全面的评估。并依据该方法设计并实现了一款灵活，简洁，易用的性能测试工具。

1.3 Web 性能测试的重要性以及研究现状

1.3.1 Web 性能测试的重要性

互联网应用的爆炸式增长为程序的快速应用提供了一个很好的发展方式，当前，许多企业发现，依靠 Web 应用来增加他们的收益是一个非常不错的方法。但是如果最终用户认为您的 Web 站点性能不好，那么他们的下一次点击很有可能就会转到您的竞争对手那里去，所以说 Web 系统的测试已经成为系统设计者和开发人员所关注的焦点。

根据 Web 工程的理论，一个完整的 Web 系统的测试包括以下几项：功能测试，性能测试，可用性测试，兼容性测试，安全性测试[6]。只有通过了这些测试，才能成为一个合格 Web 系统，否则将要进行重新设计。如图 1.1 所示：

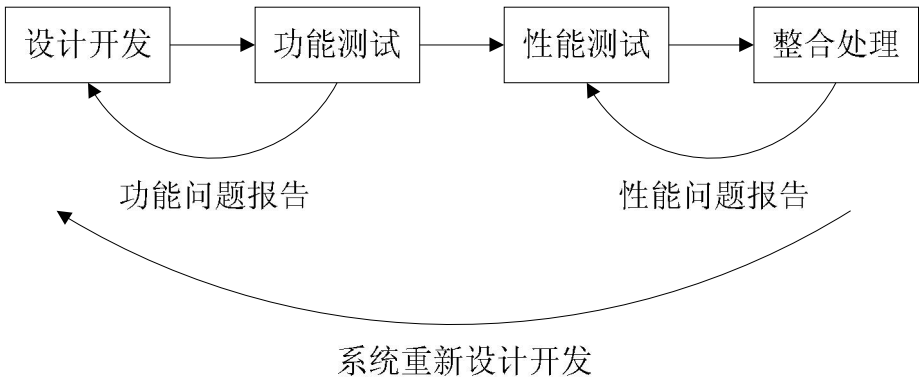

图 1.1 Web 系统开发流程

对于大多数的企业或者成熟度较高的组织来说，单元测试，功能测试和兼容性测试是很容易理解的，然而性能测试却往往被忽略，他们不能体会到性能测试的重要性，很少对即将发布的系统进行实际用户的负载测试以及压力测试，而是自己想当然地预测系统能够承受的压力，缺乏科学的测试方法和数据分析，导致系统在发布以后，当出现大量用户访问时，出现无法响应用户的请求，甚至出现系统崩溃，造成不可弥补的经济损失。

据 Mercury Interactive 公司的研究报告，百分之九十八的 Web 服务器都没有达到人们所期望的性能，平均只能发挥 1/6 的性能[7]。

为什么开发人员不愿意关注性能测试呢？因为性能测试并不像功能问题那样棘手，因为几乎任何常见的性能问题都可以通过硬件来解决，也就是花点钱买个更加强劲的硬件来提高软件的效率，其次通过性能测试后发现了性能瓶颈(一般性能瓶颈都是较为底层的问题)，修复的成本和风险也是需要考虑的问题。

1.3.2 Web 性能测试的研究现状

国外对性能测试的研究已经取得了许多成果，提出了一些性能测试方法，开发了相应的性能测试工具，主要如下。

1) 文[8]提出了使用 PePPeR(Perception Projection Performance Reliability)模型来设计性能测试。该模型如图 1.2 所示，一个 Web 站点是否成功最终是由用户来决定的，用户所感知到的性能是很重要的[8]，然而用户的性能需求是很复杂而且很难满足的，用户和设计人员对性能很难产生共同的标准。在这种情况下，我们需要分析用户的性能需求，得到可以量化的性能指标，建立性能测试模型进行性能测试，保证网站的可靠性。这种方法是以用户行为为中心来设计性能测试，这样可以增强用户对网站的信心。

图 1.2 Flow from Perception to Reliability

2) 文[9]提出了一种 Web 系统性能模型，主要是通过 ULTRA(Universal Load Testing Reply Agent)，产生大量的虚拟用户来测试 Web 在可变的负载下的性能。传统方法是利用专门的机器来生成虚拟负载的，而该方法主要是使用网络上分布的代理来增加虚拟用户的数目，每一个代理可以模拟大量的用户和会话，使用主机的 CPU 空闲时间来生成负载。ULTRA 结构是分布式的，利用分布式计算来处理 Web 负载的动态性和不可预测性而不需要专门的硬件来产生负载，尽可能的模拟真实情况，并且降低了费用[9]。

3) 商业化的性能测试工具主要来自国外，例如 HP 的 LoadRunner，Compuware 的 QALoad 等等。无可厚非，LoadRunner 是目前功能最为强大的性能测试软件，当然它的价格也很昂贵。

LoadRunner 是一种预测系统行为和性能的负载测试工具。通过模拟成千上万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试[10][11]。通过使用 LoadRunner，企业能够最大限度的缩短测试时间，优化性能和加速应用系统的发布。LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。Loadrunner 的测试对象是整个企业的系统。此外 LoadRunner 能支持广泛的协议和技术，为具体环境提供特殊的解决方案。该工具有四部分组成。

A. 脚本生成器 (Virtual User Generator)

VuGen 提供了基于录制的可视化图形开发环境，可以方便简洁地生成用于负载的测试脚本[11]。

B. 压力调度和监控系统 (Controller)

负责对整个负载的过程进行设置，指定负载的方式和周期，同时提供系统监控的功能。

C. 压力生成器 (Load Generator)

负责将 VuGen 脚本复制成大量虚拟用户对系统生成负载

D. 结果分析工具 (Analysis)

通过 Analysis 我们可以对负载测试后生成的数据进行整理分析，得出性能瓶颈，进行调优。

4) 开源性能测试软件，使用较多的有 Jmeter 和 OpenSTA。

Apache Jmeter 是一个百分之百的纯桌面应用，用于负载测试和性能测量。它最初被设计用于 Web 应用的性能测试，但后来被扩展到其他测试领域，如 FTP，SOAP/XML-RPC，JDBC 等[12]。

Apache Jmeter 可以用于对静态和动态的资源(文件，Servlet，Perl 脚本，Web 服务器，java 对象等等)的性能进行测试，甚至可以支持分布式测试场景。它可以用于对服务器，网络或对象模拟繁重的负载来测试它们的强度或分析不同压力类型下的整体性能。

OpenSTA 是一个使用 C++开发的性能测试软件，遵循 GPL 许可证。它是一个带有场景创建，监视并提供分布式测试功能的性能测试软件。然而，使用之前必须学习他自身的脚本语言，否则无法进行性能测试，增加了学习成本。

国内在 Web 性能测试的研究和开发虽然刚刚起步，但是也取得了一些成果。文[13]中根据 Web 应用的特点，将 Web 应用性能测试的内容分为应用在客户端的性能测试，应用在网络的性能测试，应用在服务器端的性能测试三个部分，提出了三层结构的性能测试模型，总结了性能测试的指标，主要对性能测试进行了理论的研究。文[14]开发了一个新的 Web 性能测试工具 WebMark，采用事件驱动的方法管理异步 I/O，通过修改 Client 的 TCP/IP 协议栈来模拟 Internet 环境，采用可靠的方法，模拟真实负载。文[15]提出了一种应用 TTCN(Tree Tabular Combine Notion)形式化描述的 WAE(Web Application Environment)性能测试方法，实现了一个基于 TTCN 的 WAE 性能测试模型。

梁晟，李明树等提出了一种模拟驱动的 Web 应用程序性能预测方法，该方法根据系统使用方式和客户端各种特征的分布信息来确定测试负载，设计测试用例，利用测试工具开发相应的测试脚本，运行测试用例模拟用户行为，收集性能数据。并对 Web 应用系统的响应时间性能指标进行了深入的实验研究，发现 Web 事务的响应时间和并发用户数呈线性关系，测试时间和测试配置对响应时间也有一定的影响[16][17]。

马琳等探讨了一种基于转导推理的性能预测算法[18]，目的主要是解决在历史测试数据有限的情况下，不再基于回归分析的方法先根据历史数据得出一个函数，再计算感兴趣的值，而是直接通过历史数据，利用转导推理计算出感兴趣的值。

1.4 研究的主要内容

本文的目的是致力于建立一个灵活实用的 Web 性能测试方法，满足现有的 Web 性能测试需求。为了实现该目的，本文主要做了如下研究。

首先，通过对 Client 端和 Server 端使用 HTTP 协议的分析，特别是对数据的消息和头域部分做了详细的分析，得出测试脚本的数据结构。

第二，深入分析现有的性能测试工具，得出它们的优点和不足，为新的性能测试系统提供技术支持。

第三，通过对现有测试方法的研究，我们建立了一种联合性能测试方法，强调在客户端进行性能测试时，同时进行服务器端的性能监控，为测试人员更加全面的了解被测系统的性能提供了帮助。

第四，依据对现有性能指标的分析和性能测试的实践，提出了一个名为容量区间的新的性能指标，该指标可以更加合理，可靠的预测被测系统的性能，并且我们讨论了如何获得该指标的具体算法。

第五，依据联合性能测试方法，设计并实现了一套简洁，灵活，易用的 Web 性能测试系统，简称为 WebPT 测试系统。

第六，通过对重庆爱思网安信息技术有限责任公司的 OA 办公系统进行性能测试，得到该 Web 系统的各项性能参数数据，如平均响应时间，系统吞吐量，最大响应时间，容量区间。测试人员可以通过对这些参数数据的分析，同时结合服务器端的监控日志，得出该办公系统的性能瓶颈以及优化建议。

1.5 全文的组织结构

第一章 引言，主要介绍了 Web 性能测试的背景和意义，Web 性能测试的研究现状，以及本文的主要研究内容。

第二章 Web 系统性能测试研究，主要介绍了性能测试的概念，Web 性能测试的指标，以及 Web 性能测试原理，讨论了 Web 应用性能测试的方法。

第三章 协议分析和测试方法的建立，介绍了 HTTP 协议的重要术语，分析了 HTTP 协议的消息，为测试脚本的组成奠定了基础，建立了一种联合性能测试方法，为下一章测试系统的设计提供了理论依据。

第四章 WebPT 测试系统的设计与实现，介绍了该系统的主要功能，以及各个

模块的设计与实现，最后介绍了该系统所用到的主要技术。

第五章 WebPT 测试系统的运行和分析， 采用该测试软件对一个真实的 Web 系统进行不同并发用户数的性能测试，得出被测系统的性能参数，为测试人员定位系统的性能瓶颈提供帮助，同时验证了该测试软件的正确性和有效性。

第六章 总结和未来工作，主要描述了本文的主要工作，以及下一步的主要工作。

第二章 Web 应用的性能测试研究

性能测试是一种信息的收集和分析过程，过程中收集的数据用来预测怎样的负载水平将耗尽系统资源[19]。性能测试保证应用系统拥有良好的性能，它考察在不同的用户负载下，Web 对用户请求做出的响应情况，以确保将来系统运行的安全性，可靠性和效率。

2.1 Web 性能测试内容

Web 性能测试的内容很多，从理解方便的角度来讲，这里把性能测试分为以下几种。

负载测试(Load Testing)

负载测试[20]是指在一定的软件，硬件及网络环境下，运行一种或多种业务，在不同虚拟用户数量的情况下，测试服务器的性能指标是否在用户的要求范围内，以此确定系统所能承载的最大并发用户数，以及不同用户数下的系统响应时间及服务器的资源利用率。

负载测试强调的是在一定的环境下系统能够达到的峰值指标，大多数的性能测试都是负载测试。

压力测试(Stress Testing)

压力测试是指在一定的软件，硬件及网络环境下，模拟大量的虚拟用户向服务器产生负载，使服务器的资源处于极限状态下并长时间运行，以测试服务器在高负载情况下是否能稳定工作[20][21]。与负载测试获得峰值数据不同，压力测试强调在极端情况下系统的稳定性，这个时候处理能力已经不重要了。

例如在 CPU 超频后需要对系统的稳定性进行测试，可以使用工具软件让系统的所有资源长时间处于消耗殆尽的状态，通过这样一段时间烤机后，如果没有出现死机现象，可以认为系统通过了压力测试。

容量测试(Volume Testing)

容量测试[22]是指在一定的软件，硬件及网络环境下，在数据库库中构造不同数量级的数据记录，运行一种或多种业务在一定虚拟用户数量的情况下，获取不同数量级别的服务器性能指标，以确定数据库的最佳容量和最大容量。容量测试不仅可以对数据库进行，还可以对硬件处理能力，各种服务器的连接能力等进行，以此来测试系统在不同容量级别下是否达到指定的性能。

容量测试和负载测试的区别在于，容量测试主要关心 how much，而负载测试同时强调 how much 和 how fast。

配置测试(Configuration Testing)

配置测试[22][23]是指的不同的软件，硬件以及网络环境配置下，运行一种或多种业务，在一定的虚拟用户数量情况下，获得不同配置的性能指标，用于选择最佳的设备及参数配置，通过产生不同的配置，来得到系统性能的变化状况

基准测试(Benchmark Testing)

基准测试[24]是指在一定的软件，硬件及网络环境下，模拟一定数量的虚拟用户运行一种或多种业务，将测试结果作为基线数据，在系统调优或系统评测的过程中，通过运行相同的业务场景比较测试结果，确定调优的结果是否达到预期效果或者为系统的选择提供决策数据。基准测试一般基于配置测试，通过配置测试得到数据，并将这个数据作为基准来比较每次调优后的性能是否有所改善。

2.2 Web 性能指标

前面我们了解了性能测试的研究现状，那么性能测试到底要测什么呢？

对于一个 Web 应用系统来说，我们所需要得到的性能指标主要有以下四点。

1) 响应时间

响应时间是反映被测系统完成某个业务所需要的时间。

例如从单击注册按钮到注册完成返回注册成功的页面所需要的时间需要消耗 1 秒钟，我们就说这个操作的响应时间是 1 秒钟。

在现有的商业性能测试软件(LoadRunner)中是通过事务函数来完成对响应时间的统计，事务时指做某件事情的操作，事务函数会记录开始做这件事情和事情做完之间的时间差。

此外，文[25]中，提出了一个新的测试指标-加权响应时间。假设一个连接建立的时间为 S；一个页面 P 包括 n 个文件，第 i 个文件的响应时间为 R_i；T 为使用 HTTP/1.1 获得页面 P 的总时间，如(2.1)所示：

$$T = S + R_1 + R_2 + \ldots + R_n = S + \sum_{i=1}^{n} R_i \tag{2.1}$$

令 $\overline{R}$ 表示各个文件响应时间的平均值，则使用 HTTP/1.1 获得一个文件的平均消耗的时间为 $\overline{T}$， 如(2.2)所示：

$$\overline{T} = \frac{T}{n} = \frac{S}{n} + \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{S}{n} + \overline{R} \tag{2.2}$$

由此，提出了一个加权响应时间的性能指标，$\overline{T} = \omega S + \overline{R}$ 这里 ω 为一个连接所传送文件数目的倒数，即通过一个连接所传送的文件分担了建立该连接的开销，加权响应时间和响应时间相比更能体现用户的感受。

2) 吞吐量

吞吐量反映被测系统单位时间内能够处理的事务数目。

例如对于某个系统来说一个用户登录需要1秒钟，如果系统同时支持10个用户登录，且响应时间是1秒钟，我们就说被测系统的吞吐量是10个/秒。此外，还有一种计算方式，计算被测系统在单位时间内传输的字节数。

在LoadRunner中，吞吐量被称为TPS(Transaction Per Second，每秒事务数)也就是在单位时间内能处理完成的事务数目。TPS 的计算一般是通过的事务除以时间。

3) 容量区间

容量区间是我们建立的一种新的性能指标，它指的是被测系统吞吐量的最大区间，由于在测试过程中一定会出现测试指标的波动，有可能出现数值异常的点，所以我们不能确定到达最大值的时候就一定是性能达到极限的时候，所以为了更好的定位系统的性能，我们通过容量区间来描述系统的性能，在测试数据分析部分，我们将给出得到容量区间的具体算法。

4) 服务器资源占用

服务器资源占用反映了在不同负载下系统的资源利用率。资源的消耗越低，说明系统越优秀。资源并不仅仅指运行系统的硬件，而是支持整个系统运行的一切软硬件平台。在性能测试中，我们需要监控被测系统在不同负载下的硬件或者软件上各种资源的使用情况，例如CPU的占用率，内存使用率，磁盘剩余空间等。

对于一个终端用户来讲，最关心的指标恐怕只有响应时间，如果响应时间长了，那么用户就会觉得系统慢，用户并不关心有多少人在使用这个系统以及系统的资源是不是足够，但是在性能测试中，服务器资源占用情况是非常重要的性能指标，方便我们选择不同的服务器配置以及对Web系统进行综合分析和性能预测。

2.3 Web性能测试原理

Web 性能测试的核心是基于 Web 负载的压力测试，通过压力测试可以测试 Web 系统的性能。通过构建一个测试环境，测试工具使用多线程技术来模拟多用户对 Web 服务器的并发访问并且接收响应数据。通过对测试结果数据的统计和分析，测试工具最终获得 Web 服务器的各项性能参数，依据优化规则，测试人员得到 Web 系统的优化建议。为了减少网络拥塞对测试的影响，测试环境应构建在局域网中。多个线程发送请求来模拟多个客户端访问服务器，每一个线程代表一个客户端。对服务器的一个访问包括一个主请求和一些内嵌资源请求。首先，浏览器向服务器发送主请求，当客户端收到来自服务器的响应后，浏览器根据来自服

务器的响应信息发送内嵌资源请求。

为了记录客户端浏览器访问服务器的过程，我们通过设置 Web 代理来捕获和记录详细信息，例如请求的顺序，请求头内容等。捕获和记录的过程称之为脚本录制过程。 根据测试脚本中请求的顺序，请求头内容，测试工具将会启动线程向服务器发送请求，同时接收来自服务器的响应数据，这个过程我们称之为测试执行过程。最后，测试系统根据服务器的原始测试数据统计出各项性能参数，以及获取性能监控日志，这个过程我们称之为数据分析过程。

2.4 Web 性能测试方法

针对不同的应用类型和侧重点，目前主要有以下几种性能测试方法，虚拟用户方法，WUS 方法和对象驱动方法，它们的比较如图 2.1 所示：

比较项	虚拟用户方法	WUS方法	对象驱动方法
负载的基本单位	真实用户的商务处理	经常被访问的路径	页面组件对象的行为
脚本的形式	(商务处理1,2，...，n)	(被访页面1,2，...，n)	(对象行为1,2，...，n)
获取负载信息的途径	人工收集和分析	挖掘日志文件	---
适合的程序类型	电子商务应用程序	有已发布版本的程序	页面组件类型多的程序
优点	方法直观，有工具支持	科学精确地确定负载	结构化的测试方法
缺点	负载靠人工收集	负载的确定依赖日志	强调局部性能

图 2.1 测试方法的比较

1) 虚拟用户方法是通过模拟真实用户的行为来对被测系统施加负载，从而获得被测系统的性能参数值，如事务的平均响应时间，服务器的吞吐量等。该方法主要适用于电子商务应用程序，它以真实用户为完成一个商业业务而执行的一系列操作作为基本单位，用模拟用户行为的测试脚本来进行负载测试。测试需求如并发用户数，测试执行频率等主要通过人工收集来获得。目前，大部分测试工具已经支持该方法，可以用较少的硬件资源来模拟成千上万的虚拟用户并发访问被测系统，同时监视系统的性能指标，帮助测试人员分析测试结果。

该方法的优点是简单，直观，已经有测试工具来支持，缺点是负载需求主要通过人工收集，精确性不高。

2) 基于网站使用签名(website usage signature)方法，一个不能反映真实负载的性能测试是没有用的，并且有可能会产生误导作用，过高的估计 Web 系统的能力可能导致系统发布之后出现崩溃的情况，过低的估计将会产生不必要的成本，而基于网站使用签名方法的提出是为了衡量测试负载和真实负载的接近程度，WUS 方法是一种系统和高效的方法，能够快速的确定真实负载需求，它是一系列能够全面刻画真实负载的参数和测量指标的集合，包括了每小时浏览的页面数，访问持续时间以及页面请求分布等。该方法认为，如果负载测试的 WUS 和实际的 WUS 相匹配，我们就能把测试结论很好的用到实际情况中。

该方法的优点是测试负载通过挖掘网站实际运行的日志文件来获得，能够反映真实负载情况，缺点是太依赖于日志文件，所以不适合新开发的系统。

3)对象驱动方法，该方法的基本是想是将被测系统分解成一个个可以测试的对象，例如按钮，链接，列表框，可以下载的文件，视频等。对象定义的粒度取决于应用系统的复杂性。一个网页可以通过对象来嵌套定义，性能测试的过程就变成测试每个对象或者某些对象的集合。这些对象的行为组成了负载的基本单位。

该方法的优点是测试结构化程度高，可重用性好，结果清晰，适合页面组件较多，业务复杂的应用系统。缺点是过于强调局部组件的性能难以反映用户对系统的整体感受。

2.5 本章小结

本章主要针对 Web 应用性能测试的相关内容进行了研究，主要包括 Web 性能指标，测试的原理以及测试方法，使我们对 Web 应用性能测试相关知识有了深入的理解，为下一章中测试方法的建立提供了帮助。

第三章 协议分析及测试方法的建立

性能测试工具的录制行为和行为模拟都是基于协议的，如果不了解协议就不能够深刻理解脚本录制和回放脚本的原理，所以首先我们需要对协议进行深入的研究。

3.1 HTTP 协议概述

超文本传送协议(Hypertext Transfer Protocol)是万维网(World Wide Web)的基础[26]，这里主要对 HTTP 进行介绍，而不介绍有关 Web 和浏览器方面的知识。

HTTP 是一个属于应用层的面向对象的协议，使用在分布式超媒体信息系统。它在 1990 年提出，经过长期的应用和发展，目前在万维网中使用的是 HTTP1.1。

HTTP 最初的设计目标就是通过网络来支持 Client 和 Server 之间的事务处理，其原型在 1990 年被提出，为了适应 WWW 的需求，在功能和性能方面进行了大量的改进，最开始出现的 HTTP 协议现在被称为 HTTP0..9，它是一个面向消息的简单协议，是现在使用 HTTP 的子集，因此它同 HTTP1.0 和 HTTP1.1 兼容，该协议描述了 Client 和 Server 之间的请求响应过程。

HTTP0.9 和现在使用的 HTTP1.1 在实现上没有太大的区别，HTTP1.0 是以 HTTP0.9 为基础发展起来的，增加了复杂网络连接下访问不同对象类型的功能。

HTTP1.1 是在 HTTP1.0 的基础之上实现了一次飞跃，主要的改进集中在性能，安全，数据类型处理方面。

提出了客户端缓冲对象的概念，其目的是为了减少网络上相同类型内容的反复传送，节省带宽，提高访问效率。

使用永久连接(Keep-Alive)作为基本连接，提高了性能。

允许客户端和服务器端对内容进行协商。

突破了 HTTP1.0 中服务器和 IP 一一对应的限制，可以通过主机名来决定由哪一台 Server 提供服务。

目前我们已经对 HTTP 的发展有了一个了解，可以把 HTTP 的特点概括为以下几点：

1) 简单快速

客户向服务器发送请求时，只需要传送请求方法，路径以及必要的参数即可，请求方法主要有 HEAD，POST，GET 等，其中又以 GET 应用最为广泛。由于 HTTP 比较简单，使得 HTTP 服务器的程序规模小，通信速度快。

2) 灵活

HTTP 允许传输任意类型的数据对象(ASC 码文本，二进制流等等)，传输数据的具体类型在 Content-type 头域中加以说明。

3) 无连接

传统 TCP 应用主要面向专用系统，这种环境中客户端的数目是有限的，意味者服务器最多开几十个服务进程修行了，而客户端与服务器需要连续地交换数据，频繁地连接和断开对两端都难以接受。

HTTP 协议产生于互联网，因此服务器需要处理同时面向全世界数十万、上百万客户端的网页访问，但每个客户端（即浏览器）与服务器之间交换数据的间歇性较大（即传输具有突发性、瞬时性），并且网页浏览的联想性、发散性导致两次传送的数据关联性很低，如果按照传统 TCP 应用的方式则需要在服务器端开的进程和句柄数目都是不可接受的，所以 HTTP 的设计者有意利用这种特点将协议设计为请求时建连接、请求完释放连接，以尽快将资源释放出来服务其他客户端。

HTTP 是一个面向事务的客户服务器协议，虽然 HTTP 使用了 TCP，但是 HTTP 是无状态的，也就是说每一个事务都单独的进行处理，当一个事务开始时，就在客户与服务器之间产生一个 TCP 连接，当事务结束后就释放这个 TCP 连接。

从 HTTP 的角度来看，每一个 WWW 浏览器就是一个 HTTP 客户端，而 Web 服务器等待 HTTP 请求的进程称之为 daemon 进程，daemon 进程在收到 HTTP 请求后，经过必要的处理，将响应数据返回给客户端。

下面我们来研究一下 HTTP 的报文结构。HTTP 有两类报文：从客户端到服务器端的请求报文和从服务器端到客户端的响应报文。两种报文都有 5 个成员组成，报文结构如下：

1. 第 1 成员：请求行(Request-Line)或状态行(Status-Line)
2. 第 2 成员：通用头(General-Header)
3. 第 3 成员：请求头(Request-Header)或响应头(Response-Header)
4. 第 4 成员：实体头(Entity-Header)
5. 第 5 成员：实体主体(Entity-Body)

简单地说，HTTP 请求主要有三部分组成，分别是：

1. 方法-URI-协议/版本
2. 请求头
3. 请求正文

图 3.1 是一个 HTTP 请求的例子：从该图中我们可以看到一个 HTTP 请求信息的具体组成结构。

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shock
wave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application
/msword, */*
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0
.50727)
Host: www.google.com.hk
Connection: Keep-Alive
Cookie: PREF=ID=3d65ca45abe94367:U=b3d83f5bdfa9dabc:FF=2:LD=zh-CN:NW=1:TM=128998
0799:LM=1293673404:S=B52NTg2IxU-pyifM; NID=44=YpG313ETqnnS4K_HV7waDuzSFcEH8fnx2Y
UjyUC0R1jRSesoKgw5TEAafMZcy7waE47qvcsYcdW-iX-aSVRprmu5QE_0WPxZ6vpoamNGUPjwGSbTtf
WFSvn6Wm-7KSzk
```

图 3.1 一个 HTTP 请求

请求报文的第一行是“方法-URI-协议/版本”：

GET / HTTP/1.1

其中 Get 就是请求方法，/表示请求的 URI，HTTP/1.1 是协议和协议的版本。

HTTP/1.1 支持 7 种请求方法，包括 GET，POST，HEAD，OPTIONS，PUT，DELETE 和 TRACE。

在 Web 应用中，最常用的请求方法是 GET 和 POST。

URI 完整地指定了要访问的网络资源，通常认为它相对于网络服务器的根目录而言，因此总是以“/”开头。URL 实际上是 URI 的一种类型，最后协议版本指明了通信过程中所使用的 HTTP 版本。

此外，请求头还包含了许多客户端环境和请求正文的有用信息，如 Accept 请求头，说明所能够解析的数据类型，也就是能够在客户端浏览器直接打开的数据格式。

例如，其中的 application/x-shockwave-flash 说明在浏览器上可以直接播放 Flash 文件。

Accept-Language：zh-cn.。

Accept-Language 请求头说明了客户端所使用的语言，这就是为什么我们访问 www.sina.com 时会自动跳转到 www.sina.com.cn 页面上的原因。如果使用中文的操作系统，一般这个属性值都是 zh-cn。

Accept-Encoding：gzip，deflate。

Accept-Encoding 请求头是指客户端所能接受的编码规则或者格式规范。这里说明客户端能够接受服务器返回通过 gzip 压缩的数据信息，这个功能也可以称为动态压缩，通过对返回数据的压缩，可以有效地减少网络传输所浪费的时间，提高传输效率。

User-Agent：Mozilla4.0(compatible，MSIE 6.0，Windows NT 5.1，.NET CLR 2.0)。

User-Agent 请求头主要包含了客户端的信息，对于服务器来讲，如果没有这个

信息就不知道客户端使用 WWW 服务的环境。这里的请求头告诉服务器客户端使用的是 IE6.0，windows NT 5.1 的操作系统信息，这也是为什么有些网站能够识别你的浏览器并跳转到不同页面的原因。

Host 请求头表示请求的主机地址，请求头和请求正文之间是一个回车换行(\r\n)，这个行非常重要，它表示请求头已经结束，接下来的是请求正文。

Cookie 是由服务器端生成，发送给 User-Agent（一般是浏览器），浏览器会将 Cookie 的 key/value 保存到某个目录下的文本文件内，下次请求同一网站时就发送该 Cookie 给服务器。

和 HTTP 请求相似，HTTP 应答也主要有三个部分组成：

1. 协议-状态码-描述
2. 应答头
3. 应答正文

图 3.2 是对应上面 HTTP 请求的 HTTP 应答，由于响应报文的 Body 数据量大，故在此省略。

```
HTTP/1.1 200 OK
Date: Fri, 04 Mar 2011 02:56:34 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 5361
X-XSS-Protection: 1; mode=block
```

图 3.2 一个 HTTP 应答

HTTP 应答的第一行类似于 HTTP 请求的第一行，它表示采用 HTTP/1.1 进行通信，服务器端已经成功处理了客户端发来的请求(状态码 200 表示成功)。

除此之外服务器还返回了以下信息：

Data：服务器上的 GMT 格林威治时间。

Content-Type：实体类型，是文本还是其它类型的数据。

Content-Length：实体长度，附在在头部之后的数据长度。

Server：服务器上的 WWW 服务名称。

最后是服务器返回的实体数据，也就是服务器返回的 HTML 页面，这里我们并没有显示出来。

当发出一个 HTTP 请求后，服务器会返回一个 HTTP 应答，所以整个网页就是这样从服务器上下载下来的，然而我们所浏览的网页并不是一次 HTTP 请求返回的结果。

事实上，我们所看到的一个完整的网页是有很多 HTTP 请求组成的，一般来

讲，我们发出的第一个请求称之为主请求，然后浏览器会对响应数据做一个解释工作，一边解析一边将内嵌资源对象(如 CSS，SWF，JPEG，GIF 等等)的请求发送出去，服务器再将包含资源对象的响应数据返回给客户端，这样我们就看到了一个完整的网页，数据交互过程如图 3.3 所示：

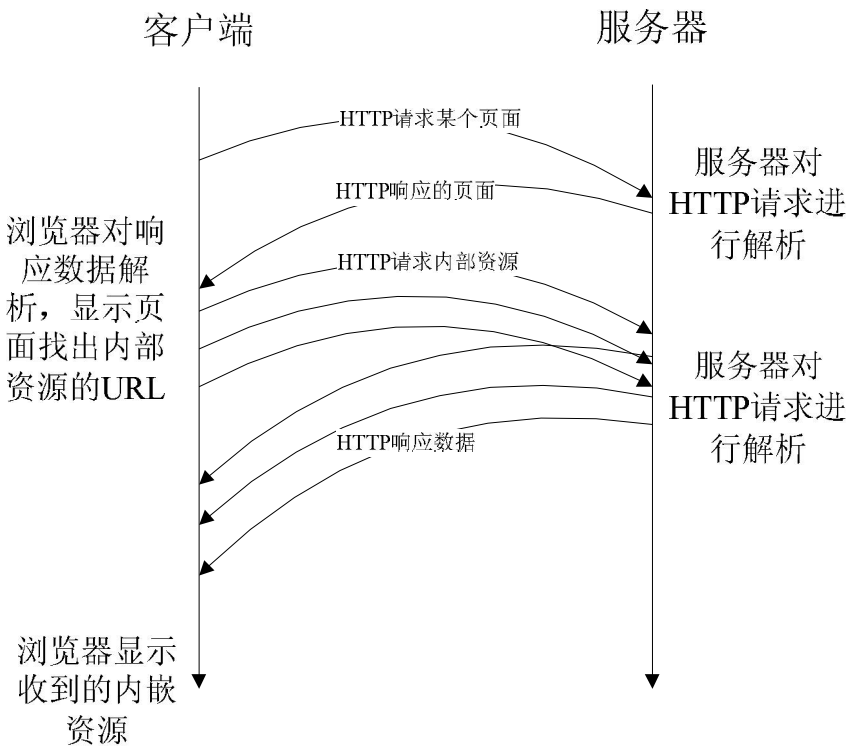

图 3.3 HTTP 协议的数据交互过程

3.2 协议分析在测试中的应用

本章前面部分对 HTTP 协议进行了研究和分析，为我们后面对性能测试软件的设计起到了积极地作用，特别是对测试用例中测试数据的组成和分析提供了很大的帮助。

我们知道模拟 Client 和 Server 进行通信的测试数据基本上就是 HTTP 协议数据，通过对协议的分析让我们知道有可能会出现以下情况：

1) 在一个真实的 Web 应用中，服务器往往需要认证与跟踪客户端的会话信息，所以服务器会生成一个包含会话 ID 的 Cookie，发送给 User-Agent（一般是浏览器），浏览器会将 Cookie 的 key/value 保存到某个目录下的文本文件内，下次请求同一网站时就发送该 Cookie 给服务器。所以说，为了模拟真实的用户行为场景，我们需要在 HTTP 请求中包含 Cookie 头域。

2) HTTP/1.1 中提出了一种 Web 缓存技术，降低了 Web 服务器的压力，提高了服务器的效率，但是对于性能测试来说，由于缓存中已经有数据而 Web 服务器

不需要重新发出相应数据，这样就对服务器来说就不能产生压力，不利于测出服务器的真实性能。

3) 上面提到了 Web 缓存技术，然而我们需要知道缓存中的数据是否过期，HTTP 协议提供了一种带条件的请求机制，使得在允许客户端进行高速缓存的同时，仍然保证传送到浏览器的数据对象都是最新的。即在请求报文中，添加了 If-Modified-Since 头域。所以说我们在进行性能测试发送请求数据后，有可能会收到状态码为 304(客户端有缓冲的文档并发出了一个条件性的请求，一般是提供 if-modified-since 头表示客户只想获得比指定日期更新的文档。服务器告诉客户，原来缓冲的文档还可以继续使用。)的响应消息，不会返回响应文件，从而不能给服务器造成有效的压力。

在我们所开发的性能测试软件，通过用代理服务器来捕获用户点击页面所发出的一个个 HTTP 请求，形成测试数据。在代理服务器模块所记录下的数据中，包含了 HTTP 请求的所有头域，当然也包括继主请求之后的嵌套资源请求和一些特殊头域。这样在形成测试脚本时我们需要对某些请求字段进行一定的修改。

3.3 一种联合性能测试方法

通过对 Web 应用性能测试理论和实践的分析和研究，我们发现 Web 应用性能测试主要包含三个方面，应用在客户端的性能测试，应用在网络的性能测试，应用在服务器端的性能测试。一般情况下，三者有效，合理的结合，才能对 Web 应用系统进行全面的性能分析和性能预测。

应用在客户端的性能测试主要是从用户的角度出发来考察 Web 应用系统的性能，测试的入口点是客户端的浏览器，通过负载测试和压力测试来分析响应数据，得到 Web 应用的性能指标，如最大并发用户数，平均响应时间，系统平均吞吐量等，来评价系统当前的性能。

应用在网络的性能测试主要是利用成熟先进的自动化技术进行应用在网络上的性能分析和预测。由于 Web 用户经历着不可预测的网络延迟，这些都取决于连接的带宽和网络拥塞。网络带宽和拥塞不在我们的研究范围之内，所以我们并没有考虑应用在网络的性能测试。

应用在服务器端的性能测试，称之为应用在服务器端的性能监控，可以采用现有的工具进行监控，也可以使用操作系统本身的监控工具或者监控命令来进行性能监控。该测试往往伴随着应用在客户端的性能测试进行，得到 Web 服务器在不同负载下的监控日志，为综合性能分析提供帮助。性能监控部分能够暴露出最终会导致服务器崩溃的内存泄漏以及硬件资源的使用情况，通过对监控日志的分

析从而决定是否要添置硬件资源或者是否要对被测系统进行整体架构的变更。

由于目前的性能测试往往侧重应用在客户端的性能测试，而忽视了应用在服务器端的性能测试，在此我们提出了一种联合性能测试方法，该方法强调在执行客户端并发性能测试的过程中，同时进行服务器端的性能监控，以便对系统的性能的作全面的评估。我们使用该方法的架构如图 3.4 所示。图中，左边部分是负载测试，右边部分是性能监控部分。首先分析 Web 应用系统，选择有代表性的，关键的业务操作来设计测试用例，然后在两部分的共同作用下得到 Web 系统的性能指标。

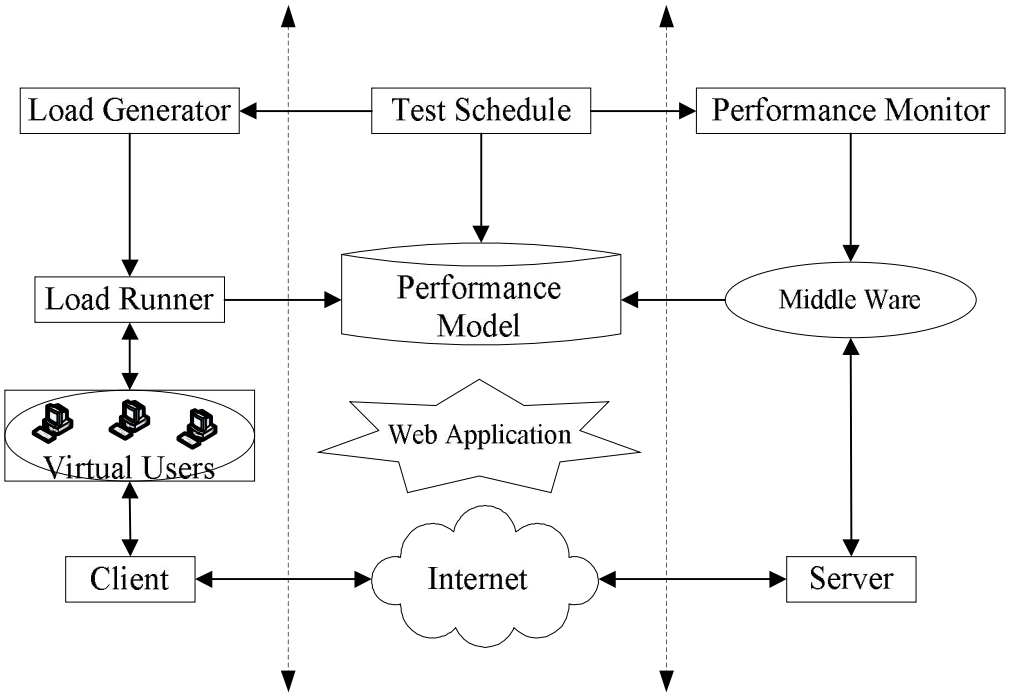

图 3.4 联合性能测试架构图

左边部分的负载测试，目的是获得被测 Web 应用系统在不同并发用户数下的性能参数。它的主要流程是：通过对被测 Web 应用系统的分析来确定测试计划，然后由脚本录制部分的代理服务器模块来形成测试脚本，然后通过负载产生器在测试端上产生若干虚拟用户，执行测试脚本，并且不断变换负载强度，得到被测系统在不同负载下的性能指标。

右边部分是性能监控，伴随着负载测试进行的，目的是获取被测系统在不同负载下的各项输出参数。如：CPU 使用率，内存使用率，磁盘空间，这些数据为测试人员对被测系统进行全面的分析提供了帮助，方便测试人员更加快速的找出性能瓶颈，进行系统优化。

接下来，我们将讨论联合性能测试方法的核心流程，如图 3.5 所示：

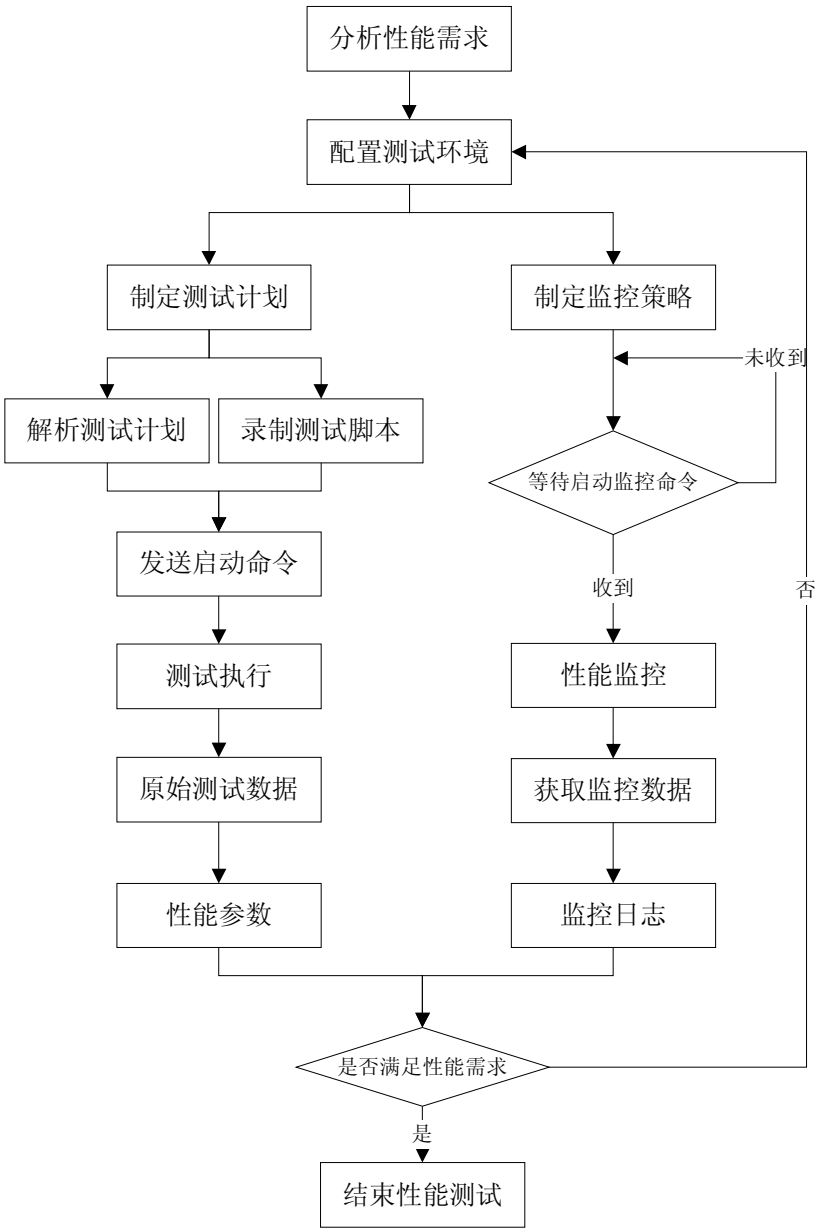

图 3.5 联合性能测试方法的核心流程图

从该测试方法的核心流程中，我们可以看出了整个性能测试过程包含了两个分支，一边是测试执行部分，一边是性能监控部分，两者有机地结合在一起。从而更加准确地得到被测系统的性能，定位系统的性能瓶颈。

关于测试执行部分，它主要是依据测试计划和测试脚本向后台服务器产生一定的负载，从而得到原始测试数据。

关于性能监控部分，由于现有的性能监控工具不够灵活，如不能同时监控多个服务器以及为每一个服务器指定不同的监控策略和异常策略，所以我们采用 Web Service 技术灵活地实现了后台监控，弥补了现有监控工具的不足。最终，我们在测试执行的同时，得到了服务器的监控日志。

通过对原始测试数据进行统计和分析，我们可以得到被测系统的性能参数，

然后测试人员结合监控日志，能够发现被测系统是否满足性能需求，如果不能满足性能需求，测试人员就需要重新进行性能测试。

所以，依据该方法我们设计并实现了一个简洁，灵活，实用的性能测试系统，并且得到了真实环境的应用。

3.4 本章小结

本章主要对 HTTP 协议进行了一定的分析，了解了协议请求以及应答的组成结构，为测试脚本的组成提供依据，此外通过上一章中对测试方法的研究以及测试的实践，我们建立了一种新的性能测试方法—联合性能测试方法，并且对该方法进行了详细的描述。

第四章 WebPT 测试系统的设计与实现

整个系统的设计和开发过程是采用统一软件开发过程 RUP[27](Rational Unified Process)的开发方法，该方法是一个面向对象且基于网络的程序开发方法论。从系统的需求分析，概要设计，详细设计，再到系统的实现以及后期的测试，一直贯穿着 RUP 方法的思想。

上一章的最后我们提出了一种联合性能测试方法，本章主要依据该方法设计并实现了 WebPT 测试系统，该系统主要包含两个部分，客户端性能测试部分和服务器端性能监控部分，在此，我们从系统的需求出发，逐一介绍了系统的功能和数据流程，在此基础上进行了系统的设计与实现。

4.1 系统设计概述

WebPT 测试系统的设计目的是为了给测试人员提供一个简洁，灵活，易用的测试工具，因此在系统设计上我们主要考虑了以下特性。

1. 易用性，界面应该简洁，清晰。为了方便用户的使用，界面部分应该清晰，明确，省略复杂的参数设置以及操作流程。用户只要在浏览器上执行一些操作，系统就会录制用户的行为，通过代理服务器获取所有的请求包，形成测试脚本，此外在性能监控部分，用户可以通过界面清晰地看到哪些服务器出现异常。所有这些操作都方便了用户的使用。

2. 直观性，测试系统能够通过测试脚本清楚地看到所有的请求信息，以及服务器端的性能参数，及时发现服务器的异常。性能监控部分可以直观地看到各个服务器的性能日志。

3. 可控性，用户可以根据自己的需要配置测试参数，如虚拟并发用户数，请求次数等等。此外在性能监控部分中，用户也可以灵活地指定每个服务器的监控策略和异常策略。这些都保证了系统的可控性。

接下来，我们在测试系统的需求分析，详细设计以及系统实现中，紧紧围绕上述特性进行。

4.2 系统功能

WebPT 测试系统的功能主要包含四个部分：录制脚本部分，测试执行部分，性能监控部分，数据分析部分。如图 4.1 所示。

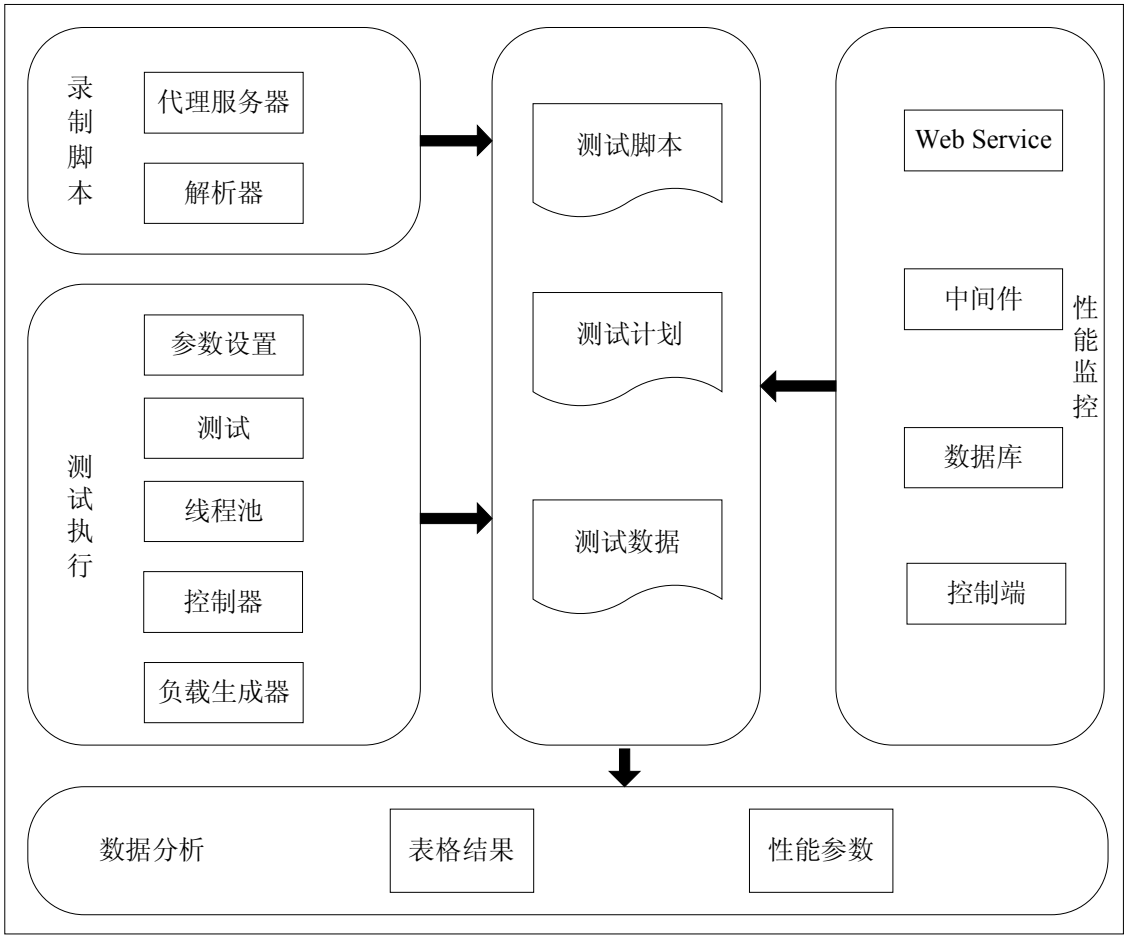

图 4.1 WebPT 测试系统功能框架图

录制脚本部分是为了录制用户的真实行为，为测试执行部分提供测试脚本，它主要有两部分组成，代理服务器和解析器。代理服务器就像一个通讯员，负责传递客户端和服务器之间的消息，本来，客户端和服务器之间是直接通信的，设置代理服务器之后，客户端发给服务器的所有请求消息都先交给代理服务器，由代理服务器转发给服务器，同样地，服务器发给客户端的所有响应消息也先交给代理服务器，由代理服务器转发给客户端。这样，代理服务器就可以捕获所有客户端传给服务器的请求消息，然后将这些消息传给数据解析器，解析器负责将这些请求消息按照 HTTP 协议进行解析，获得请求消息头中的头域值，生成请求列表，然后将这些请求写入 XML 文件中，形成测试脚本。

测试执行部分主要是分析性能需求，确定测试计划，利用录制脚本部分生成的测试脚本，对服务器产生负载，达到性能测试的目标。主要由以下几个部分组成，参数设置，线程池，控制器，负载生成器等。

参数设置主要是为了配置用于性能测试的一些参数，如并发用户数，服务器地址，服务器端口，循环次数等。为测试部分的执行提供依据。

线程池是用来存放线程的地方，我们知道，在性能测试过程中，一个线程用

于模拟一个用户，采用线程池来管理线程，可以避免由于频繁的创建线程，销毁线程所带来的开销，提高了测试系统自身的效率。线程池的大小是由测试计划中的并发用户数的多少来决定的。

控制器是整个测试执行的核心，负责控制整个性能测试。

负载生成器是真正向后台施压的部分，它依据控制器传输的不同数据集命令产生不同的操作，最终向后台发送大量的请求数据，对后台形成负载。

性能监控部分的目标是为了获得服务器在各种负载下的不同输出项，如 CPU 使用率，内存使用率，磁盘空间等等。为了克服操作系统自身命令工具和现有监控工具的局限性，我们基于 Web Service 技术设计并实现了一种后台监控工具，该工具可以同时监控多台服务器，同时后台管理人员也可以为每台服务器指定不同的监控策略和服务器异常策略，高效灵活地实现了服务器端的性能监控，弥补了现有工具的不足。该工具主要有四部分组成：部署了 Web Service 的服务器，中间件，控制端，数据库。性能监控部分的框架图如图 4.2 所示：

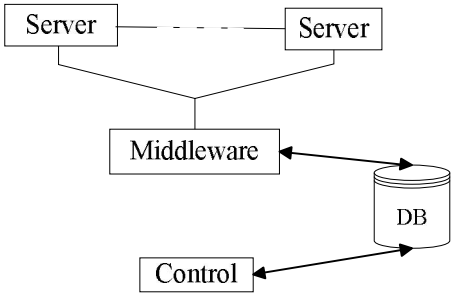

图 4.2 性能监控框架图

Web Service 是一种基于 SOAP(简单对象访问协议)和 XML 技术的互联网应用框架。Web Service 具有语言无关性和平台独立性，使用它可以向外界发布如何调用自身功能和服务的说明，包括服务地址，服务接口和每个服务所需要接受的参数等说明信息。客户端通过 HTTP 协议发送 SOAP 格式的消息，服务器端接受客户端的请求后返回处理结果。为了防止他人调用后台服务接口来获取监控数据，我们要求调用后台服务接口时必须提供服务器指定的验证码，该验证码是由服务器采用 RC2 加密算法得到的 256 位密钥。

控制端主要实现了后台服务器的添加，删除，服务器状态和监控日志的查询，以及为每台服务器指定监控策略和服务器异常策略，然后控制端将这些策略存储在数据库中。

中间件使用多线程技术，每一个线程代表了一个后台服务器，线程通过查询数据库获取自身的监控策略和服务器异常策略，依据监控策略调用后台提供的 Web Service 接口来获取相应的监控数据，根据服务器异常策略进行数据处理，更新服务器的状态，同时将这些数据写入数据库的日志表中，方便控制端的查询。

数据分析部分主要是对原始测试数据进行统计和分析，得到被测系统的性能参数，供测试人员结合性能监控日志判断被测系统是否满足性能需求。

4.3 系统设计

我们在上节中介绍了 WebPT 性能测试系统的主要功能，把整个系统划分为四个部分，并对每一个部分的功能进行了详细的描述，本节我们将按照 RUP 的开发方法，对系统的分析和设计过程进行深入的讲解。

4.3.1 系统需求分析

以系统功能为依据，建立业务模型，分析系统的需求。通过设计用例，来描述系统的功能需求，为后面的进一步设计提供铺垫，在此，我们通过用例图来描述系统的各个部分的功能需求，直观地描述了系统的需求。如图 4.3 所示：

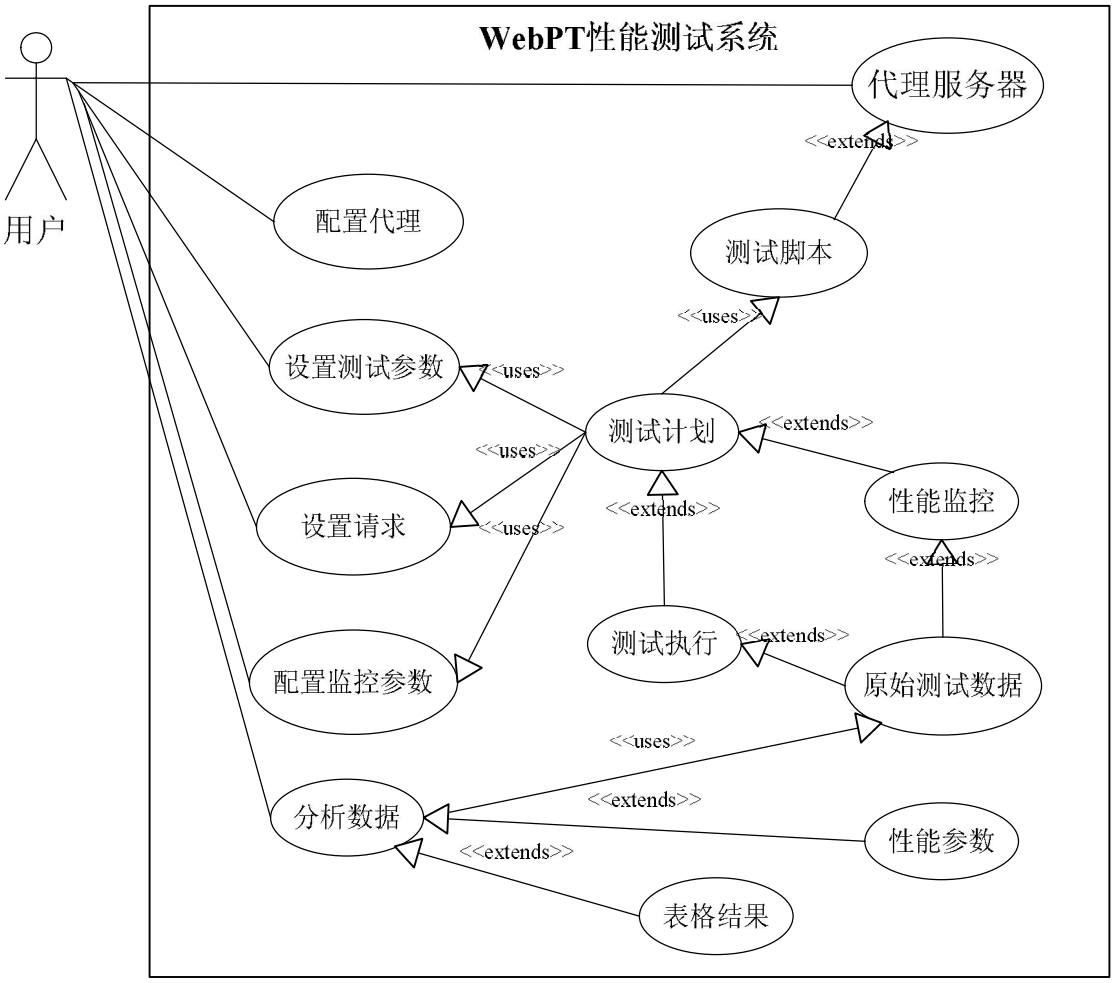

图 4.3 WebPT 测试系统的用例图

上图是对 WebPT 测试系统设计的顶级用例图，图中每一个用例代表测试系统的功能，依据业务逻辑关系，我们在各个用例之间关联了某种关系，如使用，扩

展关系，用户(Actor)通过启动代理，设置测试参数，设置请求，配置监控参数，分析数据来作用于系统，代理服务器获取用户的请求消息并解析生成了测试脚本，结合负载参数，请求参数，监控参数形成测试计划，测试用例和监控用例是整个系统中最为关键的用例，它们依据测试计划来对服务器进行测试和监控，并生成原始测试数据，数据分析用例通过统计和分析数据库中的原始测试数据得到被测系统在不同并发用户数下的性能参数。

顶级用例图以较大的用例粒度对系统进行了功能需求的描述，为后面的详细设计提供了帮助。

4.3.2 系统流程设计

上一小节我们从业务逻辑模型及关联关系的角度，对系统的需求进行了分析，并用顶级用例图对系统的功能需求进行了描述。这一小节，我们从系统的数据流向的角度出发，详细分析了系统中各个模块的业务模型和关联关系，建立了系统的数据流程图，如图 4.4 所示。

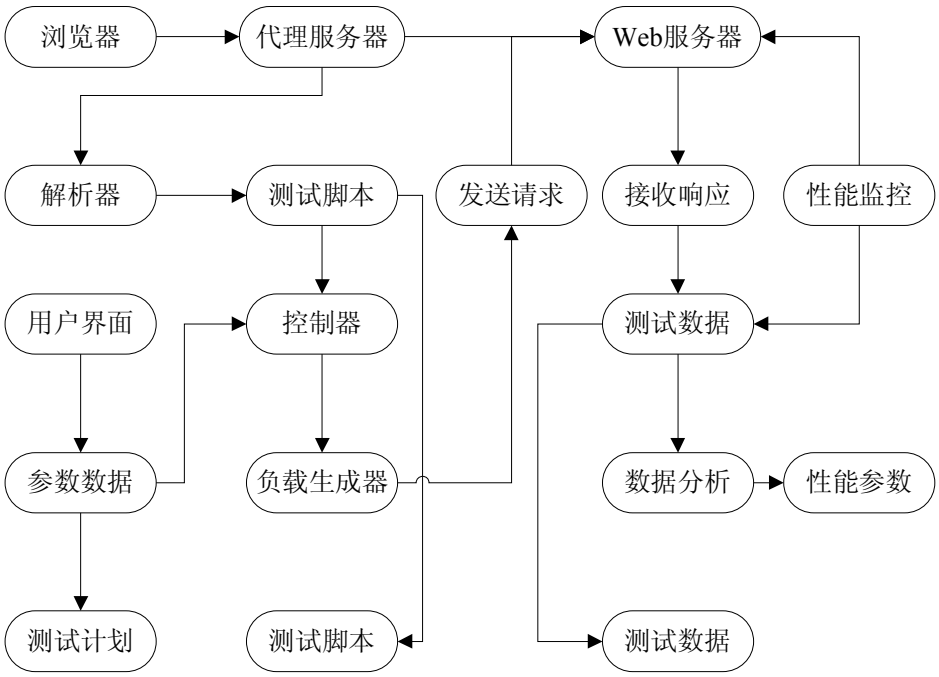

图 4.4 系统数据流程图

首先，用户需要通过浏览器设置好代理(IP 和 Port)，对需要测试的业务进行操作。代理服务器会捕获到用户发出的所有请求数据，然后传递给解析器和真实的 Web 服务器，解析器负责将这些请求消息按照 HTTP 协议进行解析，获得请求消息头中的头域值，生成请求列表，然后将这些请求写入 XML 文件中，形成测试脚本。这个过程就实现了录制测试脚本的功能。

用户界面部分主要用于用户输入参数数据和查看测试数据，用户通过界面可

以输入一些参数数据形成测试计划，如并发用户数，服务器地址，服务器端口，循环次数等。这些参数数据被保存为配置文件，方便控制器的解析。同时用户也可以通过界面来添加和删除想要监控的服务器，为每一个服务器指定不同的监控策略和异常策略，实现了灵活的性能监控。

控制器主要实现了整个测试过程的控制，位于管理端，它主要将测试计划中的参数数据和测试脚本传递给负载生成器。负载生成器依据测试计划中的参数数据产生一个线程池，线程池的大小主要由测试计划中并发用户数的大小来决定。控制器是整个系统的核心部分，通过向负载生成器和性能监控部分发送启动命令，实现了负载生成器并发向后台服务器提交请求，以及实时的性能监控。

负载生成器位于负载终端，它是真正给服务器施加压力的部分，测试执行时，每一个虚拟用户建立与 Web 服务器的连接，依据测试脚本向服务器发送请求，记录下发送时间，同时接收服务器的响应数据，记录下接收时间，响应数据的大小，形成原始测试数据，保存在数据库中。

性能监控部分往往伴随着负载生成器的执行而进行的，依据用户为每台服务器指定的监控策略和异常策略，进行后台的性能监控，得到服务器的监控日志，存储在数据库中，方便测试人员分析。

数据分析组件主要对原始测试数据进行统计，计算和分析，得出性能参数，供测试人员结合性能监控日志来判断被测系统的性能瓶颈。

4.3.3 系统详细设计

在系统详细设计过程中，我们将按照如下三个步骤来展开：

1. 设计分析类，依据对 WebPT 性能测试系统的用例分析，获得测试系统活动的关键抽象，以及每一个用例的功能描述，然后从不同角度设计那些能够协作完成功能用例的分析类。分析类可以形象的称之为点的集合。

2. 转述需求场景，我们把分析类的对象作为行为的载体，通过消息传递的方式实现了用例场景的分析，把上一小节中讨论的数据流程图中的数据流向转化为消息，用顺序图来描述功能需求和系统数据流程，这种方式可以形象地称之为线的集合。

3. 整理分析类，依据分析类的对象在用例场景中的消息传递关系，归纳分析类所承担的责任和它们之间的关系。分析结果用参与类图来说明，参与类图对分析类的责任和它们之间的关系进行描述。这个过程可以形象的理解为一个面。

围绕以上三个步骤，我们可以把系统的用例图和数据流程图转换为顺序图和协作图，通过设计分析类和整理分析类，我们可以绘制参与类图以及确定类的属

性和方法。

由于 WebPT 测试系统的亮点在于灵活，高效，易用的性能监控，在此，我们首先介绍性能监控部分的详细设计。性能监控部分主要由四部分组成：部署了 Web Service 的服务器，中间件，控制端，数据库。由于控制端和中间件都需要频繁地访问数据库，数据库成为联系中间件和控制端的纽带，它是整个服务器性能监控部分的核心所在，因此设计出一个合理的数据库，不仅能够提高数据库本身的执行效率，而且能够使整个监控部分稳定，高效的执行。那么首先我们就来介绍数据库的设计。

该监控工具采用 SQL Server 2005 作为数据库管理系统，通过对业务数据的分析，我们建立了 Server 表，PolicyFile 表，PolicyList 表，CheckLogs 表，DataType 表，StateType 表。数据库主要维护了三类数据，服务器，服务器策略，服务器监控日志。Server 表主要维护服务器的基本信息，如服务器的 WSDL URL 和服务器的状态。PolicyList 表主要维护各个服务器的监控策略和异常策略。CheckLogs 表主要维护了每台服务器的日志数据，方便控制端的查询。而其它数据表的添加是为了防止数据冗余。正是由于数据库的使用，不仅实现了数据的持久存储，而且灵活的实现了后台服务器的监控策略和判定异常的标准。

接下来我们讨论中间件的设计，中间件的主要功能是依据监控策略，调用后台 Web Service 服务接口，获取监控数据。服务接口有: GetCpu()获取当前 CPU 使用率、GetFileCheck()获取指定目录下文件个数、GetProcessList()获取进程列表、GetDisk()获取当前磁盘下各个盘符的剩余空间、GetMemory()获取当前内存使用情况、GetServerDate()获取指定数据库中一张数据表的某个字段的值、GetServerDataCount()获取指定数据库中一张表的某个字段的记录个数。然而中间件要同时监控多台服务器，所以我们采用多线程技术，根据 Server 表中后台个数，建立多个线程，每个线程代表一个后台服务器。线程执行算法如下所示：

1) 建立数据库连接，得到连接对象和游标对象，如果失败，则线程退出。

2) 查询 Server 表中自身的状态，如果服务器状态为 4，即服务器正在维护，则重新查询自身的状态。

3) 查询 Server 中的 WSDL URL，获取 WSDL 文件的本地代理，如果失败，更新服务器状态为 2，即服务器网络中断，转到 2)。

4) 依据 Server 表中的 ServerID，从 PolicyList 表中获取自身的所有监控策略和异常策略，利用 2)中的本地代理，调用 Web Service 服务接口，获取监控数据，并根据异常策略判定数据是否异常，如果出现异常，更新服务器状态为 3，即服务器异常。如果执行完所有的监控策略没有出现异常，则更新服务器状态为 1，即服务器正常。当执行完所有的监控策略后，休眠一定时间后，转到 2)。

由于中间件是持续执行的，为了防止程序退出执行，我们采用了异常处理机制，增强了程序的健壮性，此外我们采用锁对象实现了线程的同步，使多个线程有条不紊的高效执行。

接下来我们介绍性能测试部分的设计，这里主要介绍界面和测试执行部分的设计。通过对界面部分分析类的设计，确定了每一个类的责任和它们之间的关系，为每一个类添加了属性和方法，在此我们给出了界面的参与类图 4.5。

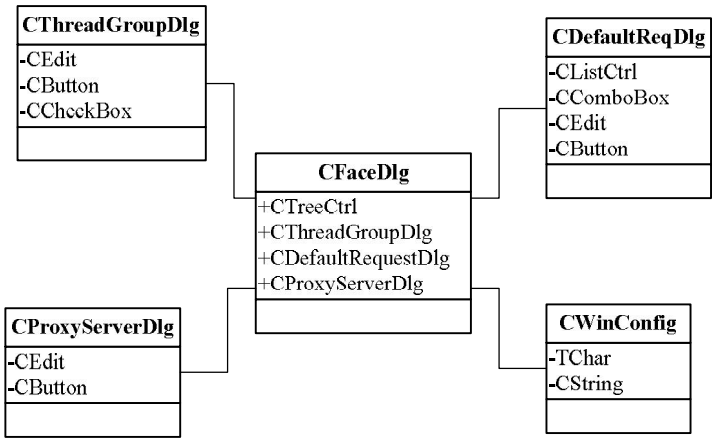

图 4.5 界面参与类图

主界面 CFaceDlg 类是系统的主界面类，它和 CTreeCtrl 类是组合关系，CTreeCtrl 类的对象是树形控件的实例，供用户进行选择不同的设置，CThreadGroupDlg 类是用户设置线程组属性的界面类，CDefaultReqDlg 是用户录入默认请求的界面类，用来录入默认请求的一些参数，CProxyDlg 是用户设置代理服务器的界面类，用于设置代理服务器地址和端口。

前面我们对用户使用的界面设计进行了分析，接下来我们讨论测试执行的具体设计。首先，我们给出了测试执行部分的体系构架图。如图 4.6 所示：

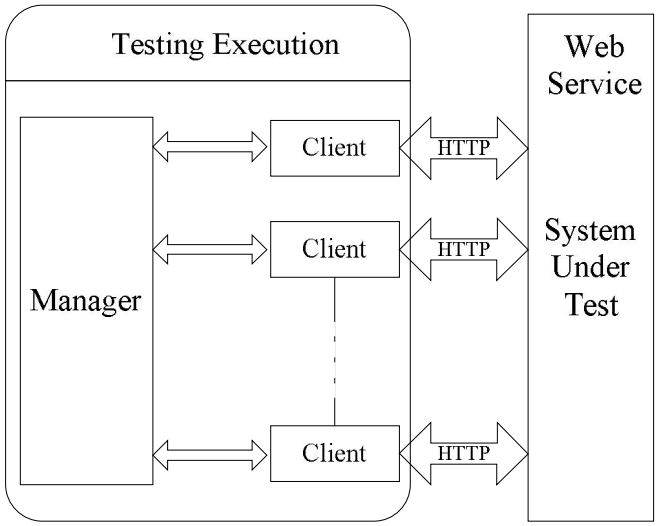

图 4.6 测试执行部分的体系架构图

整个测试执行部分主要由三部分组成，管理端，测试端，被测系统组成。其中管理端主要实现了测试计划配置文件和测试脚本 XML 文件的解析，以及将解析后的数据集传送给负载终端，如测试计划数据集，测试脚本数据集，启动命令数据集。在此，由于负载终端的不确定性，我们通过 UDP 广播的方式将数据集传输到负载终端。这些数据集是由管理端根据用户的命令生成的，如用户输入 transTestPLan 命令，控制器就会通过解析配置文件生成测试计划数据集，并通过 UDP 广播到测试端。

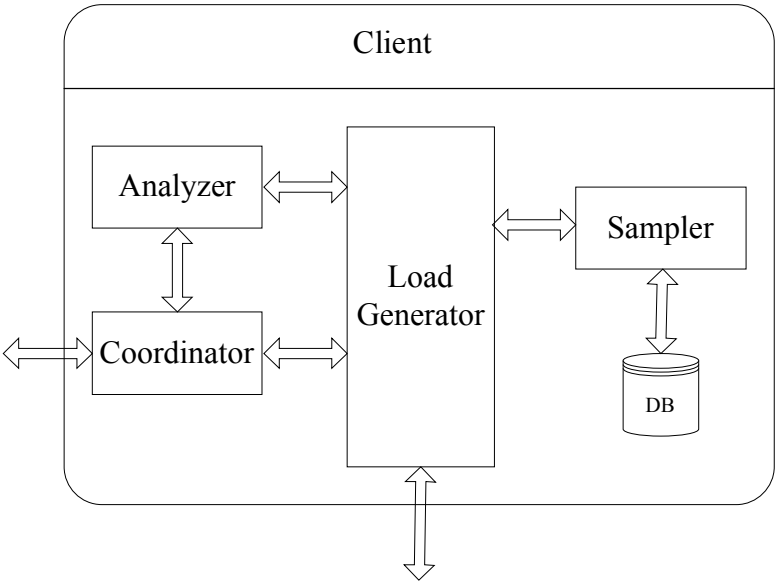

图 4.7 测试端的体系架构图

测试端主要由数据集解析器，协调器，负载生成器，采样器四部分组成。协调器主要负责接收由控制端发送的数据集，然后将数据集传送给数据集解析器，解析器根据不同的数据集名称做出相应的处理，如果是测试计划数据集，解析器就会依据数据内容产生一个线程池，由线程池动态产生一定数量的测试线程。如果是测试脚本数据集，解析器就会依据数据内容解析出符合自己发送格式的测试脚本，供测试线程向后台发送数据。如果是启动命令数据集，解析器就会启动线程池中的测试线程，负载生成器中的测试线程依据测试计划和测试脚本向后台服务器发送数据，同时记录请求 URI，发送时间，响应时间，数据长度，响应状态形成原始测试数据，采样器主要负责从负载生成器中取出这些原始测试数据并将这些数据写到数据库中，供数据解析部分进行统计和分析。

4.4 系统实现

本节依据对 WebPT 的功能需求的分析，分别对系统各个功能部分的实现进行了描述，主要包括录制脚本功能的实现，测试执行功能的实现，性能监控功能的实现，数据分析功能的实现。

4.4.1 脚本录制功能的实现

脚本录制功能的实现主要有两部分组成，代理服务器和数据解析器。代理服务器的功能是捕获用户浏览网页时所发出的所有请求消息，而数据解析器实现了对这些请求消息的解析，得到各个请求消息的 HTTP 头，并把它们写到 xml 文件中，形成测试脚本。

首先用户通过界面输入代理服务器的名称和端口，名称是代理服务器的名称，而端口是代理服务器启动后所监听的端口。界面如图 4.8 所示：

图 4.8 WebPT 的代理服务器设置

当用户在浏览网站时，首先会和代理服务器建立一条连接 cs1，用户发送的主请求消息通过 cs1 发送到代理服务器，代理服务器对主请求消息解析后，获取目标服务器的 IP 和 Port，然后和目标服务器建立一条连接 ws1，将客户端发来的所有请求消息通过 ws1 传递给目标服务器，同时将请求消息复制一份保存在本地的一个链表中，供解析器获取。如图 4.9 所示：

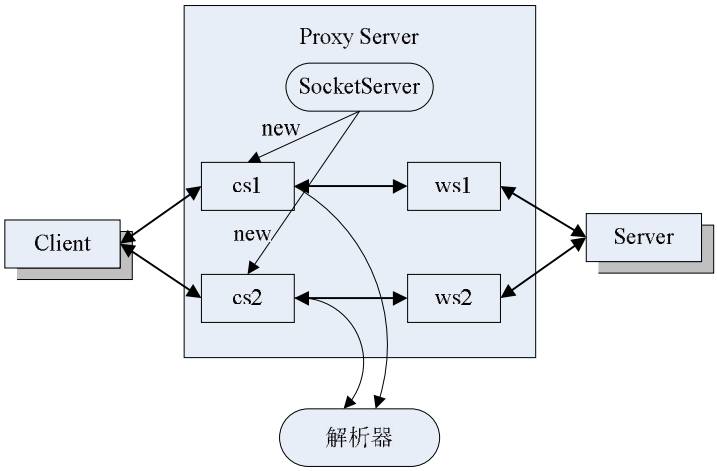

图 4.9 代理服务器实现原理

数据解析器负责从请求链表中取出请求消息按照 HTTP 协议来解析，得到每一个请求消息的 HTTP 请求头，然后将这些请求消息写到 xml 文件中，形成测试脚本。由于代理服务器和数据分析器都要对消息链表进行操作，在此我们通过临界区来实现它们对数据链表的互斥访问。以上就是对录制脚本部分的实现进行了简单的描述。

4.4.2 测试执行功能的实现

测试执行功能是整个系统的核心，它实现了应用在客户端的性能测试。它包含了测试计划的制定，测试计划的执行，测试结果的存储三个部分。首先我们需要分析性能需求，制定测试计划，测试计划主要包括线程组属性设置，默认 HTTP 请求设置等等。

线程组属性设置是测试计划中十分关键的部分，它主要包含了线程数，服务器地址，服务器端口，循环次数。我们知道，在测试过程中，每一个线程模拟了一个虚拟用户，线程数的大小代表了需要并发执行的虚拟用户数，循环次数代表了每一个虚拟用户执行测试脚本的次数。服务器地址和端口代表了后台服务器的 IP 地址和端口号。线程组设置的界面如图 4.10 所示：

图 4.10 WebPT 的线程组属性设置

当用户需要对某个页面进行测试时，我们可以不通过测试脚本，直接设置 Web 服务器地址和端口号以及 HTTP 请求参数，如协议，方法，路径和参数。但是 WebPT 对系统进行测试时，只能选择一种测试方式。HTTP 请求设置界面如图 4.11 所示：

图 4.11 WebPT 的 HTTP 请求设置

测试计划的执行主要由管理端的控制器和测试端的负载生成器组成。

用户通过界面来设置测试计划的相关参数，主要包括线程数，测试脚本循环次数，服务器地址和端口号等等。然后将这些参数保存在配置文件中，供控制器解析。控制器是由 Python 脚本语言来实现的，主要负责将测试计划数据集，测试脚本数据集，启动测试命令数据集通过 UDP 广播传输到测试端。

当测试端接收到来自控制器的数据集后，首先发送一个确认数据包，告知测试端已经收到数据集命令，然后将数据集中的数据存放到一个 Queue 中，解析器将会从 Queue 中取出数据，依据数据集名称做出不同的处理工作，如测试计划数据集：TestPlan user:10;repeat:5;server:192.168.12.40;port:8080。解析器将会获取数据集名称 TestPlan，然后解析数据内容，获得虚拟用户数为 10，测试脚本循环次数为 5，服务器 IP 为 192.168.12.40，服务器端口为 8080。然后，就会启动一个线程池，线程池依据虚拟用户数产生一定数量的测试线程。当测试端接收到 start 数据集时，线程池就会启动测试线程，负载生成器依据服务器 IP，端口以及测试脚本向后台服务器发送请求数据，形成负载，同时记录发送时间，响应时间，内容类型，数据长度，形成原始测试数据。

测试结果的存储主要是由测试端的采样器从负载生成器中获取原始测试数据，将这些数据存储在数据库中，数据表的设计如图 4.12 所示。由于采样器和负载生成器都要对原始测试数据进行一定的处理，所以在此我们采用生产者—消费者算法来实现数据的同步。

表 - dbo.testdata

列名	数据类型
number	nvarchar(50)
url	nvarchar(MAX)
sendtime	float
recvtime	float
reponsetime	float
lenth	int
status	int

图 4.12 原始测试数据表

4.4.3 性能监控功能的实现

性能监控功能是整个系统的重要组成部分，它实现了应用在服务器端的性能测试，即服务器端的性能监控。

在此，我们主要讨论中间件的实现，控制端是用.net 开发的一个管理界面，在此不再详细阐述。中间件采用当前流行的脚本语言 Python 进行开发，执行界面如图 4.13 所示。

```
executeMethod 35 GetFileCheck
proxy.GetFileCheck(SecCode=' ********** -BD-D2-72-FF-F3-67-F2-91-9C-17-68-34-43-C
D-C6-26-A8-98-2F-6C-39-9F-51-BC-41-23-17-8C',DirName='F:\\swich\\send')
==========ret: 27


executeMethod 37 GetCpu
proxy.GetCpu(SecCode=' ********** -BD-D2-72-FF-F3-67-F2-91-9C-17-68-34-43-CD-C6-2
6-A8-98-2F-6C-39-9F-51-BC-41-23-17-8C')
==========ret: 25.8


executeMethod 38 GetCpu
proxy.GetCpu(SecCode=' ********** -BD-D2-72-FF-F3-67-F2-91-9C-17-68-34-43-CD-C6-2
6-A8-98-2F-6C-39-9F-51-BC-41-23-17-8C')
==========ret: 4.2


executeMethod 25 GetServerData
proxy.GetServerData(SecCode=' ********** -BD-D2-72-FF-F3-67-F2-91-9C-17-68-34-43-
CD-C6-26-A8-98-2F-6C-39-9F-51-BC-41-23-17-8C',DataBase='Userinfo',Table='dbo.Act
iveLogs',Fields='SessionId',StrWhere=' ',OrderBy='SessionId Desc',Paras=' ')
==========ret: 331919410264273
```

图 4.13 中间件执行界面图

通过对中间件的设计分析，我们将中间件分为三个功能模块。

(1)：配置模块，主要实现了对配置文件的解析，配置文件主要包含数据库连接的一些字段，如 host，user，password，database 等，以及线程休眠时间和调用服务接口的验证码。

(2)：服务器线程模块：主要实现了一个服务器线程类，线程类中关于数据库的处理我们采用了 pymssql 模块。部分代码如下所示：

```
class ServerThread(threading.Thread):
    def __init__(self, id, lock, wsdlurl):
        threading.Thread.__init__(self)
        self.id=id
        self.lock=lock
        self.wsdlurl=url
    def getSleepTime(self)
    def run(self):
```

(3)：Web 服务模块：主要实现了一个 web 服务解析类，采用 SOAPpy 模块，获取 WSDL 文件的本地代理。部分代码如下所示：

```
class Analyzer:
    def __init__(self, wsdlurl):
        self.wsdlurl=wsdlurl
        self.proxy=WSDL.Proxy(wsdlurl)
    def getMethods(self):
    def getMethodInfo(self):
```

4.4.4 数据分析功能的实现

数据分析功能主要包括对原始测试数据的统计和分析，原始测试数据的表格显示，以及被测系统在不同并发用户数下的性能参数。

通过对原始测试数据的统计和分析，我们可以得到一些性能参数，如最大响应时间，平均响应时间，平均吞吐量等。

原始测试数据的表格显示我们采用 MFC 的列表控件来实现。通过列表控件我们可以给用户展示数据库中的原始测试数据。

性能参数主要通过数据库的 SQL 语句来统计和分析。

4.5 WebPT 测试系统的主要技术

4.5.1 线程池技术

通过前面对整个系统设计与实现的分析，我们知道在负载终端上执行的负载生成器需要模拟多个用户对服务器并发访问，这时我们就要启动多个线程来实现，为了高效的管理这些线程，我们采用线程池技术。

传统的多线程方案是创建一个线程，让它来执行一定的任务，任务执行完毕后，线程退出。这就是即时创建，即时销毁的策略。尽管与创建进程相比，创建

线程的时间已经大大的缩短。但是如果提交给线程的任务是执行时间较短，而且执行次数及其频烦，那么我们就会处于不停得创建线程，销毁线程的状态。

因此线程池的出现大大减少了传统多线程方案所带来的开销。线程池采用预创建的技术，当负载生成器接收到测试计划数据集后，依据虚拟用户参数，立即创建一定数量得测试线程，将这些测试线程放入一个空闲列表中。这些线程处于阻塞状态(Suspended State)，并没有开始执行，所以不会消耗系统资源，只是占用较少的内存空间来存放这些线程对象。当负载生成器接收到启动命令后，线程池就会启动空闲列表中的所有线程，有效的实现了并发操作。

Python 脚本语言的多线程技术是实现并发的一种非常有效的手段，而且使用起来非常方便，我们只需要继承 threading 模块的 Threading 类，重写 run 方法就可以了。而且 threading 模块中包含了多种线程同步的资源，如 Lock 类，Condition 类，Event 类，Semaphore 类等等。

因此我们采用 python 的 threading 模块来建立了线程池框架，主要包含了工作者线程，线程池类，任务接口。主要代码如下所示：

```
class Worker(Thread):
    worker_count = 0
    def __init__( self, workQueue, resultQueue, timeout = 0, **kwds):
        Thread.__init__( self, **kwds )
        self.id = Worker.worker_count
        Worker.worker_count += 1
        self.setDaemon( True )
        self.workQueue = workQueue
        self.resultQueue = resultQueue
    def run( self ):
        while True:
            try:
                callable, args, kwds = self.workQueue.get(timeout=self.timeout)
                res = callable(*args, **kwds)
                self.resultQueue.put( res )
            except Queue.Empty:
                break
            except :
                print 'worker[%2d]' % self.id, sys.exc_info()[:2]
```

```
class ThreadPool:
    def __init__( self, num_of_workers=10, timeout = 1):
        self.workQueue = Queue.Queue()
        self.resultQueue = Queue.Queue()
        self.workers = []
        self.timeout = timeout
        self._recruitThreads( num_of_workers )
    def _recruitThreads( self, num_of_workers ):
        for i in range( num_of_workers ):
            worker = Worker( self.workQueue, self.resultQueue, self.timeout )
            self.workers.append(worker)
    def start(self):
        for w in self.workers:
            w.start()
    def add_job( self, callable, *args, **kwds ):
        self.workQueue.put( (callable, args, kwds) )
    def get_result( self, *args, **kwds ):
        return self.resultQueue.get( *args, **kwds )
    def job(id, sleep = 0.001 ):
        pass
```

该线程池框架高效地实现了线程池的创建，以及对空闲列表中线程的管理。当负载生成器接收到来自控制器的start命令后，通过线程池调用start方法就可以启动所有的线程(虚拟用户)，达到并发访问的目标，同时减少了频繁创建，销毁线程所带来的开销，节省了系统资源，提高了系统利用率。

4.5.2 XML 技术

XML是可扩展的标记语言，标记语言就是对文档的哪部分是内容，哪部分是标记以及这些标记的含义是什么的形式化的描述。它的根源为文档管理，并由一种用来构造大型文档的语言(被称为标准通用标记语言(Standard Generalized Markup Language), SGML)派生出来。但是不同于SGML和HTML，XML不仅能表示许多其他类型的结构化数据，而且能表示数据库数据。

在一个应用程序必须与另一个应用程序进行通信的时候，或是从一些其他的应用程序整合信息的时候，XML作为一种数据格式特别有用，和HTML不同，XML

没有指定的标签集，每个应用可以选择自己需要的标签集。这项特性是XML主要用在数据表示和交换而HTML主要用于文档格式化的关键所在。

在WebPT测试系统中，XML主要使用在XML文档的生成，解析，处理等操作。在录制测试脚本时，由解析器依据HTTP协议分析请求消息，生成XML文档，形成测试脚本。控制器负责解析和处理测试脚本。Python在解析XML文档的方式主要有两种：SAX(Simple API for XML)和DOM(Document Object Model)。SAX的工作方式是，一次读出一点XML，对发现的每个元素调用一个方法。SAX是XML语法分析器的公用语法分析器接口，它允许应用程序作者编写使用XML语法分析器的应用程序，当XML文档较大时，由于DOM解析的效率低下，建议使用SAX来解析。DOM的工作方式是，一次读出整个文档，通过将本地的python类链接到一个树型结构中，生成对文档的内存表示。

我们在整个WebPT测试系统中采用DOM方式来解析XML文档。首先在录制脚本功能中，解析器通过使用tinyxml的第三方库来实现测试脚本的生成。TinyXML是一个开源的解析XML的解析库，使用起来非常方便，能够用于C++，能够在Windows或Linux中编译。这个解析库的模型通过解析XML文件，然后在内存中生成DOM模型，从而让我们很方便的遍历这棵XML树。测试脚本的XML文档如图4.14所示：

```
<?xml version="1.0" ?>
<TestScript>
    <HttpRequest>
        <RequesetHeader>GET http://192.168.12.99:8080/oa/ HTTP/1.1</RequesetHeader>
        <RequesetHeader>Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg */*</RequesetHeader>
        <RequesetHeader>Accept-Language: zh-cn</RequesetHeader>
        <RequesetHeader>Accept-Encoding: gzip, deflate</RequesetHeader>
        <RequesetHeader>User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)</RequesetHeader>
        <RequesetHeader>Host: 192.168.12.99:8080</RequesetHeader>
        <RequesetHeader>Proxy-Connection: Keep-Alive</RequesetHeader>
    </HttpRequest>
    <HttpRequest>
        <RequesetHeader>GET http://192.168.12.99:8080/oa/images/css2007.css HTTP/1.1</RequesetHeader>
        <RequesetHeader>Accept: */*</RequesetHeader>
        <RequesetHeader>Referer: http://192.168.12.99:8080/oa/</RequesetHeader>
        <RequesetHeader>Accept-Language: zh-cn</RequesetHeader>
        <RequesetHeader>Accept-Encoding: gzip, deflate</RequesetHeader>
        <RequesetHeader>User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)</RequesetHeader>
        <RequesetHeader>Host: 192.168.12.99:8080</RequesetHeader>
        <RequesetHeader>Proxy-Connection: Keep-Alive</RequesetHeader>
        <RequesetHeader>Cookie: JSESSIONID=58C1B9DCB82DBF5486524E9EDD9D097F</RequesetHeader>
    </HttpRequest>
</TestScript>
```

图4.14 测试脚本XML文档图

在上图所示的这个XML文档中，TestScript元素包含了所有HTTP请求，每一个HttpRequest元素代表了一个HTTP请求，它的第一个子元素代表了一个请求行，包含了请求方法，统一资源标识符，和协议类型，其余的子元素代表了请求头域。该文档构成了一个完整的测试脚本，供控制器解析和处理。

在控制器中，我们通过 xml.dom.minidom[28][29]模块来实现对文档的解析，首先通过该模块的 parse 接口获取整个文档的一个本地树型结构，然后根据文档结构一步一步的解析，形成一定格式的测试脚本数据，通过 UDP 广播传送给测试端。

4.6 本章小结

本章主要对 WebPT 测试系统的设计与实现进行了详细的描述，依据上一章的联合性能测试方法设计了 WebPT 测试系统，并且对该测试系统各个功能模块的实现进行了深入的描述，最后我们讲解了该测试系统中所采用的主要技术，线程池技术和 XML 技术。

第五章 WebPT 测试系统的运行结果和分析

为了验证 WebPT 测试系统的正确性和有效性，我们对重庆爱思网安信息技术有限公司的 OA 办公系统进行了性能测试，尝试找到该办公系统的性能问题，此外 WebPT 测试系统中的性能监控部分成功地应用于重庆市 Ikeeper 网吧计费管理系统的后台服务器性能监控，及时的发现了服务器的异常，同时进行产品的升级和优化，保证了 Web 应用系统的畅通运行。

在对 Web 应用系统进行性能测试之前，必须对被测系统进行充分的分析，找到被测系统的核心业务，一般业务，确定被测系统的性能目标，然后部署测试环境和设计测试用例。

5.1 被测系统分析

前面，我们对 Web 应用系统的性能测试进行了深入的研究和分析，并且探讨了一种联合性能测试方法，并依据该方法设计并实现了一套性能测试系统，现在我们利用该测试系统对 OA 办公系统进行性能测试。

5.1.1 被测系统架构分析

被测的 OA 办公系统主要提供了信息浏览，员工登录，员工签到，员工发帖等功能，由于该办公系统保存了员工的很多信息，诸如签到信息，发帖信息等等，这些信息包含了大量的文字描述。网站如果只是使用 HTML 静态网页技术，对这些大量信息的保存和管理就会非常复杂，所以该办公系统采用了动态 JSP 技术和数据库技术相结合的方式，包含了三层架构，第一层是 Web 层，第二层是逻辑业务层，第三层是数据库服务层。Web 服务器使用的是 Tomcat6.0，数据库采用的是 SQL Server 2005 数据库，在数据库连接部分我们还使用了数据库连接池技术。

Tomcat 是 Apache 软件基金会(Apache Software Foundation)的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现，Tomcat 6 支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定，而且免费，成为目前比较流行的 Web 应用服务器[30]。

5.1.2 被测系统的性能需求分析

对被测系统的性能需求进行分析是很重要的，它是整个性能测试工作的最基本前提，如果不能保证性能测试需求分析的正确性，即使性能测试工具使用的再

正确，性能测试执行的再顺利，也无法保证性能测试达到想要的效果，即可能无法发现性能瓶颈，或者无法发现在实际情况中可能出现的瓶颈，甚至给测试人员造成错误的判断。

简单地说，性能测试需求必须要包含有多少用户(who)在什么时间(when)或者持续多久(when)进行了什么业务，最终需要关注怎样的指标(how)。这就是性能需求的 4W1H 原则。

因此，在测试之前我们必须对被测系统的架构，所使用的技术进行研究，前面部分我们已经对这一部分就行了分析，此外我们还要了解被测系统的基本业务流程，发现哪些是关键业务，哪些是一般业务，从而为测试用例的编写提供依据。然后我们要了解被测系统想要达到的性能指标，通常情况下，系统拥有者都希望知道系统所能承受的最大并发用户数，平均响应时间，系统的吞吐量等参数，以及系统是否存在内存泄露等性能瓶颈和可能发现的性能瓶颈的优化方案，所以说性能需求分析是性能测试的前提。

5.1.3 被测系统的核心业务

通过对被测系统业务流程的分析和研究，我们得出了以下活动为关键活动，如下所示：

(1) 浏览主页：员工必须首先打开主页

(2) 员工登录：员工通过登录页面登录

(3) 员工签到：每天来到公司的考勤

(4) 信息浏览：员工可以浏览各种信息

(5) 员工发帖：员工根据个人需要可以向部门，或者公司进行发帖操作。

得到上述关键活动后，我们可以通过代理服务器录制需要进行性能测试的核心业务，形成测试脚本，得到测试脚本后，我们可以搭建测试环境来对被测系统进行性能测试。

5.2 测试实施过程

5.2.1 测试环境的搭建

接下来，我们需要搭建一个测试环境，运行 WebPT 测试系统对被测系统的核心业务进行性能测试，为了减少网络拥塞对测试的影响，我们在局域网中构建了一个测试环境，该测试环境如图 5.1 所示：

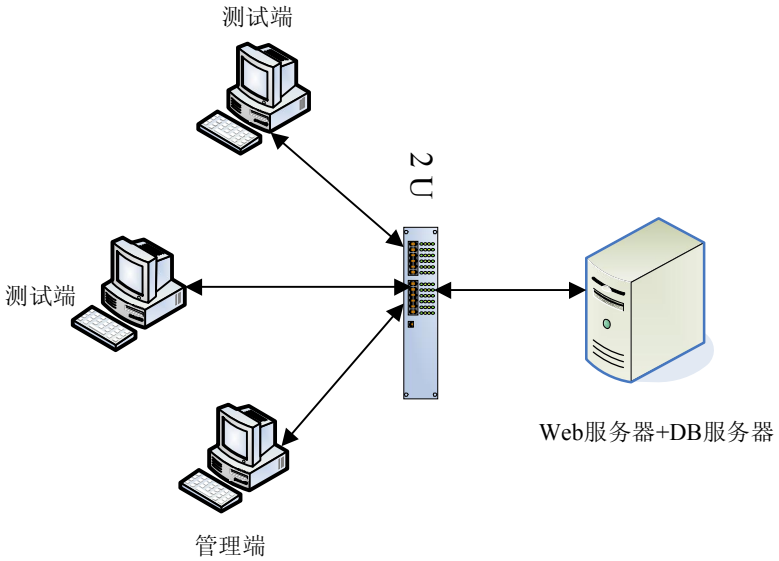

图 5.1 测试环境网络拓扑图

计算机配置：

控制端：Windows XP Professional SP3，IE8.0

CPU: AMD Dual Core Processor 5200+ 2.71GHZ

Memory: 2GB

测试端：Windows XP Professional SP3，IE8.0

CPU: Intel T1400 1.73GHZ

Memory: 1.5GB

服务器：Windows XP Professional SP3，IE8.0

CPU: AMD Dual Core Processor 5200+ 2.71GHZ

Memory: 2GB

Tomcat6.0, SQL Server 2005

5.2.2 性能测试步骤

前面部分我们已经对被测系统进行了研究和分析，确定了性能测试的需求，依据性能需求，我们可以设计测试用例，接下来主要有以下性能测试步骤：

(1) 录制脚本，并对脚本进行一定的修改

(2) 测试场景的设计

(3) 测试执行与性能监控

(4) 统计和分析原始测试数据，给出性能参数

录制脚本主要是通过代理服务器和解析器来完成，首先启动代理服务器，在指定的端口上进行监听，代理服务器将会捕获用户所发送的所有 HTTP 请求，同

时将请求放到一个队列中，解析器负责从队列中取得HTTP请求信息，并依据HTTP协议进行解析，存入到 TestScript.xml 文件中。由于 Web 服务器具有缓存技术，我们必须对录制好的脚本进行一定的修改，如设置 Cache-Contro 头域的值为 no-cache，Pragma 头域的值为 no-cache，同时删除 Last-Modified 头域。这样可以对服务器施加真正的压力。

测试场景的设计主要是让测试人员通过界面设置测试计划的相关参数(线程组属性，服务器地址和端口等)，添加监控策略和异常策略。

测试执行是通过管理端传输不同的数据集命令来进行的，当测试端接收到管理端的启动命令数据集后，便会启动所有的测试线程，依据测试脚本数据集，向 Web 服务器发送请求，同时形成原始测试数据，主要包括发送时间，接收时间，数据长度等等，将这些原始测试数据保存到一个同步队列中，采样器将会从这个同步队列中取出这些测试数据，写入到数据库中，性能监控是伴随着负载测试进行的。

5.3 实验数据验证

为了验证 WebPT 测试系统是否对后台服务器形成了真正的负载，我们在 tomcat 服务器上配置了访问日志，配置如下：

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs” prefix="localhost_access_log" suffix=".txt" pattern="common" resolveHosts="false"/>

访问日志与 catalina 容器相关，记录了该容器处理的所有请求。每条日志主要包含了访问者 IP，访问时间，HTTP 请求行，响应状态，响应数据大小，以及响应时间，实际的响应日志如下所示：

192.168.12.99 - - [22/Mar/2011:16:00:16 +0800] GET /oa/ HTTP/1.1 200 4137 0.000，该日志表示来自 192.168.12.99 的用户在下午四点访问了/oa/的资源，响应状态是 200，返回数据大小为 4137，由于只是对静态网页的访问，并不需要访问数据库，所以响应时间是 0.000。

这样我们在进行完性能测试后，通过把数据库中的原始测试数据和 Tomcat 服务器得到的日志数据进行比较，发现二者是一致的，验证了测试系统的正确性和原始测试数据的可用性。

5.4 实验数据分析

对实验数据的分析就是对负载生成器产生的原始测试数据进行统计和分析，得到应用在客户端的性能参数，如网站最多能够承受多少用户同时在线(最大并发

用户数)，平均响应时间，吞吐量，容量区间等，测试人员同时结合性能监控日志，定位被测系统的性能瓶颈，给出系统的优化建议。

本文用WebPT测试系统对公司OA办公系统进行了性能测试，主要测试了主页浏览，员工签到的核心业务，通过不断递增测试计划中的并发用户数和循环次数，观察被测系统的性能参数。

目前，WebPT测试系统能够记录下来的原始测试数据主要有以下几项：

请求序号

请求URI

请求发送时间

请求接收时间

请求响应时间

接收数据长度

响应状态码

图5.2是WebPT测试系统记录的原始测试数据：

num...	url	sendtime	recvtime	reponsetime	lenth	status
19-1	http://192.168.12.99:8...	1300954301.206	1300954301.315	0.109	9553	200
1-1	http://192.168.12.99:8...	1300954301.19	1300954301.315	0.125	9553	200
18-1	http://192.168.12.99:8...	1300954301.206	1300954301.315	0.109	9553	200
5-1	http://192.168.12.99:8...	1300954301.206	1300954301.315	0.109	9553	200
11-1	http://192.168.12.99:8...	1300954301.206	1300954301.315	0.109	9553	200
13-1	http://192.168.12.99:8...	1300954301.206	1300954301.331	0.125	9553	200
17-3	/oa/images/share2/login...	1300954301.299	1300954301.409	0.11	1090	200
15-2	/oa/images/share2/login...	1300954301.284	1300954301.409	0.125	48159	200
16-3	/oa/images/share2/login...	1300954301.299	1300954301.424	0.125	1090	200
0-3	/oa/images/share2/login...	1300954301.299	1300954301.424	0.125	1090	200
10-3	/oa/images/share2/login...	1300954301.299	1300954301.424	0.125	1090	200
14-2	/oa/images/share2/login...	1300954301.315	1300954301.424	0.109	48159	200
4-2	/oa/images/share2/login...	1300954301.299	1300954301.424	0.125	48159	200
7-3	/oa/images/share2/login...	1300954301.315	1300954301.424	0.109	1090	200
12-2	/oa/images/share2/login...	1300954301.315	1300954301.44	0.125	48159	200

图5.2 WebPT测试系统记录的原始测试数据

通过对原始测试数据的统计和分析，我们可以得到被测系统不同测试用例的性能参数。

测试用例：主页浏览和员工签到

功能：被测系统支持多个用户访问

目的：测试大量用户访问网站时系统的处理能力

方法：通过负载生成器模拟大量用户并发访问的操作

测试最好分阶段进行，通常从较轻的负载到中等负载再到重负载，这样做的好处是可以有效的调试系统问题，并且通过逐渐增加的用户数可以精确地找到性能下降的转折点。

表 5.1 性能参数表格

并发用户数	平均响应时间 (s)	最大响应时间 (s)	平均吞吐量 (B/S)
25	0.18398	0.828	3326.4586
50	0.38415	1.953	6041.7859
75	0.71392	3.703	12212.1563
100	0.90052	5.063	17507.3360
150	1.29092	6.016	22083.1206
200	2.60940	13.016	41040.6091
250	3.59623	15.231	25702.7645
300	4.23594	16.659	11653.7439

通过对原始测试数据的统计和分析，我们得到了表 5.1 所示的性能参数，我们可以发现被测系统的平均吞吐量开始时较小，随着负载的不断增加逐渐增大，增长到一个最大值后开始下降，其曲线类似于正弦曲线在[0 , Π]区间内的曲线，由于被测系统的性能测试受系统性能很多方面因素的影响，在测试过程中一定会出现测试指标的波动，有可能出现数值异常的点，所以我们不能确定到达最大值的时候就一定是性能达到极限的时候，所以在此我们给出了一个平均吞吐量的最大指标区间，下面是最大指标区间的算法：

第一步，将不同并发用户数的平均吞吐量读入。

第二步，求出平均吞吐量的均值 AVG。

第三步，找到平均吞吐量的最大值 Tp(i)，次大值 Tp(j)，记录下来。

第四步，求出 $avgTp(i)=\frac{1}{3}\sum_{k=i-1}^{i+1}Tp(k)$, $avgTp(j)=\frac{1}{3}\sum_{k=j-1}^{j+1}Tp(k)$

//找出 avgTp(i)和 avgTp(j)两者之间较大的值，则其对应的吞吐量为最大值，另一个为次大值。

if(avgTp(i) > avgTp(j))　then maxTp = Tp(i) , secTp = Tp(j)

else maxTp = Tp(j) , secTp = Tp(i)

//如果 avgTp(i)和 avgTp(j)相等，则如果 avgTp(i)与相邻的两个值的差不大，而 avgTp(j)与相邻的两个值的差很大，则 maxTp = Tp(i)。

第五步，从剩下的平均吞吐量中取出一个最大值 Tp(i)，同时令 Tp(j) = secTp,

如果 Tp(i) > AVG，则重复步骤四，否则结束算法。

第六步，最终我们得到了一个被测系统最大吞吐量的指标区间[maxTp , secTp] = [Tp[i] , Tp[j]]

这个算法是根据 Web 服务器的吞吐量本身的特性，在其服务器的吞吐量的变化过程中可以找到一个指标区间，在这个区间内服务器的吞吐量处于峰值，在这个区间之外吞吐量处于上升或下降趋势，通过对最大平均吞吐量区间的分析，我们可以确定服务器在并发用户数为[i ， j]内平均吞吐量最大，被测系统在并发用户数为 j 后开始下降。

通过该算法我们可以得到被测系统的容量区间为[22083 , 41040]，为了更加精确地确定容量区间，测试人员可以在 150 到 200 并发用户数之间进行更为精确地性能测试。

此外，我们使用 WebPT 测试系统中的性能监控部分对重庆市 Ikeeper 计费管理系统进行后台性能监控，该系统在全国拥有多台服务器。通过在后台部署 Web 服务，该监控工具成功地应用于后台监控，为后台管理人员进行服务器的维护和升级提供了帮助。中间件执行界面如图 5.3 所示：

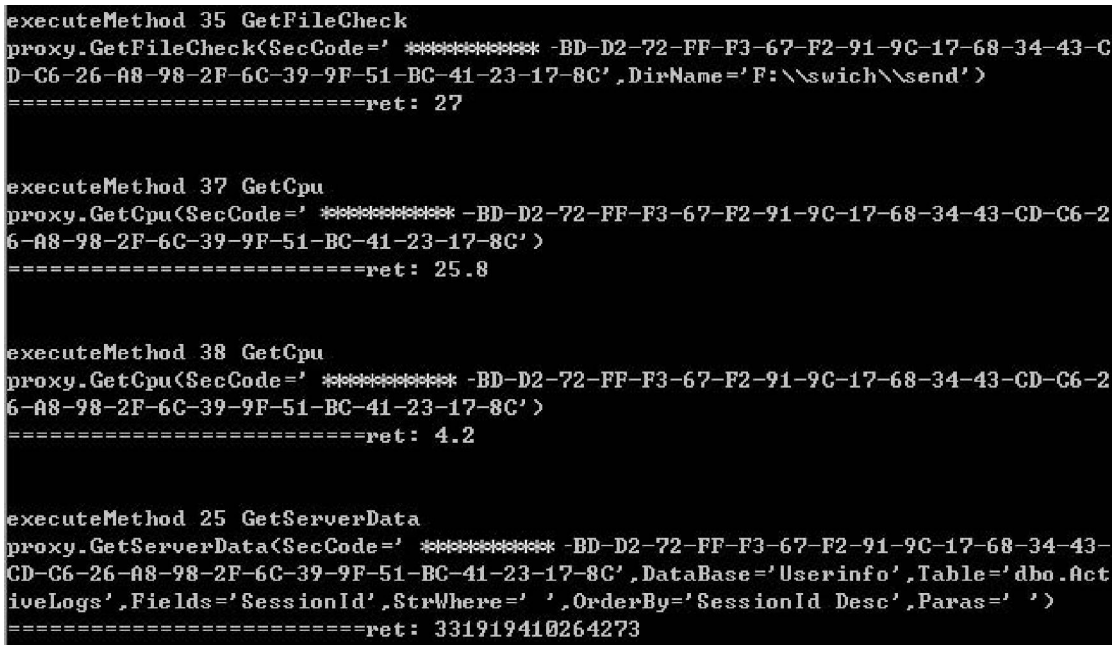

图 5.3 中间件执行界面

executeMethod 后包含了服务器 ID 和 web 服务接口，下面是通过本地代理调用服务接口以及参数信息，我们可以看到每一个接口的第一个参数都是 SecCode(验证码)，该验证码是服务器端采用 RC2 加密算法得到的 256 位密钥。最后一行是调用每一个接口的返回值。控制端管理界面如图 5.4 所示：

图 5.4 控制端管理界面

从图中我们可以看出，当服务器出现异常时，将会变成黄色，点击日志，我们可以看到日志信息，从而找到异常原因，进行服务器的维护。

5.5 本章小结

本章对被测系统进行了一定的分析，了解了被测系统的架构和核心业务，采用 WebPT 测试系统对核心业务进行了性能测试，得到了被测系统的原始测试数据，通过对原始测试数据的分析，得到了被测系统的性能参数，验证了该测试系统的正确性和有效性。

第六章 总结及未来工作

6.1 总结

1. 本文首先对 Web 系统的性能测试进行了研究和分析，主要分析了 Web 系统性能测试的重要性，性能测试的内容和现有的性能测试工具，并且对性能测试的原理进行了描述。

2. 本文对 Web 应用中使用的 HTTP 协议进行了研究，主要包括了协议数据的交互流程和协议数据的组成结构，描述了协议分析在性能测试中的应用，并且探讨了一种联合性能测试方法，提出了一种新的性能指标—容量区间，给出了得到容量区间的算法。

3. 本文依据前面研究的联合性能测试方法，采用 RUP 统一软件开发过程的开发方法，设计并实现了一个 Web 应用性能测试系统 WebPT，主要包含了脚本录制功能，测试执行功能，性能监控功能和数据分析功能，该工具实现了基于 HTTP 协议的网站的性能测试，给出网站的性能参数，方便测试人员定位系统瓶颈。

4. WebPT 测试系统的脚本录制功能实现了脚本的自动生成，我们依据 HTTP 协议支持通过代理来实现通信的特点，实现了一个代理服务器，通过设置 IE 通过代理来访问站点，代理服务器可以捕获到所有的 HTTP 请求，然后通过解析器来对这些 HTTP 请求进行解析，生成测试脚本。

5. WebPT 测试系统的测试执行功能主要由控制器和负载生成器来实现，分别位于管理端和测试端，通过控制器来实现将测试计划和测试脚本传输到测试端，负载生成器依据测试计划形成线程池，位于线程池中的测试线程依据测试脚本向后台施压，并形成原始测试数据，写入到数据库中。

6. WebPT 测试系统的性能监控功能是伴随着测试执行功能进行的，该功能依据 Web Service 技术来实现。通过测试人员编写监控策略和异常策略，灵活，实用地实现了性能监控，成功地应用于重庆市 Ikeeper 网吧计费管理系统的后台性能监控。

7. WebPT 测试系统的数据分析功能主要是对数据库中的原始测试数据进行统计和分析，得到被测系统的性能参数，供测试人员定位性能瓶颈，给出优化建议。

8. 最后，我们用 WebPT 测试系统对重庆爱思网安信息技术有限公司的 OA 办公系统进行了性能测试，得到了被测系统的性能参数，验证了 WebPT 测试系统的正确性和有效性。

6.2 未来工作

本文描述了 Web 系统性能测试系统的设计和实现方法，并依据联合性能测试方法设计并实现了一套性能测试系统，由于时间和精力有限，还有许多需要优化和完善的地方，主要包括以下几个方面：

1) 该测试系统只支持 HTTP 协议，我们应该增加对其它协议的支持，如 FTP，POP，SMTP 等常用协议的支持。

2) 该测试系统没有用代码来实现自动给出性能优化建议，这一点还需要逐步完善。

致谢

三年的研究生生涯即将结束，不论在学习上还是生活上，感触颇深，在这三年中我收获了很多，学习了许多理论知识的同时，也在两年的实习期间得到了许多实践的机会，使我慢慢的成熟起来。

感谢我的父母，是您们对我的无私奉献，对我成长的关爱，全力地支持我的学业，才使得我有今天的成绩，您们的恩情我会永远地铭记在心中。

感谢我的母校重庆邮电大学给我提供生活上的帮助和丰富的学习资源，以及我的恩师杜江副教授，给我提供了宝贵的实习机会，让我在实际的项目中得到了成长，使我学习到了很多在学校无法学习到的知识，领悟了工作和生活的道理，为我日后在社会上打拼奠定了坚实基础。

感谢重庆智多信息技术有限公司的杨德静，胡杨，蔡明辉等同事的帮助和指导，以及重庆爱思网安信息有限公司的陈巍、梅江等同事关心和支持，是你们的精心指导，才使我快速的掌握了软件研发的技巧，才得以胜任软件研发工程师这个岗位。和你们一起愉快地度过了两年实习工作生活，将是我人生中美好的回忆。

感谢同门蒋文豪，林珍妮，许德昭，李晔强，聂国新，黄华，褚帅，是你们陪我度过了研究生生涯，我们一起学习，一起玩耍，一起进步。我会时常想起你们，相信我们未来前途一片光明。

感谢所有关心和支持过我的同学与朋友们，我的成绩也是与你们的支持分不开的，向评阅论文的老师和答辩组专家表示由衷的感谢。

签名：王雷雷 日期：2011.5.27

攻读硕士学位期间从事的科研工作及发表的论文

- 从事的科研工作

  在爱思网安实习的两年里，主要参与了以下项目：

  [1] IKeeper 计费管理系统

  [2] 服务器性能监控系统

- 发表的论文

  [1] Jiang Du,Leilei Wang. Design and Implementation of Background Monitor Based on Web Service. 已录用, Second International Conference on Future Computer and Communication.Shanghai.2010: 301-303.

参考文献

[1] Lan Molyneaux 等, 李刚, 陈宇星主译. 应用程序性能测试的艺术[M]. 北京：机械工业出版社, 2010: 4-11.

[2] Zona Research lnc. The economic impacts of unacceptable web-site download speed[EB/OL]. http://www.keynote.com/downloads/whitepapers/economic_impact_of_downloadspeed.pdf: 3-5.

[3] ESRI System Integration Technical Brief. Web Application Stress Test Methodology[EB/OL]. http://www.esri.com/systemsint/kbase/docs/stress-test.pdf: 2-3.

[4] Mercury Interactive Corp. Load testing to predict Web performance[EB/OL]. http://www.bitpipe.com/resource/org/load_testing_bitpipe.pdf: 4-5.

[5] Savoia A. The science and art of web site load testing[EB/OL]. http://www.51testing.com/ddimg/200704/THE_SCIENCE%20AND_ART_OF_WEB_SITE_LOAD_test.pdf: 3-5.

[6] 朱连章, 田超. 改进 Web 应用性能方法及性能测试分析[J]. 计算机工程与设计,2008, 29(7): 1817-1819.

[7] Woodside M, Hrischuk C. Automated performance modeling of software generated by a design environment[J]. Performance Evaluation, 45: 107-124.

[8] Surbraya B M, Subrahmanya S V. Design for Performance using PePPeR Model[EB/OL]. http://www.softwaredioxide.com/Channels/Content/Infosys-Design-Performance-PeppeR.pdf: 4-6.

[9] Distributed Computing, lnc. A New Model for Measuring Web Site Performance[EB/OL]. http://www.CapCal.com: 3-6.

[10] 牛霜霞, 龚永鑫. 性能测试进阶指南[M]. 北京: 电子工业出版社, 2009: 23-28.

[11] 赫建营, 晏海华. 一种有效的 Web 性能测试方法及其应用[J].计算机应用研究. 2008, 16(2): 260-275.

[12] 朱晶, 沈美明. Web 服务系统的性能分析与测试[J]. 计算机工程与应用. 2007, 17(6): 736-742.

[13] 袁才国. Web 性能测试研究及工具开发[D]. 长春：吉林大学, 硕士学位论文, 2006: 34-38

[14] 张广艳, 郑名扬. WebMark:一个 Web 服务器性能测试工具[J]. 软件学报, 2003, 14(7): 1318-1322

[15]张卫星. 基于 TTCN 的 WAE 性能测试研究[D]. 合肥: 中国科学技术大学, 硕士学位论文, 2003: 40-43.

[16]梁晟, 李明树. 一种模拟驱动的 Web 应用程序性能测试方法[J]. 计算机研究与发展, 2008, 40(7): 53-55.

[17]梁晟, 李明树. Web 应用程序响应时间的实验研究[J]. 计算机研究与发展, 2007, 40(7): 1076-1080.

[18]马琳, 罗铁坚等. Web 应用性能测试及优化[J]. 计算机应用, 2008, 24: 38-41.

[19]中国软件评测中心测试中心. 性能:软件测试的重中之重[J]. 中国计算机用户, 2006, 31: 42-43.

[20]迟瑞铮, 钟亦平, 张世永. 基于协议分析的自动化 Web 性能测试[J]. 计算机工程，2005, 31(7): 7-8

[21]苏波, 李克文. 基于 Web 应用的性能测试与优化[J]. 计算机工程与设计. 2007, 28(18): 11-15.

[22]邵燕琳. Web 系统性能测试工具的研究[D]. 呼和浩特: 内蒙古大学, 硕士学位论文, 2007: 13-18.

[23]芮素娟, 丁晓明. Web 应用性能测试进展[J]. 计算机科学. 2006, 33(8): 264-265.

[24]田超. Web Application 系统的测试技术研究与应用[D]. 东营：中国石油大学, 硕士学位论文, 2007: 10-15.

[25]卢琰琰, 吴海燕等. 基于联合压力测试的 Web 应用程序性能预测方法[J]. 计算机应用，2006, 26(6): 1472-1474.

[26]RFC2616: Hypertext Transfer Protocol-HTTP/1.1[S], http://www.ietf.org/rfc.

[27]尤克滨. UML 应用建模实践过程[M]. 北京: 机械工业出版社. 2009: 85-104.

[28]Wesley J. Chun 著, 宋吉广译. Python 核心编程[M]. 北京：人民邮电出版社. 2008: 170-198.

[29]Pilgrim, Mark. Dive into python[M]. New York: Springer-Verlag New York Inc: 240-260.

[30]Bruce Eckel 著, 陈昊鹏译. JAVA 编程思想[M]. 北京：机械工业出版社.2007:230-233