

性能测试的专业工具书，  
软件测试工程师的良师益友。

# 性能测试 进阶指南

## ——LoadRunner 11实战

51 Testing软件测试网 组编  
陈 霁 编著

## 内 容 简 介

本书是一本基于 HP LoadRunner 11 工具的指导用书,从性能测试原理到工具使用再到项目实施,全面介绍了性能测试的各个方面,其内容基本主线说明如下。

第一步(了解理论):磨刀不误砍柴工,打下基础;第二步(掌握工具):深入介绍 LoadRunner 11 工具三大部分(Virtual User Generator、Controller、Analysis)如何实现用户行为的模拟、性能指标的监控、负载的生成及后期的数据分析;第三步(项目实施):理论联系实际,介绍性能测试项目实施的流程和性能测试部门的组织管理;第四步(进阶提升):对一些当下流行的或比较特殊的协议和开发技巧通过真实案例进行介绍。

本书结合了很多工作中的实际案例,图文并茂,既适合渴望了解性能测试的新人,也适合对性能测试有一定认识和经验的中、高级测试工程师。同时,本书也可以作为高校开展性能测试课程的参考教材,让在校学生能对性能测试的本质和价值有一定的认识。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有,侵权必究。

## 图书在版编目(CIP)数据

性能测试进阶指南:LoadRunner 11 实战 / 陈霁编著; 51Testing 软件测试网组编.

北京:电子工业出版社,2012.6

(测试实践丛书)

ISBN 978-7-121-16739-3

I. ①性... II. ①陈... ②5... III. ①性能试验—软件工具, LoadRunner IV. ①TP311.56

中国版本图书馆 CIP 数据核字(2012)第 065650 号

策划编辑:李 冰

责任编辑:葛 娜

文字编辑:赵树刚

印 刷:北京东光印刷厂

装 订:三河市皇庄路通装订厂

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1092 1/16 印张:40.5 字数:1037 千字

印 次:2012 年 6 月第 1 次印刷

印 数:3500 册 定价:79.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zltts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

# 序

2012 年 1 月 8 日至 2 月 10 日，全球最大规模的“迁徙活动”浩浩荡荡地展开，[www.12306.cn](http://www.12306.cn) 成了全球最炙手可热的网站，每天高达数十亿次的点击量，平均刷新 500 次才能买到一张票。“一票难求”成了大量期盼回家过年的人最头疼的大事。

虽然铁道部一再想尽办法动足脑筋，可是网络购票难的问题始终没有解决。能从网上购买到一张回家的车票，真的是一件难如登天的事情。

为什么车票如此难买，网站如此难登？我觉得从本质上而言是设计者在最初乐观地估计了网站的访问流量，致使网站的性能指标完全无法应对真实的用户需求。简而言之一句话：性能测试没有达标。

这次春运网上购票暴露出来的问题再次向我们敲响了警钟，如何有效地开展性能测试，如何有效地将性能测试融入到百姓的生活中去，是众多企业、众多测试从业者毕生追求的目标。

作为一名测试工程师，性能测试也是我所关注和关心的问题。如何发掘系统潜在的性能瓶颈，如何准确地定位瓶颈的所在位置，如何进一步提高系统的性能指标，也是我在不断学习和提高的。

在学习的过程中，一本好的教材往往可以起到融会贯通、醍醐灌顶的作用。虽然古语有云“师傅领进门，修行靠个人”。可是领进门这个工作，真的不容小觑，少走冤枉路，往往可以起到事半功倍的作用。一些好的方法，一些有用的经验总结，一些具体的工程实践，可以帮助我们更好地深入理解性能测试的内涵，从而不断地完善自己。

本书就是这样一把“利器”，由浅入深，从理论联系到实际，一步步深入地向我们剖析了性能测试的内涵。工具的使用只是一个开始，真正地了解系统的结构、协议、数据库等各方面的内容，才能让我们更好地完成作为一名性能测试工程师应尽的职责，从而成长为一名优秀的性能测试工程师。

“序”话不多，马上展开我们精彩的性能测试之旅吧。



永远支持你的老婆  
沈蕾鸣

2012 年 1 月 28 日

# 前言

性能测试并不是一个很新的名词，但自从 LoadRunner 进入国内并流行起来，带动了性能测试的潮流，一时间测试人员的口边已经离不开这个 HP 的性能测试神器。如果你不会使用它进行性能测试，出门都不好意思和同行打招呼，似乎掌握了该工具就成为了测试的资深专家。可以说性能测试的流行从某些角度也反映了国内软件测试技术的进步，从过去的手工化功能测试逐步开始涉及通过编程的方式对系统进行功能测试，从而进一步地开展性能测试。

## 为什么写这本书

在游戏中身为一个 DPS 总是不断地在天赋、装备属性、输出手法、站位、技能优先级等问题上反复琢磨调优，力争成为该职业的高手。在不同的论坛会参考别人的推荐来选择 BIS（最优配置装备）、合适的天赋、最佳的重铸属性，熟悉和计算各种技能的优先级，在战斗的过程中动态监控各种 BUFF 和 DEBUFF 来让自己的核心技能能够在最合理的时间爆发并且获得最大利益。当每次实战时都能看到自己的 DPS 牢牢地处于榜首，那种感觉是妙不可言的。与职业玩家的互相交流，探讨心得可以更好地选择不同技能应对不同的战斗，性能测试及调优就这样在游戏中淋漓尽致地闪耀着光芒。当去指导朋友如何玩好这复杂的职业时，需要使用很多策略来让他了解游戏的逻辑本质，掌握技能的意义及优先级排列，并在战斗中合理应用。

而另一方面对于软件来说，性能测试、调优又是如此雷同而又复杂。看过很多性能测试方面的书，也在论坛上不断地解决各种问题，但总会发现新手问题反复发，高手问题无人解的问题，而企业中的各种问题也大同小异，仔细一看其实在很多书中都有介绍。在上一本《性能测试进阶指南——LoadRunner 9.1 实战》中介绍了 LoadRunner 9.1，后两年中并没有看到市面上有关 LoadRunner 11 的图书，而这两年中自己也有很多新的理解和对于部分章节描述的补充，在老婆的鼓励和支持下这本书就诞生了。限于作者的经验和水平，书中的不足和纰漏之处在所难免，恳请广大读者批评指正。

疑问和建议可以访问笔者的 BLOG：<http://www.51testing.com/?104>

## 本书面向的读者

本书并不是完全为初学者设计的，因为性能测试本来就不是一个新人能够涉及的内容，想要掌握性能测试需要多年的测试基础及广泛的知识面。这里需要读者具备一定的动态页面开发基础及 Linux 系统和数据库基本操作能力。

对于一个刚刚涉及性能测试的朋友来说，这本书可以解答你遇到的所有问题，虽然刚开始看会稍显吃力，但随着逐步的深入会越来越轻松。对于已入门的朋友，在简单地了解性能测试基



础及 LoadRunner 的实现原理后，本书的深度会让你进一步看透一些朦胧的东西。而对于高手来说，项目的实施策略、扩展工具的辅助及特殊的脚本开发可以帮助你进一步扩展思路。

相对于上一本《性能测试进阶指南》来说本书做了大量篇幅的修改，强化了很多细节的知识，将部分基础知识进行了精简。对于基础较差的入门级读者，建议先阅读笔者的上一本《性能测试进阶指南——LoadRunner 9.1 实战》以熟悉基础知识，从而降低阅读中的困难。

## 本书作者

陈霁，51Testing 论坛中 ID 为云层，集上海人和四川人的优缺点于一身，常常被朋友指责为典型处女座。自认公司内皮肤最白，经常被学员笑称“云版”或“棉花糖”老师，现任 51Testing 高级讲师，主攻性能测试。

2001 年至 2004 年在多家软件公司任职测试工程师，在底层摸爬滚打多年，对各种类型的软件测试都有所涉及。

2004 年至 2007 年在 Gameloft 上海部曾任测试经理、SQA 及 SCM。

2007 年中加入 51Testing 任职高级讲师，负责软件测试培训、咨询、企业应用解决方案等工作。

## 本书结构

本书分为 8 章、5 个，其内容如下表所示。

篇	章 节	内 容 介 绍
基 础 篇	第 1 章	性能测试基础，主要是为初学的朋友介绍简单的性能测试原理及相关知识
工 具 篇	第 2 章	LoadRunner 综述，全面介绍了 LoadRunner 工具的组成、安装及性能测试协议基础
	第 3 章	用户行为模拟，详细介绍了如何录制用户行为生成脚本并进行脚本开发，实现完美的用户行为模拟
	第 4 章	负载生成及监控 Controller，使用该工具实现场景设计及系统监控
	第 5 章	数据收集分析 Analysis，如何使用该工具对场景负载后的数据进行整理分析
实 战 篇	第 6 章	性能测试实战，通过对流行系统进行一次真实的性能测试，全面介绍在工作中如何进行性能测试的需求分析、测试设计、脚本开发、环境搭建及各种性能分析的方法
	第 7 章	性能测试组织，介绍企业中性能测试团队及版本控制自动化性能测试策略
	第 8 章	高级脚本开发，除了介绍常见的 AJAX、Windows Sockets 以外，还提供了基于开心网、Flex、Web Service、Sliverlight、Java Vuser、.Net Vuser、SMTP/POP 等比较特殊的开发案例
附 录	附录 A	常见 HTTP 请求返回简介
	附录 B	几款性能测试工具入门速成
	附录 C	常见 LoadRunner 问题索引
	附录 D	常见性能测试工具

附录 E	常用文档模板
附录 F	基于 XAMPP 测试环境搭建

## 从本书能收获什么

对于读者来说，通过本书的学习可以完全掌握 LoadRunner 11 的各种功能，并了解性能测试的实施过程，做到理论联系实际。本书中所有的例子都可以在本地或者权威网站进行实践，包括项目实践的所有内容也可在本地完成，从而真正实现自学成才。虽然本书是介绍如何使用 LoadRunner 这一性能测试工具来进行性能测试的，但是书中还穿插了大量的辅助工具，从而弥补了 LoadRunner 自身工具的不足；在实战阶段提供了详尽的性能需求分析方法、测试数据构造方式、性能测试分析方法及性能测试团队构建方式，让菜鸟迅速成长为高手。

## 致谢

测试做久了总习惯挑别人毛病或者具备一点完美主义精神，看了很多论坛上的提问和市面上的书籍后，我自信地认为能写本“比别人好点的书”。当开始执笔时，一本“完美”的书一直压迫得我喘不过气来，经过多次辗转，最终这个相对“完美”的版本没有胎死腹中，这里不得不感谢那些在这些年中和我互相学习分享知识的朋友。

对我来说最幸福的事情是在进入高中时拥有了一台小小的学习机，当第一次接触到键盘，第一次编写 BASIC 小程序时仿佛失了魂，按照现在的说法也就是“宅”了。这里要感谢我的父母，他们并没有阻止我沉迷于计算机之中，从而让我找到了自己的兴趣，并进一步将兴趣转化为了职业。

第一次接触软件测试是在 2001 年，作为一名刚毕业、自我感觉良好的大学生，我只身来到了上海。四处碰壁后“委身”于一家软件公司任职测试工程师，就这样误打误撞地进入软件测试行业，多亏当时面试我的原 IBM 测试经理田芳女士给了我这样的机会。“师傅领进门，修行靠个人”，待我明白这个道理并奋发努力后，在 Wang Sheng 和 Steven Ye 的栽培下我逐渐成为了独当一面的测试经理。

加入 51Testing 对我来说是个不小的转变，作为原 Etang 的 SQA 和 Gameloft 的测试经理，游戏测试及管理 and 软件测试还有一条不小的鸿沟，而性能测试以前并不是我的主攻方向，期间受到 Sincky、海龙、老朴、徐林林的细心指点，使我对性能测试的理解逐步加深，也逐渐接过了性能测试培训的教鞭。

在本书的编写过程中，宋锋、徐林林、田威峰、刘德宝、邓强等老师都在各个方面给予了我很多支持。为了给我更多的闭关写书的时间，小师妹和师弟不得不帮忙顶了很多课程，在此表示感谢。网上的很多朋友给了我很多知识点和参考案例，这里需要感谢一下 QQ 群中的 Fin、Mike、土匪、千里、友船、一个人、Yettie 等几大版主和热心网友帮助我完成了 Beta 版本的测试工作，并在校稿中做了很多的工作，包括内容的收集、评审和整理，在此表示感谢。

参与本书编写的人员有王威、王琰、朴春龙、邓强、周峰、周春江、徐林林、商莉、宋锋、宋光照、刘德宝、李波等。

最后还是要感谢一下亲爱的老婆，写书这段时间让你多费心了，这是一本为你而写的书。

陈 霁

# 目 录

## CONTENTS

# 性能测试进阶指南——LoadRunner 11 实战

## 基础篇

第 1 章 性能测试基础	1
1.1 性能测试工程师的标准及挑战	1
1.1.1 性能测试工程师的考评指标	1
1.1.2 性能测试工程师的挑战	3
1.2 性能测试基础	4
1.2.1 性能定义	4
1.2.2 性能指标	13
1.2.3 单机与网络性能测试	14
1.2.4 性能测试的流程	15
1.2.5 性能测试招聘要求	15
1.2.6 性能测试学习阶段	16
1.3 性能分析与调优	17
1.3.1 性能分析及调优原理	19
1.3.2 常见系统性能瓶颈	27
1.3.3 性能测试的注意事项	35
1.4 小结	36

## 工具篇

第 2 章 LoadRunner 综述	37
2.1 LoadRunner 简介	37
2.2 LoadRunner 工具组成	40
2.3 性能测试原理	40
2.4 自动化测试工具和性能测试工具的区别	42
2.5 协议分析	42
2.5.1 HTTP 详细介绍	43



# 目 录

## CONTENTS

## 性能测试进阶指南——LoadRunner 11 实战

2.5.2	HTTP 报文结构 .....	43
2.5.3	HTTP 请求 .....	44
2.5.4	HTTP 应答 .....	45
2.5.5	HTTP 捕获 .....	46
2.5.6	HTTP 回放 .....	48
2.6	安装 .....	49
2.6.1	在 Windows 下安装 LoadRunner .....	52
2.6.2	安装 Load Generator .....	53
2.6.3	附加组件 .....	56
2.6.4	LoadRunner License .....	57
2.7	LoadRunner 性能测试操作流程预览 .....	58
2.8	小结 .....	61
第 3 章 用户行为模拟 .....		62
3.1	VuGen 界面介绍 .....	63
3.1.1	Tree 图形化模式 .....	63
3.1.2	脚本模式 .....	65
3.1.3	Output Window .....	65
3.2	录制用户行为 .....	66
3.2.1	录制流程 .....	66
3.2.2	协议选择 .....	66
3.2.3	录制选项 .....	67
3.2.4	开始录制 .....	96
3.2.5	插入命令 .....	96
3.2.6	结束录制 .....	96
3.2.7	回放验证 .....	97
3.2.8	录制回放常见问题 .....	98
3.3	脚本开发 .....	99

## 性能测试进阶指南——LoadRunner 11 实战

3.3.1	Action 简介	99
3.3.2	脚本如何注释	100
3.3.3	语言规则	100
3.3.4	如何使用模板	101
3.3.5	脚本的导入导出	102
3.3.6	脚本调试	103
3.4	通用选项 ( General Options )	103
3.4.1	Parameterization 标签	104
3.4.2	Replay 标签	104
3.4.3	Environment 标签	105
3.4.4	Display 标签	106
3.4.5	Correlation 标签	107
3.5	运行设置 ( Run-Time Setting )	107
3.5.1	Run Logic	108
3.5.2	Pacing	114
3.5.3	Log	115
3.5.4	Think Time	116
3.5.5	Additional attributes	117
3.5.6	Miscellaneous	118
3.5.7	Speed Simulation	119
3.5.8	Browser Emulation	119
3.5.9	Preferences	120
3.5.10	Download Filters	121
3.5.11	Data Format Extensions	122
3.6	参数化	122
3.6.1	参数化的操作	123
3.6.2	Parameter List 界面介绍	127
3.6.3	VuGen 中的参数	129

# 目 录

## CONTENTS

## 性能测试进阶指南——LoadRunner 11 实战

3.6.4	Controller 中的参数 .....	135
3.6.5	同行取值 (Same line as) .....	140
3.6.6	参数类型 .....	141
3.6.7	数据向导 (Data Wizard) .....	146
3.6.8	参数和变量 .....	149
3.7	关联 .....	161
3.7.1	关联原理 .....	161
3.7.2	自动关联 .....	166
3.7.3	手动关联 .....	170
3.7.4	一边录制一边关联 .....	171
3.7.5	关联函数 web_reg_save_param_ex 详解 .....	172
3.7.6	关联函数 web_reg_save_param_regexp 详解 .....	183
3.7.7	关联函数 web_reg_save_param_xpath 详解 .....	185
3.7.8	关联函数的高级使用 .....	187
3.8	检查点 .....	196
3.8.1	文本检查点 .....	198
3.8.2	自动检查点 .....	199
3.8.3	图片检查点 .....	200
3.9	事务 .....	201
3.9.1	响应时间 .....	201
3.9.2	添加事务 .....	203
3.9.3	事务时间 .....	205
3.9.4	手工事务 .....	208
3.10	集合点 .....	211
3.11	小结 .....	213
第 4 章 负载生成及监控Controller .....		214
4.1	设计场景 .....	214

## 性能测试进阶指南——LoadRunner 11 实战

4.1.1 新建场景 .....	214
4.1.2 负载生成器管理 .....	225
4.1.3 用户管理 .....	228
4.1.4 运行设置 .....	228
4.1.5 IP 虚拟 .....	229
4.1.6 场景运行原理 .....	231
4.1.7 Service Level Agreement (服务品质保障) .....	233
4.2 系统监控 .....	236
4.2.1 Scenario Groups (场景用户状态) .....	236
4.2.2 Scenario Status (场景运行状态) .....	238
4.2.3 计数器原理 .....	238
4.2.4 计数器管理 .....	240
4.2.5 SiteScope .....	247
4.3 场景运行 .....	248
4.4 QTP 脚本在场景中的运行 .....	249
4.5 场景数据 .....	250
4.6 小结 .....	251
 第 5 章 数据收集分析 Analysis .....	 252
5.1 新建 Analysis 分析 .....	252
5.2 Analysis Summary .....	253
5.2.1 Analysis Summary (场景的摘要) .....	253
5.2.2 Statistics Summary (场景状态的统计说明) .....	253
5.2.3 5 Worst Transaction (SLA 失败事务) .....	254
5.2.4 Scenario Behavior Over Time (场景行为综述) .....	254
5.2.5 Transaction Summary (事务摘要) .....	254
5.2.6 HTTP Responses Summary (HTTP 响应摘要) .....	256
5.3 Graphs (数据图) .....	256

# 目 录

## CONTENTS

## 性能测试进阶指南——LoadRunner 11 实战

5.3.1	Vusers (虚拟用户状态)	258
5.3.2	Errors (错误统计)	259
5.3.3	Transactions (事务)	259
5.3.4	Web Resources (网页资源信息)	262
5.3.5	Web Page Diagnostics (网页分析)	264
5.3.6	Network Monitor (网络监控)	269
5.3.7	Resources (资源监控)	270
5.4	图设置与操作	280
5.4.1	Merge Graphs (合并图)	281
5.4.2	Auto Correlate (自动定位瓶颈)	283
5.5	Transaction Report (事务报告)	286
5.6	SLA Report (系统阈值监控报告)	286
5.7	External Monitor (外部监控数据导入)	287
5.8	Cross with result (跨脚本横向比较)	289
5.9	生成测试报告	289
5.9.1	New Report (新建报告)	289
5.9.2	Report Templates (报告模板)	291
5.9.3	HTML 报告	292
5.10	小结	292

## 实战篇

第 6 章	性能测试实战	294
-------	--------	-----

6.1	计划测试	295
6.1.1	分析系统阶段	295
6.1.2	定义测试目标	301
6.1.3	明确定义概念	321
6.1.4	编写性能测试计划	322



## 性能测试进阶指南——LoadRunner 11 实战

6.1.5 编写性能测试方案 .....	326
6.1.6 编写性能测试用例 .....	330
6.2 搭建测试环境 .....	331
6.2.1 测试平台评估 .....	331
6.2.2 数据生成 .....	332
6.2.3 测试环境搭建手册 .....	340
6.3 创建脚本 .....	347
6.3.1 用户注册 .....	347
6.3.2 用户查询 .....	349
6.3.3 用户看帖 .....	350
6.3.4 用户回帖 .....	351
6.4 创建场景 .....	359
6.4.1 场景设计 .....	360
6.4.2 负载监控 .....	361
6.5 运行场景 .....	367
6.5.1 场景运行 Checklist .....	367
6.5.2 场景运行记录 .....	368
6.6 分析性能数据 .....	369
6.6.1 性能调优原理 .....	369
6.6.2 前端性能分析 .....	375
6.6.3 后端性能分析 .....	382
6.7 性能测试报告 .....	404
6.7.1 平台对比性能测试报告 .....	406
6.7.2 Phpwind85 性能分析报告 .....	414
6.7.3 DiscuzX2 VS Phpwind85 性能对比报告 .....	436
6.7.4 Phpwind85 验收指标性能测试报告 .....	446
6.7.5 Phpwind85 压力测试报告 .....	450
6.8 小结 .....	454

# 目 录

## CONTENTS

## 性能测试进阶指南——LoadRunner 11 实战

第 7 章 性能测试组织	455
7.1 性能测试团队	455
7.2 性能测试流程分工	456
7.3 配置管理	457
7.4 性能测试自动化	463
7.5 小结	467
第 8 章 高级脚本开发	469
8.1 AJAX	469
8.1.1 使用 HTTP/HTML 模式开发 AJAX 脚本	472
8.1.2 使用 Click and Script 模式开发 AJAX 脚本	477
8.1.3 使用 TruClient 模式开发 AJAX 脚本	480
8.2 本地动态 JS Session	482
8.3 基于域权限的登录	492
8.4 Flex	494
8.5 Silverlight	508
8.6 Web Service	514
8.6.1 基于 WSDL 的调用	514
8.6.2 基于 SOAP 的调用	518
8.6.3 基于 HTTP 的调用	521
8.6.4 基于 Windows Sockets 的调用	524
8.6.5 扩展 Oracle 数据库性能测试	527
8.7 Windows Sockets	531
8.8 E-mail ( SMTP/POP3 )	539
8.9 FTP/HTTP 混合协议	545
8.10 .NET Vuser	549

## 性能测试进阶指南——LoadRunner 11 实战

8.10.1 使用.NET Vuser 测试 SQL Server 2008 数据库性能 .....	551
8.10.2 使用.NET Vuser 测试 C# 类库 .....	552
8.11 Java Vuser .....	554
8.11.1 使用 Java Vuser 测试 MySQL 数据库性能 .....	555
8.11.2 使用 Java Vuser 测试 JAR 包 .....	557
8.12 iPhone4 Vuser .....	558
8.13 小结 .....	562
附录A 常见HTTP请求返回简介 .....	563
附录B 几款性能测试工具入门速成 .....	566
B.1 VSTS2010 .....	566
B.2 Apache AB .....	570
B.3 WebBench .....	574
B.4 HTTP_Load .....	574
B.5 Siege .....	576
B.6 JMeter .....	576
附录C 常见LoadRunner问题索引 .....	584
附录D 常见性能测试工具 .....	596
D.1 应用性能测试工具 .....	596
D.1.1 商用 .....	596
D.1.2 开源 .....	597
D.2 系统监控 .....	598
D.3 硬件瓶颈定位 .....	599
D.4 白盒分析工具 .....	601

# 目 录

---

## CONTENTS

## 性能测试进阶指南——LoadRunner 11 实战

D.5 网络工具 .....	602
附录E 常用文档模板 .....	604
E.1 性能测试需求分析 .....	604
E.2 性能测试计划 .....	606
E.3 性能测试方案 .....	610
E.4 性能测试报告 .....	615
E.5 性能测试申请单 .....	616
E.6 性能测试脚本业务报告 .....	617
E.7 场景运行 checklist .....	618
附录F 基于XAMPP测试环境搭建 .....	619
F.1 搭建 XAMPP .....	619
F.2 搭建 PhpWind 测试环境 .....	622
索引 .....	624
参考资料 .....	630

## 1.3 性能分析与调优

在谈“性能分析与调优”这个话题前我们先来聊聊关于制动的故事。

制动是在任何运输工具上必备的一个重要功能，制动的性能关系到行驶安全。在我们的自行车上普遍使用了 V 刹，而在电动车上经常会使用碟刹（线碟、油碟）。那么 V 刹和碟刹性能上有什么区别呢？

### 1. 制动力方面

V 刹是刹的轮边，碟刹是刹的轴心。刹车位点到轴心的距离也就是刹车的动力臂长度，学过物理的人都知道动力臂越长，动力越大（杠杆原理），V 刹的位点在轮边上，而碟刹的位点几乎接近轴心，也就是说 V 刹的动力臂要比碟刹的动力臂长很多，也就意味着山地车在行驶时候，一个人以同样的力量去捏刹车，装 V 刹的车子要比装碟刹的车子刹车距离近，更容易停住。但有时由于 V 刹制动力过大，轮子抱死，会出现翘尾的情况。

对于线碟和油碟来说，线碟是使用钢丝来传导手刹的力量。油碟是用液压油传导手刹的力量，它们的工作原理基本相同。但是由于液压油在导油管内的阻力要远远小于钢丝在线管里的阻力，所以油碟对手刹力量的损耗要小于线碟的损耗。也就是说油碟的制动力要比线碟的制动力略强一点。

### 2. 耐热性方面

V 刹是用橡胶和金属摩擦制动，碟刹是用金属与金属摩擦制动。在温度低于 60℃ 的情况下，橡胶与金属摩擦的阻力大于金属与金属摩擦的阻力。而当温度超过 60℃ 时金属会由于热胀冷缩使表面变得凹凸不平（当然这是微观的，是肉眼看不见的），阻力会增大。橡胶则由于温度过高，橡胶变软，制动力下降。温度升高是由于刹车摩擦所致的，刹车时间越长，温度越高。也就是在短时间刹车时 V 刹优于碟刹，但在长时间刹车的时候碟刹优于 V 刹。

### 3. 结构方面

V 刹结构简单，可靠性高，容易维修。碟刹架构复杂，可靠性略低于 V 刹（因为碟刹内部情况不是像 V 刹那样可以看见），维修较难，非专业人员维修很可能越修越坏。特别是油碟，非专业的野蛮盲目维修会让油碟送命的。

### 4. 重量方面

这个我想大家都知道，V 刹比碟刹轻。

所以从性能角度来说自行车普遍选择 V 刹，而更快的燃气助动车甚至汽车应该选择使用碟刹。

在汽车上随着车辆的自重和速度增长，普通的人力机械助力和油碟被真空助力以及现在常用的液压助力方式取代，大大提升了刹车的可操作性和安全性。而为了避免刹车抱死的情况，ABS 防抱制动系统也从两轮发展到了四轮。

而到了火车，汽车所使用的碟刹系统已经完全不能满足其性能需要。一般来说火车都



在使用闸瓦制动，用铸铁或其他材料制成的瓦状制动块，在制动时抱紧车轮踏面，通过摩擦使车轮停止转动。在这一过程中，制动装置要将巨大的动能转变为热能消散于大气之中。而这种制动效果的好坏，却主要取决于摩擦热能的消散能力。使用这种制动方式时，闸瓦摩擦面积小，大部分热负荷由车轮来承担。列车速度越高，制动时车轮的热负荷也越大。如用铸铁闸瓦，温度可使闸瓦熔化，即使采用较先进的合成闸瓦，温度也会高达 400~450℃。当车轮踏面温度增高到一定程度时，就会使踏面磨耗、裂纹或剥离，既影响使用寿命也影响行车安全。可见，传统的踏面闸瓦制动适应不了高速列车的需要。于是一种新型的制动装置盘形制动应运而生。盘形制动是在车轴上或在车轮辐板侧面安装制动盘，用制动夹钳使以合成材料制成的两个闸片紧压制动盘侧面，通过摩擦产生制动力，使列车停止前进。由于作用力不在车轮踏面上，盘形制动可以大大减轻车轮踏面的热负荷和机械磨耗。另外制动平稳，几乎没有噪声。盘形制动的摩擦面积大，而且可以根据需要安装若干套，制动效果明显高于铸铁闸瓦，尤其适用于时速 120 千米以上的高速列车，这正是各国普遍采用盘形制动的原因为所在。但不足的是车轮踏面没有闸瓦的磨刮，将使轮轨黏着恶化；制动盘使簧下重量及冲击震动增大，运行中消耗牵引功率。

如果到了更快的飞机上我们的制动系统是什么？如果还是使用轮胎减速的方案是肯定不行了，这里需要使用一些空气动力学。飞机在落地后发动机反推系统会产生向前推动的空气流，产生主要减速动力。而机翼上的扰流板会上翻，气流压力集中在机翼上，产生向前的阻力快速降低飞机速度，以及向下的作用力加大机轮和跑道的摩擦，提高刹车效率。

到了更快一点的战斗机通常不会有客机这样大型的减速扰流板，所以着陆跑道会比客机更长，甚至会有减速伞在降落时使用。而特殊一点的航母舰载战斗机需要使用 ILS 着陆阻拦索减速，强行把飞机拉住。

如果再快一点在火箭上或者返回宇宙仓呢？光有一个大大的降落减速伞是不够的，通常还有小型的反推火箭系统来帮助减速着陆。

在上面这段制动发展历史中，我们可以看到分析和调优是贯穿其中的，无论是材料学、空气动力学、热学、机械学等都在影响着制动的性能，通过实验获得性能数据并分析得到瓶颈后，等待基础学科克服自身缺点给出新的解决方案，调优的效果才逐渐体现，这也是性能分析及调优为何十分复杂的一个原因。

对于软件来说当我们真的要要进行性能分析和调优时，又需要多少基础学科来支持呢？

### 1.3.1 性能分析及调优原理

性能测试的目的是评估当前系统性能指标，分析定位解决性能瓶颈，预防规避性能风险。性能分析是为了确定导致性能瓶颈的原因，而调优就是用来解决性能瓶颈。通过某些手段来让系统的性能得到提升是性能调优的主要目的。

性能分析主要有以下两种方法。

#### 1) 指标达成法

将测试结果与用户需求进行比较，如果达到用户需要则测试通过。

- 系统满足 10 万注册用户（其中活跃用户数为 1 万）访问。
- 系统处理能力：20 个注册/秒、45 个并发浏览/秒、30 个登录操作/秒。

- 服务器资源利用率在满负荷的情况下，忙时峰值 CPU 负载不超过 75%，内存占用不超过 80%。

例如，需要对一个参加 100 米跑的选手在比赛前进行性能测试，确保其能获得冠军，那么首先需要明确第一名所需要达到的性能目标（100 米短跑总时间），对其进行性能测试，当发现测试结果能够达到冠军所具备的条件后，性能测试即可结束。

## 2) 最优化分析法

通过分析并消除系统性能瓶颈，使系统的处理能力最大化，系统资源实现充分利用。

例如，需要对一个参加 100 米跑的选手进行技术指导，并不在乎他是否能够拿到冠军，而是重点强调能否提升自己的比赛成绩，那么就需要进行系统的训练和指导，如规范起跑动作、强化肌肉及协调性等，最终实现运动成绩的提高。

对应的性能调优方法也分为两大方向，如图 1.6 所示。

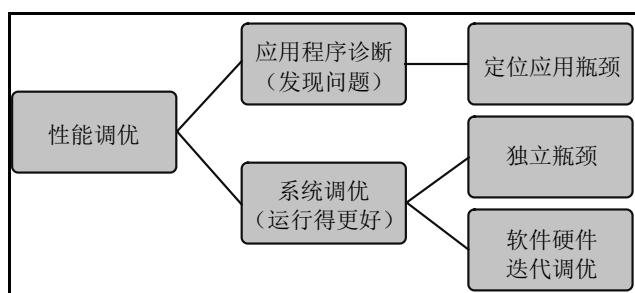


图 1.6 性能调优方向

### 1) 应用程序诊断

应用程序的诊断是性能测试的最初目的。通过模拟多用户操作形成负载，检验应用程序是否能够满足用户性能需求。如果不能满足，则定位应用瓶颈，并寻找解决该瓶颈的方案，确保系统在修正后能够满足用户需求。对于一个项目来说，一般都以应用诊断为主。

### 2) 系统调优

在性能调优中最基本的目标是为了满足用户，而进一步的要求是超越自己，这个时候需要做的是让系统能够比以前更加优秀地运行，通过生成负载，对测试结果进行分析，并且准备大量的软硬件环境进行迭代测试，找出影响性能的要害，最终提升系统的性能。一般产品都会采用系统调优的方式来逐步完善系统性能。

常见的性能瓶颈有如下一些情况。

#### 1) 硬件上的性能瓶颈

一般指的是 CPU、RAM 方面的问题，分为服务器硬件瓶颈、网络瓶颈（对局域网可以不考虑）、服务器操作系统瓶颈（参数配置）、中间件瓶颈（参数配置、数据库、Web 服务器等）、应用瓶颈（SQL 语句、数据库设计、业务逻辑、算法等）。

例如，确定了在数据库服务器上需要 6 个 CPU、12GB 内存。但是在测试时，发现 CPU 的持续利用率超过 95%，这时可以认为在硬件上出现了性能瓶颈。

#### 2) 应用软件上的性能瓶颈

一般指的是应用服务器、Web 服务器等应用软件，还包括数据库系统。

例如：在 WebLogic 平台上配置了 JDBC 连接池的参数，最大连接数为 50，最小连接

数为 5，增加量为 10。在测试时发现，当负载增加时，现有的连接数不足，系统会动态生成 10 个新的连接，导致交易处理的响应时间大大增加。这时可以认为在应用软件上出现了性能瓶颈。

### 3) 应用程序上的性能瓶颈

一般指的是开发人员新开发出来的应用程序。

例如，某程序员开发了一个缴费处理程序。在测试时发现，这个缴费处理程序在处理用户的并发缴费请求时，只能串行处理，无法并行处理，导致缴费交易的处理响应时间非常长，这时可以认为在应用程序上出现了性能瓶颈。

### 4) 操作系统上的性能瓶颈

一般指的是 Windows、UNIX、Linux 等操作系统。

例如，在 Windows 操作系统中，对某软件进行性能测试，出现物理内存不足时，如果虚拟内存设置也不合理，虚拟内存的交换效率就会大大降低，从而导致行为的响应时间大大增加。这时可以认为在操作系统上出现了性能瓶颈。

### 5) 网络设备上的性能瓶颈

一般指的是防火墙、动态负载均衡器、交换机等设备。

例如，在动态负载均衡器上设置了动态分发负载的机制，当发现某个应用服务器上的硬件资源已经到达极限时，动态负载均衡器将后续的交易请求发送到其他负载较轻的应用服务器上。在测试时发现，动态负载均衡机制没有起到相应的作用，这时可以认为在网络上出现了性能瓶颈。

性能瓶颈出现的原因及其定位是十分复杂的，这里只是简单介绍常见的几种瓶颈类型和特征，而性能测试所需要做的就是根据各种情况因素综合考虑，然后协助开发人员一起定位性能瓶颈。

## 1. 一般性能问题调优的步骤

### STEP 01 确定问题。

- 应用程序代码：在通常情况下，很多程序的性能问题都是写出来的，因此对于发现瓶颈的模块，应该首先检查一下代码。
- 数据库配置：经常引起整个系统运行缓慢，一些诸如 Oracle 的大型数据库都是需要 DBA 进行正确的参数调整才能投产的。
- 操作系统配置：不合理就可能引起系统瓶颈。
- 硬件设置：硬盘速度、内存大小等都是容易引起瓶颈的原因，因此这些都是分析的重点。
- 网络：网络负载过重导致网络冲突和网络延迟。

### STEP 02 确定原因。

当确定了问题之后，我们要明确这个问题影响的是响应时间吞吐量，还是其他问题？是多数用户还是少数用户遇到了问题？如果是少数用户，这几个用户与其他用户的操作有什么不同？系统资源监控的结果是否正常？CPU 的使用是否到达极限？I/O 情况如何？问题是否集中在某一类模块中？是客户端还是服务器出现问题？系统硬件配置是否够用？实际负载是否超过了系统的负载能力？是否未对系统进行优化？

通过这些分析及一些与系统相关的问题，可以对系统瓶颈有更深入的了解，进而分析

出真正的原因。

**STEP 03** 确定调整目标和解决方案。

提高系统吞吐量，缩短响应时间，更好地支持并发。

**STEP 04** 测试解决方案。

对通过解决方案调优后的系统进行基准测试。

**STEP 05** 分析调优结果。

系统调优是否达到或者超出了预定目标？系统是整体性能得到了改善，还是以牺牲某部分性能来解决其他问题。调优是否可以结束了？

最后，如果达到了预期目标，调优工作就基本可以结束了。

例如，在数据库中经常会出现查询响应时间较长的问题，解决这种 SQL 查询响应时间较长的问题通常会使用索引来解决。

什么是索引呢？

打个比方，你有很多的小抽屉，每个抽屉里面放一些杂物，假如你要找东西，最原始的方法就是一个个抽屉翻，这就是没有索引的情况。

假如聪明一点，给抽屉编号（唯一键），把哪个号码的抽屉有什么东西记录在纸上，找东西先看看这张纸，这就是普通索引，假如你要知道哪个抽屉有什么，你可以在纸上迅速找到抽屉号码（大家知道这是使用查找树），然后得到相关的信息，这种情况普通索引是很快的；但是要找到一个特定的东西哪些抽屉有，你就要把整张纸遍历一次，这就是 LIKE 查询，假如你要找哪些抽屉同时有两种甚至更多种物品，LIKE 就更加烦琐了。假如一个表有上千万的记录，大家可以想象查询的代价。

在索引中又分为聚集索引和非聚集索引两种索引模式。

### 1) 聚集索引

表中存储的数据按照索引的顺序存储，检索效率比普通索引高，索引占用硬盘存储空间小（1%左右），但对数据新增/修改/删除的速度影响比较大（降低）。

如下。

- 无索引，数据无序。
- 有索引，数据与索引同序。
- 数据会根据索引键的顺序重新排列数据。
- 一个表只能有一个索引。
- 叶节点的指针指向的数据也在同一位置存储。

TSQL 语法：create CLUSTERED INDEX idxempID ON emp(empID)

### 2) 非聚集索引

不影响表中的数据存储顺序，检索效率比聚集索引低，索引占用硬盘存储空间大（30%~40%），对数据新增/修改/删除的影响很少。特点如下。

- 一个表可以最多可以创建 249 个非聚集索引。
- 先建聚集索引才能创建非聚集索引。
- 非聚集索引数据与索引不同序。
- 数据与非聚集索引在不同位置。



- 非聚集索引在叶节点上存储，在叶节点上有一个“指针”直接指向要查询的数据区域。
- 数据不会根据非聚集索引键的顺序重新排列数据。

TSQL 语法: `create NONCLUSTERED INDEX idximpID ON emp(empID)`

对于索引有一些错误观点，具体如下。

- 主键就是聚集索引。
- 只要建立索引就能显著提高查询速度。
- 把所有需要提高查询速度的字段都加进聚集索引，以提高查询速度。

一般来说以下规则是正确的：

- 用聚集索引比用非聚集索引的主键速度快。
- 用聚集索引比用一般的主键做 `order by` 时速度快，特别是在小数据量情况下。
- 使用聚集索引内的时间段，搜索时间会按数据占整个数据表的百分比成比例减少，而无论聚集索引使用了多少个。
- “水可载舟，亦可覆舟”，索引也一样，不要过度使用索引。

对于 SQL 语句优化的基本原则如下。

- 使用索引来更快地遍历表。默认情况下建立的索引是非聚集索引，但有时它并不是最佳的。在非聚集索引下，数据在物理上随机存放在数据页上。合理的索引设计要建立在对各种查询的分析和预测上。一般来说：
  - 有大量重复值且经常有范围查询 (`between`, `>`, `<`, `>=`, `<=`) 和 `order by`、`group by` 发生的列，可考虑建立聚集索引。
  - 经常同时存取多列，且每列都含有重复值可考虑建立组合索引。
  - 组合索引要尽量使关键查询形成索引覆盖，其前导列一定是使用最频繁的列。
- `IS NULL` 与 `IS NOT NULL` 不能用 `NULL` 作为索引，任何包含 `NULL` 值的列都不会被包含在索引中。即使索引有多列这样的情况下，只要这些列中有一列含有 `NULL`，该列就会从索引中排除。也就是说如果某列存在空值，即使对该列建索引也不会提高性能。任何在 `WHERE` 子句中使用 `IS NULL` 或 `IS NOT NULL` 的语句优化器是不允许使用索引的。
- `IN` 和 `EXISTS`，`EXISTS` 要远比 `IN` 的效率高，里面关系到 `FULL TABLE SCAN` 和 `RANGE SCAN`，几乎将所有的 `IN` 操作符子查询改写为使用 `EXISTS` 的子查询。
- 在海量查询时尽量少用格式转换。
- 当在 SQL Server 2000 中，如果存储过程只有一个参数，并且是 `OUTPUT` 类型的，必须在调用这个存储过程的时候给这个参数一个初始的值，否则会出现调用错误。
- `ORDER BY` 和 `GROUP BY`：使用 `ORDER BY` 和 `GROUP BY` 短语，任何一种索引都有助于 `SELECT` 的性能提高。注意如果索引列中有 `NULL` 值，`Optimizer` 将无法优化。
- 任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等，查询时要尽可能将操作移至等号右边。
- `IN`、`OR` 子句常会使用工作表，使索引失效。如果不产生大量重复值，可以考虑把子句拆开，拆开的子句中应该包含索引。



- 谨慎使用游标。在某些必须使用游标的场合，可考虑将符合条件的数据行转入临时表中，再对临时表定义游标进行操作，这样可使性能得到明显提高。

注释：所谓的优化就是 WHERE 子句利用了索引，不可优化即发生了表扫描或额外开销。经验显示，SQL Server 性能的最大改进得益于逻辑的数据库设计、索引设计和查询设计方面。反过来说，最大的性能问题常常是由其中这些相同方面中的不足引起的。其实 SQL 优化的实质就是在结果正确的前提下，用优化器可以识别的语句，充分利用索引，减少表扫描的 I/O 次数，尽量避免表搜索的发生。

其实 SQL 的性能优化是一个复杂的过程，上述这些只是在应用层次的一种体现，深入研究还会涉及数据库层的资源配置、网络层的流量控制及操作系统层的总体设计。

对于某一条查询语句我们可以使用查询计划获得查询性能，来帮助我们确认如何编写性能更加优秀，举例如下。

- 仅在主键上建立聚集索引，并且不划分时间段

```
Select gid,fariqi,neibuyonghu,title  
from tgongwen
```

用时：128470 毫秒（128 秒）

- 在主键上建立聚集索引，在 fariqi 上建立非聚集索引

```
select gid,fariqi,neibuyonghu,title  
from Tgongwen  
where fariqi> dateadd(day,-90,getdate())
```

用时：53763 毫秒（54 秒）

- 将聚集索引建立在日期列（fariqi）上

```
select gid,fariqi,neibuyonghu,title  
from Tgongwen  
where fariqi> dateadd(day,-90,getdate())
```

用时：2423 毫秒（2 秒）

- 用聚集索引比用非聚集索引的主键速度快。下面是实例语句（都是提取 25 万条数据）。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where  
fariqi='2004-9-16'
```

使用时间：3326 毫秒

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where gid<=250000
```

使用时间：4470 毫秒

这里，用聚集索引比用非聚集索引的主键速度快了近 1/4。

- 用聚集索引比用一般的主键做 order by 时速度快，特别是在小数据量情况下。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen order by fariqi
```

使用时间：12936 毫秒

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen order by gid
```

使用时间：18843 毫秒

这里，用聚集索引比用一般的主键做 order by 时，速度快了 3/10。

事实上，如果数据量很小的话，用聚集索引作为排序列要比使用非聚集索引速度快得多。而数据量如果很大的话，如 10 万条以上，则二者的速度差别不明显。

- 使用聚集索引内的时间段，搜索时间会按数据占整个数据表的百分比成比例减少，而无论聚合索引使用了多少个。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where  
fariqi>'2004-1-1'
```

用时：6343 毫秒（提取 100 万条）

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where  
fariqi>'2004-6-6'
```

用时：3170 毫秒（提取 50 万条）

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen where  
fariqi='2004-9-16'
```

用时：3326 毫秒（与上句的结果一模一样。如果采集的数量一样，那么用大于号和等于号是一样的）

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen  
where fariqi>'2004-1-1' and fariqi<'2004-6-6'
```

用时：3280 毫秒

- 日期列不会因为分秒的输入而减慢查询速度

下面的例子中，共有 100 万条数据，2004 年 1 月 1 日以后的数据有 50 万条，但只有两个不同的日期，日期精确到日；之前有数据 50 万条，有 5000 个不同的日期，日期精确到秒。

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen  
where fariqi>'2004-1-1' order by fariqi
```

用时：6390 毫秒

```
select gid,fariqi,neibuyonghu,reader,title from Tgongwen  
where fariqi<'2004-1-1' order by fariqi
```

用时：6453 毫秒

## 2. 性能问题分析原则

- 原则 1：把事实与推测分开，总是用实际的证据来证明你的推测。
- 原则 2：在没有足够证据之前，不对程序进行优化。
- 原则 3：优先验证简单的假设。
- 原则 4：日志文件中没有错误并不代表真的没有错误。
- 原则 5：从系统到应用、从外到内进行层层剥离，缩小范围。确认是系统级问题还是应用级问题；确认是否为外部系统问题（如密码鉴权问题、EJB 问题等）；确认是应用程序问题还是数据库问题。
- 原则 6：范围缩小后，再分割成多个小单元，对每个小单元反复进行压力测试，来确定是哪个单元引起性能问题。

诊断性能问题，最常见的也是较难的判断的问题是：是应用程序还是数据库出了问题，还是两者都有？

这是因为应用程序、数据库、WebLogic Server (Tomcat) 都不是孤立运转的。因此脱离应用架构单独运行测试（如 SQL 计时、JDBC 计时、线程计时等）几乎没有作用。关键是对相互作用的了解。要熟知系统的性能度量方法，了解 SQL 的结构，了解用户发出的请求在跨越整个系统时的端对端/点对点计时、SQL 的计时等；了解用户发出请求后所关联的

线程、JDBC 连接、数据库的活动及其之间的交互关系。

应用数据库典型的三大类性能问题分析如下。

### 1) 过量的数据库调用

**问题：**常见的性能瓶颈来自过量的数据库调用，引发这些问题不一定是 SQL 查询的 Execute()或 Update()，而是应用程序与数据库的交互有关，例如，ResultSet 操作，常见的问题是指定了过于精细的查询条件，然后使用 ResultSet.Next()详细搜寻返回的数据。

**解决办法：**

从数据库中大量取得所要求的数据，避免应用程序反复回调数据库。

### 2) 数据库连接池问题

**问题 1：**连接池资源泄漏。

虽然可以通过 WebLogic 自带工具、Jprofiler 工具或自编工具检测到数据库连接池资源泄漏，但是很难在应用程序代码本身准确定位泄漏的源头。

**解决办法：**仔细分析程序代码，是否没有 close()连接？是否遗漏了 finally 块？或者尽管有 close()但并没有成功？

**问题 2：**连接池大小。

连接池过小会导致连接池满后，新的客户无法连接上系统，在日志中出现错误信息。一般的解决方法是增大连接池。但另一方面，连接池过大会造成资源无效损耗，可能会出现新的性能问题，那么连接池调到多大比较合适呢？

**解决办法：**

经验法则 1，数据库连接池数=线程池数×每个线程需要连接数据库的平均数×1.1（1.1 的含义是增加 10%的峰值期负载），通常每个线程需要连接数据库的平均数是 1，即当线程池数为 120 时，数据库连接池数就是 132。

经验法则 2，设置最初池大小=最大池大小。

### 3) SQL 语句及其索引或锁定属性问题

**问题：**SQL 语句及其索引或锁定属性不合理可能引发 DISKIO 过忙（磁盘读/写数据）或者 CPU 过忙（在内存中索引排序），造成执行时间过长，阻塞线程的执行，最终引发系统挂起或者执行超时引发系统挂起，例如，错误信息：

```
oracle.jdbc.driver.OracleStatement.doExecuteWithTimeout(OracleStatement.  
java:2857) at oracle.jdbc.driver.OraclePreparedStatement.executeUpdate
```

死锁引发系统挂起，例如错误信息：

```
java.sql.SQLException: ORA-00060: deadlock detected while waiting for  
resource at oracle.jdbc.dbaccess.DBError.throwSQLException(DBError.java:170)
```

**解决办法：**优化 SQL 语句及其索引或锁定属性。

## 3. 性能调优应该注意的要点

- 要点 1：在应用系统的设计开发过程中，应始终把性能放在考虑的范围内。
- 要点 2：确定清晰明确的性能目标是关键。
- 要点 3：必须保证调优后的程序运行正确。

- **要点 4:** 系统的性能更大程度上取决于良好的设计，调优技巧只是一个辅助手段。
- **要点 5:** 调优过程是迭代渐进的过程，每次调优的结果都要反馈到后续的代码开发中去。
- **要点 6:** 性能调优不能以牺牲代码的可读性和可维护性为代价。

性能调优是一个很复杂的过程，就好比一个人每天在上班路上需要消耗 1.5 小时，如何调优让这个上班路上开销的时间变小。性能调优需要有全面的知识和背景，需要对被调优对象有全面的了解和跟踪，才能逐步抽丝剥茧找到某块短板，将这块短板加长后，再找到下一块短板，依此类推。

很多公司在招聘性能测试工程师的时候都会提及性能调优这个职责，但是只靠一个性能测试专家是很难的。因为如果你想要对一个三层架构的网络应用进行调优，那么需要对网络架构、WWW 服务、APP 应用、DB 数据库都相当精通，才能完成所谓的调优工作，而这样的人已经可以胜任架构设计师的职位了，如果让他去做性能测试工程师是否有些大材小用了呢？

不过作为一个性能测试工程师，在成长的初期需要对性能测试调优的各个对象都有所了解，通过相关部门的人员协助完成调优工作，随后就是逐步加深，并且选择一个具体的方向进行深化并成为某一方面的性能测试及调优专家。

### 1.3.2 常见系统性能瓶颈

#### 1. 硬件瓶颈

任何系统的性能问题最终都会体现在硬件上，影响系统性能的主要硬件部件通常都是 CPU、内存、硬盘，而其中的性能瓶颈往往是在硬盘上。作为计算机硬件中发展最慢的设备，很多常见瓶颈都是由于硬盘的读/写速度慢导致的。提高硬盘读/写性能的方法无非是提高转速、提高单碟容量、增加缓存和更新接口，传统的温氏硬盘已经基本没有太大的调优空间了，因为从稳定性的角度 15 000 RPM（RPM 是 Revolutions Per Minute 的缩写，是转/每分钟。RPM 值越大，内部传输速率就越快，访问时间就越短，硬盘的整体性能也就越好）已经几乎是极限了，而提高单碟容量也存在一定的技术瓶颈，1TB 单碟的容量想要突破还需要比较长的时间。

对于现在的主流配置硬件来说，每秒钟十几 GB 的数据读取写入速度（内存）不在话下，而传统的温氏硬盘到现在也只能做到平均 120MB/s 的读取速度，这个速度还是针对大文件读/写的，而对于服务器（特别是 WWW 服务器）大量 4KB 小文件读/写速度，会惊人地下跌至 1MB 不到，而对应的 IOPS（每秒磁盘读/写次数）会低得可怜，大量的数据都在排队从硬盘上读取到内存中，再利用内存的超大带宽完成操作，这也是为什么内存大系统会运行比较快的原因。而一旦内存耗尽，数据不得不从硬盘上读取，这时系统的响应时间会大幅下降。在现在的很多系统中都会使用 Memory Cache 技术来解决小文件读/写的问题，但是这种技术存在一个比较致命的缺点就是一旦内存掉电，那么就意味着数据丢失，关键业务不能使用这种技术（例如，买入股票这样的操作不能写在内存中，否则一旦掉电，这笔业务就没了）。而除了 Memory Cache 技术以外，固态硬盘给了我们更多的选择，对于传统硬盘的读/写速度瓶颈及 IOPS，固态硬盘提供了全面的提升。



何为 IOPS?

**IOPS** (Input/Output Per Second) 即每秒的输入/输出量 (或读/写次数), 是衡量磁盘性能的主要指标之一。IOPS 是指单位时间内系统能处理的 I/O 请求数量, 一般以每秒处理的 I/O 请求数量为单位, I/O 请求通常为读/写数据操作请求。随机读/写频繁的应用, 如 OLTP (Online Transaction Processing), IOPS 是关键衡量指标。另一个重要指标是**数据吞吐量** (Throughput), 指单位时间内可以成功传输的数据数量。对于大量顺序读/写的应用, 如 VOD (Video On Demand), 则更关注吞吐量指标。

传统温氏磁盘本质上是一种机械装置, 如 FC, SAS, SATA 磁盘, 转速通常为 5400/7200/10K/15K RPM 不等。影响磁盘的关键因素是磁盘服务时间, 即磁盘完成一个 I/O 请求所花费的时间, 它由寻道时间、旋转延迟和数据传输时间三部分构成。

- **寻道时间 (Tseek)** 是指将读写磁头移动至正确的磁道上所需要的时间。寻道时间越短, I/O 操作越快, 目前磁盘的平均寻道时间一般在 3~15ms。
- **旋转延迟 (Trotation)** 是指盘片旋转将请求数据所在扇区移至读/写磁头下方所需要的时间。旋转延迟取决于磁盘转速, 通常使用磁盘旋转一周所需时间的 1/2 表示。比如, 7200 RPM 的磁盘平均旋转延迟大约为  $60 \times 1000 / 7200 / 2 = 4.17\text{ms}$ , 而转速为 15 000 RPM 的磁盘其平均旋转延迟约为 2ms。
- **数据传输时间 (Ttransfer)** 是指完成传输所请求的数据所需要的时间, 它取决于数据传输率, 其值等于数据大小除以数据传输速率。目前 IDE/ATA 能达到 133MB/s, SATA II 可达到 300MB/s 的接口数据传输速率, 数据传输时间通常远小于前两部分时间。

因此, 理论上可以计算出磁盘的最大 IOPS, 即  **$\text{IOPS} = 1000 \text{ ms} / (\text{Tseek} + \text{Trotation})$** , 忽略数据传输时间。假设磁盘平均物理寻道时间为 3ms, 磁盘转速为 7200/10K/15K RPM, 则磁盘 IOPS 理论最大值分别为

$$\text{IOPS} = 1000 / (3 + 60\,000/7200/2) \approx 140$$

$$\text{IOPS} = 1000 / (3 + 60\,000/10\,000/2) \approx 167$$

$$\text{IOPS} = 1000 / (3 + 60\,000/15\,000/2) = 200$$

固态硬盘 SSD 是一种电子装置, 避免了传统磁盘在寻道和旋转上的时间花费, 存储单元寻址开销大大降低, 因此 IOPS 可以非常高, 能够达到数万甚至数十万。实际测量中, IOPS 数值会受到很多因素的影响, 包括 I/O 负载特征 (读/写比例、顺序和随机、工作线程数、队列深度、数据记录大小)、系统配置、操作系统、磁盘驱动等。因此对比测量磁盘 IOPS 时, 必须在同样的测试基准下进行, 即使这样也会产生一定的随机不确定性。通常情况下, IOPS 可细分为如下几个指标:

- **Total IOPS**, 混合读/写和顺序随机 I/O 负载情况下的磁盘 IOPS, 这个与实际 I/O 情况最为相符, 大多数应用关注此指标。
- **Random Read IOPS**, 100%随机读负载情况下的 IOPS。
- **Random Write IOPS**, 100%随机写负载情况下的 IOPS。
- **Sequential Read IOPS**, 100%顺序读负载情况下的 IOPS。
- **Sequential Write IOPS**, 100%顺序写负载情况下的 IOPS。



IOPS 的测试 benchmark 工具主要有 **Iometer**、**oZone**、**FIO** 等，可以综合用于测试磁盘在不同情形下的 IOPS。对于应用系统，需要首先确定数据的负载特征，然后选择合适的 IOPS 指标进行测量和对比分析，据此选择合适的存储介质和软件系统。

如表 1.1 所示的磁盘 IOPS 数据来自 <http://en.wikipedia.org/wiki/IOPS>，给大家一个基本参考。

表 1.1 常见 SSD 磁盘 IOPS 指标

Device (设备名)	IOPS (每秒的输入/输出量)	Interface (接口类型)	Notes (备注)
Simple SLC SSD	~400 IOPS	SATA 3 Gb/s	
Intel X25-M G2(MLC)	~8600 IOPS	SATA 3 Gb/s	使用 Intel 测试数据文件在 160GB 型号上能够达到 8 600IOPS 80GB 型号能够达到 6,600IOPS 随机 4KB 读/写能够达到 35 000IOPS
Intel X25-E (SLC)	~5000 IOPS	SATA 3 Gb/s	3300 IOPS 及随机读/写 35 000 IOPS
G.Skill Phoenix Pro	~20000 IOPS	SATA 3 Gb/s	使用 SandForce-1200 主控芯片为基础的 SSD，通过更新固件后可以提升到 50 000 IOPS，但在基准测试时发现随机读为 25 000 IOPS，随机写 15 000 IOPS
OCZ Vertex 3	Up to 60000 IOPS	SATA 6 Gb/s	平均随机写 4K
Texas Memory Systems RamSan-20	120,000+ 随机读/写 IOPS	PCIe	启用缓存集
Fusion-io ioDrive	140,000 读 IOPS 135,000 写 IOPS	PCIe	
Virident SystemstachIOn	4KB 持续读 320000 IOPS 4KB 持续写 200000 IOPS	PCIe	
OCZ RevoDrive 3 X2	4K 随机写 200000 IOPS	PCIe	
Fusion-io ioDrive Duo	250000+ IOPS	PCIe	
Violin MemoryViolin 3200	250000+ 随机读/写 IOPS	PCIe /FC/Infiniband/iSCSI	使用快闪内存组
WhipTail, XLR8r 6TB	250000+ 读/写 IOPS	FC, iscsi, Infiniband, NAS	启用快闪存储组
DDRdrive X1,	512B 随机读 300000+IOPS 512B 随机写 200000+IOPS	PCIe	

Texas Memory Systems RamSan-720 Appliance	4KB 随机读/写 400000 IOPS	FC / InfiniBand	5000 MB/s, 高可靠性
OCZ Single SuperScale Z- Drive R4 PCI-Express SSD	Up to 500000 IOPS	PCIe	3800MB/s
Texas Memory Systems RamSan-630 Appliance	4KB 随机读/写 1,000000+ IOPS	FC / InfiniBand	10000 MB/s

续 表

Device (设备名)	IOPS (每秒的输入/输出量)	Interface (接口类型)	Notes (备注)
Fusion-io ioDrive Octal (single PCI Express card)	1 180 000+随机读/写 IOPS	PCIe	
OCZ 2x SuperScale Z-Drive R4 PCI-Express SSD	Up to 1 200 000 IOPS	PCIe	5 600MB/s
Texas Memory Systems RamSan-70	随机读/写 1 200 000IOPS	PCIe	包含内存 Cache 模块及扩展防掉电备份模块, 2 500 MB/s

在表 1.1 中我们可以看到虽然固态硬盘已经大幅提升 IOPS 了，但是高端策略还是要使用 Memory Cache 从本质上解决。而且固态硬盘还有数据损坏无法恢复，以及长时间使用会存在碎片导致读/写速度大幅下降的问题。

除了上面的物理解决方法以外我们还可以通过 RAID (Redundant Array of Inexpensive Disks, 独立磁盘冗余阵列) 技术来解决单硬盘的瓶颈。RAID 0 连续地分割数据并并行地读/写于多个磁盘上，因此具有很高的数据传输率，但 RAID 0 在提高性能的同时并没有提供数据可靠性，如果一个磁盘失效，将影响整个数据。所以一般我们通过 RAID0+1 来确保数据可靠性并享受 RAID 0 技术带来的磁盘性能提升。

那么，为什么固态硬盘无法做到内存的存取速度呢？这是因为 ROM 固态硬盘和 RAM 内存的实现原理不同导致的。

内存的发展速度已经到达了第五代 DDR 内存（一般使用在显卡上），而我们通常主板上使用的都是第三代 DDR 内存。内存的主要性能指标是在读/写带宽上，而影响带宽上的指标主要是内存通道数及内存频率。

现在常见的内存一般型号为 DDR3 1333MHz（工作在 1333MHz 的频率下），我们可以通过更换内存为更高频率及更低时序的方式来提升内存带宽。

何为时序呢？

内存时序是描述内存条性能的一种参数，一般存储在内存条的 SPD 中，由“A-B-C-D”这样的结构组成。一般“A-B-C-D”分别对应的参数是“CL-tRCD-tRP-tRAS”，它们的含义依次为：CAS Latency（简称 CL 值）内存 CAS 延迟时间，它是内存的重要参数之一，某些品牌的内存会把 CL 值印在内存条的标签上；RAS-to-CAS Delay (tRCD)，内存行地址传输到列地址的延迟时间；RAS Precharge Delay (tRP)，内存行地址选通脉冲预充电时间；Row Active Delay (tRAS)，内存行地址选通延迟。这是我们最关注的 4 项时序调节，在

大部分主板的 BIOS 中可以设定，内存模组厂商也有计划地推出了低于 JEDEC 认证标准的低延迟型超频内存模组，在同样频率的设定下，最低“2-2-2-5”这种序列时序的内存模组确实能够带来比“3-4-4-8”更高的内存性能。

内存频率就比较好理解了，现在“发烧”级别的内存频率可以做到 2400MHz，相对于 1333MHz 的默认频率几乎有了一倍的提升，而这种频率的提升可以换来带宽从 16GB 到 24GB 的提升。如果再能降低时序，那么结果会进一步提升，但是由于高频率内存一般都是超频获得的，本来就不太稳定，所以在这种情况下通常都对时序做了调慢处理，从而确保稳定性。

通过调整内存时序和频率我们可以大幅提升内存的处理能力，而另外一种提升策略就是通道，简单来说就是让多根内存并行和内存控制器进行交互，从而成倍地提升吞吐能力。从过去的单通道内存，到现在常见的双通道内存，再到 x58 主板提供的三通道内存，以及最新 x79 主板提供的四通道内存。我们在不断地调优内存的吞吐能力，但是实际情况并不是和我们理论上的情况一样。也就是说并不是四通道内存的吞吐量就是单通道内存的 4 倍，这是由于处理器和内存控制器已经无法有效地使用那么强大的带宽了（在未来出现更强的处理器和内存控制器时，四通道内存的优势才能真正发挥）。就现在的测试结果来说，双通道内存明显优于单通道，三通道内存和四通道内存接近双通道内存。

最终通过高频率的内存及多通道内存我们即可解决内存读/写的带宽问题，进一步配合超大的内存容量，我们即可实现大量操作的内存化，这样处理速度就可以得到非常大的提高。所以在很多设备上都会拥有内存，而其中 CPU 上的集成内存是速度最快的，从而确保 CPU 可以无须等待地从一级缓存中读/写数据。

在谈完硬盘和内存后我们来说一下 CPU 的常见性能指标。通常说到 CPU 我们第一反应的就是 CPU 工作频率，就是 CPU 的时钟频率，简单说是 CPU 运算时工作频率（1 秒内发生的同步脉冲数）的简称，单位是 Hz。它决定计算机的运行速度，随着计算机的发展，主频由过去 MHz 发展到了现在的 GHz（1GHz=1000MHz=1 000 000kHz=1 000 000 000Hz）。通常来讲，在同系列微处理器，其主频越高，速度也越快，但对于不同类型的处理器，它就只能作为一个参数来参考。另外 CPU 运算速度还要看 CPU 的流水线各方面的性能指标。由于主频并不直接代表运算速度，所以在一定情况下，很可能会出现主频较高的 CPU 实际运算速度较低的现象。因此主频仅仅是 CPU 性能表现的一个方面，而不代表 CPU 的整体性能。

说到处理器主频，就要提到与之密切相关的两个概念：倍频与外频，外频是 CPU 的基准频率，单位是 MHz。外频是 CPU 与主板之间同步运行的速度，而且目前的绝大部分计算机系统中外频也是内存与主板之间的同步运行速度，在这种方式下，可以理解为 CPU 的外频直接与内存相连通，实现两者间的同步运行状态；倍频即主频与外频之比的倍数。主频、外频、倍频，其关系式：主频=外频×倍频。早期的 CPU 并没有“倍频”这个概念，那时主频和系统总线的速度是一样的。随着技术的发展，CPU 速度越来越快，内存、硬盘等配件逐渐跟不上 CPU 的速度了，而倍频的出现解决了这个问题，它可使内存等部件仍然工作在相对较低的系统总线频率下，而 CPU 的主频可以通过倍频来无限提升（理论上）。我们可以把外频看做是机器内的一条生产线，而倍频则是生产线的条数，一台机器生产速度的快慢（主频）自然就是生产线的速度（外频）乘以生产线的条数（倍频）了。

所以我们经常可以通过提高外频降低倍频的方式来提升周边设备的处理能力，而并不提高 CPU 最终的工作频率，最终调优系统。但是只是依赖 CPU 的工作频率并不能解决我们的实际问题，所以各个厂商都推出了自己的特定指令集（例如 MMX、SSE4、EM64T、x86、RISC），通过对常见操作的底层支持大幅提升运算能力。我们现在使用的 CPU 基本上都是基于 x86 兼容指令集的，也可以这样简单认为我们使用的 CPU 都还不是 64 位指令集的，虽然可以在上面运行 x64 的系统，但是这里需要存在一个内部指令集转换，所以我们常用的 CPU 并不能完全发挥 64 位指令集的优势，但是通过兼容的方式提供了更多的内存寻址能力。

当我们提高了 CPU 的频率，优化了指令集后我们发现仍然无法解决 CPU 处理能力瓶颈的问题，而且随着 CPU 工艺变得越来越复杂，我们已经很难再进一步提升频率了（工艺、散热及电子迁移影响）。而大多数时候我们的问题还不是在 CPU 处理一件事情上慢，而是 CPU 拥有一个很大的任务队列，依次处理慢。所以现在我们会看到 CPU 在往多核化发展，通过在一个 CPU 中集成多个核心，每个核心再支持超线程技术，让应用程序可以更好地通过将任务分配给多个 CPU 上进行并行计算，从而解决性能问题。

除了上面介绍的硬盘、内存和 CPU 以外，接口的类型也可能成为系统的瓶颈。为了确保设备之间的读取速度，我们在不断地更新接口类型。最早我们使用的 USB1.0 设备属于低速设备，当我们通过这种接口的 U 盘来传输文件结果可想而知，而最新的 USB3.0 技术提供了 5Gb/s 全双工（USB2.0 则为 480Mb/s 半双工）带宽，几乎可以和 SATA6 的 6Gb/s 相媲美，这个时候我们会发现瓶颈重新回到了 U 盘自身。而对于高端的固态硬盘普通的 SATA6 都无法满足其强大的处理能力，往往会选择 PCI-E 这种高速接口来进行连接。

对于网络应用来说网络接口也比较容易出现瓶颈。我们常见的网络传输速率都是 100Mb/s，折合网络的理论值大约为 12MB/s，当我们在局域网上传输文件时，通常的瓶颈就是在网卡上（硬盘传输速度都在 100MB/s 以上）。如果升级到 1000Mb/s 时，网络理论传输速度可以达到 120MB/s 了，这个时候瓶颈会出现在发送数据的硬盘或者接收数据的硬盘上了。我们普通使用的双绞线一般为五类线，其提供了 100MHz 的传输频率，到了千兆位网络这种五类线已经不能满足实际要求，所以我们一般会升级选择超五类双绞线，虽然超五类非屏蔽双绞线也能提供高达 1000Mb/s 的传输速率，但实际使用中是无法达到的，由于我们使用的双绞线在屏蔽和加工工艺上的缺陷，导致超五类非屏蔽双绞线并不能发挥千兆位网络的最大速度，一般只能达到 80% 左右的使用率，带宽峰值约为 700Mb/s。所以在高端应用中我们会放弃使用传统的双绞线技术，而转用光纤技术，彻底解决传输材质带来的瓶颈。另一方面当我们使用万兆位交换机/路由器的时候，交换机/路由器是否能够处理如此大数量的多协议并进行路由转发会成为另外一个瓶颈。甚至在路由协议上也会存 EIGRP 和 OSPF 两种不同的寻路模式，提供更为优秀的路由路径选择方案。

对于硬件上的常见性能指标我们可以通过 AIDA64、SiSoftware、PC Mark、HD Tune Pro、Iometer、IoZone、FIO 等工具进行基准测试获得。

例如，我们使用 AIDA64 获得 CPU 上各种缓存的性能指标，如图 1.7 所示。



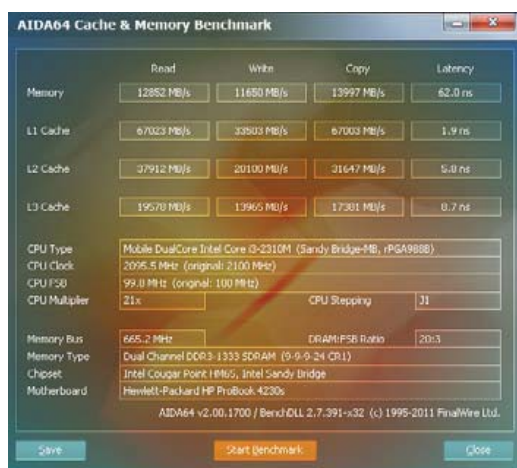


图 1.7 使用 AIDA 测试 Cache 和内存的性能指标

## 2. 操作系统

操作系统作为与硬件的交互层，它的性能决定了在操作系统下运行的应用的性能。对于操作系统的调优主要是在硬件支持和自身开销上体现的。

在 Windows Vista 中内存管理和文件系统方面引入了 SuperFetch 技术，可以把经常使用的程序预存入到内存，提高性能，此外你的后台程序不会夺取较高的运行等级了，不用担心突然一个后台程序运作导致你前台任务动弹不得。所以在使用 Vista 及 Win7 的时候会发现内存占用率相对以前的操作系统有所上升。这是一种更加合理使用空闲内存的策略，通过这种策略可以大幅提升我们访问常见应用的响应时间。除此以外一般新版本的操作系统都会对新硬件提供更好的支持，从而最大化发挥硬件本身的性能，例如在 Win7 中支持固态硬盘的 Trim 命令，并且会禁用 SuperFetch、ReadyBoost，以及启动和程序预取，这些功能都是针对传统硬盘设计的。

对于新版本的 Windows Server 系列操作系统，除了前面硬件支持以外也会提供对应组建的更新，从而实现性能的优化，例如，IIS 的版本升级。

除了操作系统的更新以外选择合理的操作系统平台也会极大地影响系统性能，我们常见的操作系统平台一般为 I386 和 x86-x64。I386 通常被用来作为对 Intel（英特尔）32 位微处理器的统称。而 x86-64 可在同一时间内处理 64 位的整数运算，并兼容 x86-32 架构。通常为了更好地支持大容量内存（32 位系统只能寻址 4GB 内存），我们都会选择 64 位系统，但是由于我们使用的 CPU 都是兼容模式，所以并不会非常明显地体现出 64 位系统在处理能力上的提升。而不同操作系统所能提供的 TCP/IP 连接上限也是不同的，这也是家用操作系统和服务器操作系统的区别之一。

所以从性能角度考虑在操作系统的选择上我们一般会尽可能选择最新版本，再选择 64 位系统完成对硬件的支持，但从功能、稳定性、可靠性角度考虑，作为服务器我们不应该选择太新、没有广泛使用的操作系统。这也是 2012 年伦敦奥运会会选择 Windows Vista 操作系统作为官方主要操作系统，而不是选择更好的 Win7 操作系统的原因。

而 Linux 或 UNIX 作为一个多用户多任务操作系统，强大、稳定的安全机制及高效客户端，使得选择它作为服务器操作系统成为更加主流的策略。而 Windows Server 系统为了弥补自己华丽桌面的开销也提供了只有 Power Shell 端的简化服务器 Core 核心版本，减小

桌面的资源开销。

对于操作系统的性能测试并没有有效的方法，只能通过在系统上进行硬件、软件配置测试来模拟对比。

### 3. 服务/框架

我们的代码通常都是运行在服务或者框架下的，例如，Java 语言就需要 JVM 环境作为底层框架才能运行，而 C# 需要 .NET 框架。而正是因为框架原因为了兼容平台导致了运行效率的低下，而 C 语言或者汇编语言由于自身可以直接与硬件交互，从而获得了超强的处理能力。

随着 Java 语言的 JVM 版本提升性能已经有了极大的提高。据 IBM 的统计数据，在同样的硬件上 2001 年时的 IBM JDK 版本的性能是 1996 年的 JDK 版本的 10 倍左右。而即使是在同一时期，不同公司的 JDK 和 JRE 的性能也不一样，比如 SUN、IBM、BEA 等公司都有自己开发的 JDK 和 JRE。而现在的 JVM 在采用了 Just-In-Time (JIT) 编译器后，程序的运行性能在很多 JVM 下可以与本地编译的程序一争高下，甚至在一些计算比较密集的数值计算领域也是这样。目前，Java 已经使用更先进的 HotSpot 技术来代替 JIT 技术，Java 的性能有了更进一步的提升。另外，在使用 -server 选项运行 Java 程序时，也可以对 Java 进行更深入的优化，比如在运行时将调用较多的方法内联 (inline) 到程序中来提高运行速度，这就是所谓的“动态优化”，而本地编译器是无法做到这一点的；这也是一些 Java 代码比对应 C/C++ 等语言编写的本地代码运行得更快的原因之一。微软的 .NET 平台也使用 JIT 编译器，所以也有类似问题。

而在服务平台我们一般分为表示层、业务逻辑层和数据访问层，每一层服务上又会有不同的性能区别。

表示层用于显示数据和接收用户输入的数据，为用户提供一种交互式操作的界面，通常是指我们的 Web 服务器，而最常用的就是 Apache。Apache 可以运行在几乎所有广泛使用的计算机平台上，由于其跨平台和安全性被广泛使用，是最流行的 Web 服务器端软件之一，但是 Apache 也有自己的缺点，所以在行业中会有针对性的 Lighttpd 和 Nginx 作为替代品出现。Lighttpd 是一个单线程的针对大量持续连接做出专门优化的 Web 服务器（这正是多数高流量网站和应用程序需要的），而 Nginx 对静态页面的支持相当出色。IIS 由于它只是为了支持微软的 ASP.NET 技术存在，并且包含表示层和应用层，所以相对来说没有多余的选择余地，只有等待微软自身版本更新来获得更好的性能。

业务逻辑层的关注点主要集中在业务规则的制定、业务流程的实现等与业务需求有关的系统设计，也就是说它是与系统所应对的领域 (Domain) 逻辑有关的，很多时候，也将业务逻辑层称为领域层。在业务逻辑层上我们会有比较多的服务选择，包括 PHP、JBoss、Tomcat、WebLogic、WebSphere 等。PHP 包括两种运行模式，即 CGI 模式或者 Apache 整合模式，其中 Apache 模式相对来说更为稳定，再通过 Zend 编译和运行优化可以进一步提升 25 倍的性能。而对于 Java 的运行效率来说 WebSphere > WebLogic > JBoss > Tomcat。

数据访问层的主要功能是负责数据库的访问，可以访问数据库系统、二进制文件、文本文档或 XML 文档。简单地说就是实现对数据表的 Select、Insert、Update、Delete 的操作。如果要加入 ORM 的元素，那么就会包括对象和数据表之间的 mapping，以及对象实体的持

久化。常见数据库服务的包括 Oracle、DB2、MS SQL Server、MySQL。从自身的性能角度考虑，效率为 DB2>Oracle>MS SQL Server>MySQL，但是这些数据库都是关系型数据库。这种数据库无法满足超巨型数据处理的要求（如微博关注），所以 NoSQL 非关系型数据库像满足极高读/写性能需求的 Key-Value 数据库：Redis、Tokyo Cabinet、Flare；满足海量存储需求和访问的面向文档的数据库：MongoDB、CouchDB；满足高可扩展性和可用性的面向分布式计算的数据库：Cassandra、Voldemort，这些 NoSQL 数据库在各种 SNS 网站中大量应用。

我们可以通过对框架或者服务的配置进行调整来让系统运行得更快，也可以通过负载均衡（Load Balancing）策略在现有网络结构之上，有效、透明地扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

对于框架或服务的性能测试我们可以通过编写某种负载的业务代码来测试获得。一般服务或框架设计厂商都会提供一定的基准数据。

#### 4. 代码

到了代码级别影响性能的只有算法结构了。在表示层上如何合理地使用 JS 及 CSS 可以改变用户渲染页面的开销，在代码中使用合理的语法规则和算法结构可以大幅提升系统运算处理能力，而在数据库中我们可以优化 SQL 语句，创建存储过程，设置索引来提升数据库操作性能。

对于代码的性能测试我们一般可以通过静态或动态代码分析软件来实现。在单元测试阶段可以通过 JunitPerf 这样的框架来帮助我们方法进行性能测试，而在系统测试阶段我们可以使用 DevPartner Studio 分析代码的执行开销，如图 1.8 所示。

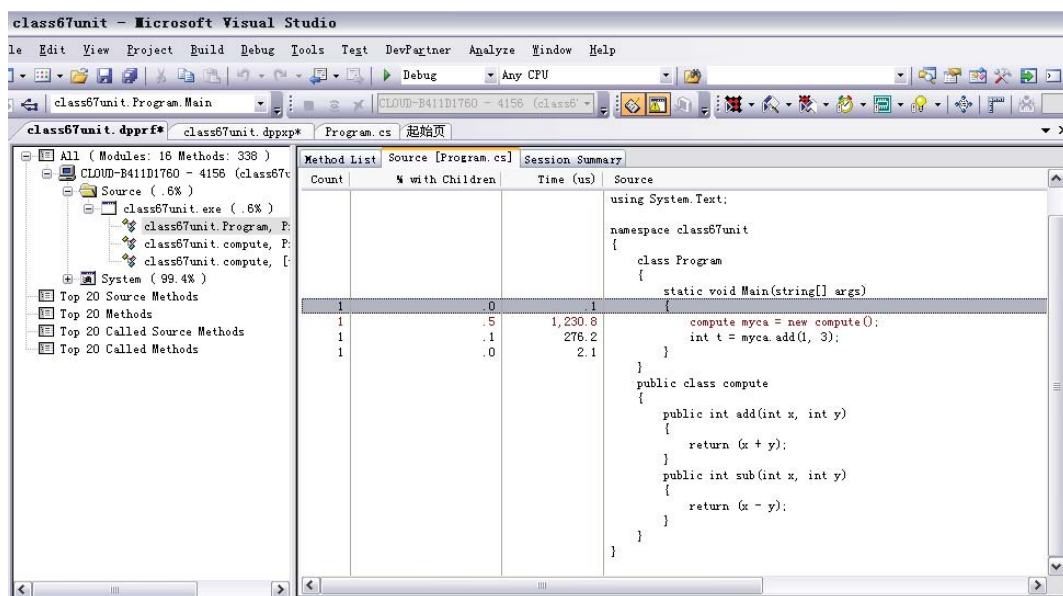


图 1.8 使用 DevPartner 分析定位脚本执行开销

对于代码级别常用的性能测试工具有 DevPartner Studio、Red Gate、Jtest、Jprobe、JunitPerf、Xdebug 等，这些软件可以帮我们对常见的 Java、C#、PHP 代码进行性能分析；而在数据库方面我们可以使用 SQL Expert、SQL Tuning Advisor STA、SQL Server Management Studio 中的执行计划、Toad、PL/SQL developer、Quest Central for Databases 等

工具来帮助我们对 SQL 语句执行分析。

### 1.3.3 性能测试的注意事项

#### 1. 性能测试应尽可能早地进行

与功能测试相同，性能测试进行得越早越容易发现并修复问题，当系统集成后，想要从众多模块中分析定位模块瓶颈是十分困难的，但是如果在项目初期就能对每个模块进行性能测试，问题自然会迎刃而解。

#### 2. 性能测试需要团队支持

质量是做出来的，而不是测出来的。性能也是同样，并不是有了一个性能测试部门，发现并定位了问题就能够提升软件的性能，性能的优化需要开发部门和相关部门的通力合作。

#### 3. 性能测试需要独立的测试环境

性能测试的测试环境相对功能测试有着更为严格的要求，需要独立的网络和硬件环境，来保证被测系统是独立可控的，甚至需要专门的管理员和流程来对被测环境进行控制。

#### 4. 测试前定义明确的测试目标

性能测试的执行成本较高，为了确保性能测试执行的有效性，在每一次性能测试前应明确本次性能测试的目标，并对这个目标进行监控和验证。

#### 5. 不要在服务器上进行性能测试

虽然服务器可以用来作为负载生成和被负载的对象，但是如果在服务器上进行这样的操作，系统资源会被负载消耗，导致得出的性能测试数据脱离实际情况。

#### 6. 创建的负载应该是模拟用户最常见、最密集的操作

在进行性能测试时，我们应该模拟用户最常使用的功能，来了解在这种操作下系统的资源消耗情况及用户体验。

#### 7. 在真正的性能测试前尽可能多地进行预测试

在性能测试前尽可能多地进行预测试，发现负载生成的结果及负载生成是否存在瓶颈，由于性能测试执行的成本较高，所以通过多次的预测试，可以降低最终测试的成本开销。

#### 8. 使用同一用户进行长时间大量操作是否存在内存泄漏或者类似的错误

通常这样做会发现系统某些功能设置上的问题。例如，当使用同一用户长期进行负载操作后，系统可能会出现线程崩溃。



### 3.7.5 关联函数 web\_reg\_save\_param\_ex 详解

上面说了常见的 3 种关联应用的方式，可以看到所谓关联都是使用 `web_reg_save_param_ex` 函数将服务器返回的内容进行收集过滤的过程，接着我们来仔细研究一下关联函数提供的选项。

首先介绍一个函数 `web_set_max_html_param_len`，当关联出错的时候 VuGen 都会提示以下内容：

```
Action.c(20): Error -26377: No match found for the requested parameter "WCSPParam2". Check whether the requested boundaries exist in the response data. Also, if the data you want to save exceeds 1024 bytes, use web_set_max_html_param_len to increase the parameter size [MsgId: MERR-26377]
```

很多朋友看到这个错误就会头皮发麻，完全不知所措。这种错误 99%都是由于关联的边界设置不合理导致没有关联到所需要的内容。系统提示使用 `web_set_max_html_param_len` 函数的目的是提醒如果被关联内容超出了默认的 1024 字节就会导致存放数据溢出，就会产生参数值为空、关联失败的情况（做附件下载关联或者多项关联大数据返回的脚本就可能会遇到这个问题），但通常都不会关联到如此巨大的内容。

`web_set_max_html_param_len` 函数可以自定义关联返回值存放的参数最大长度。打开 Insert/Add Step 窗口，找到对应的 `web_set_max_html_param_len` 函数，如图 3.124 所示。

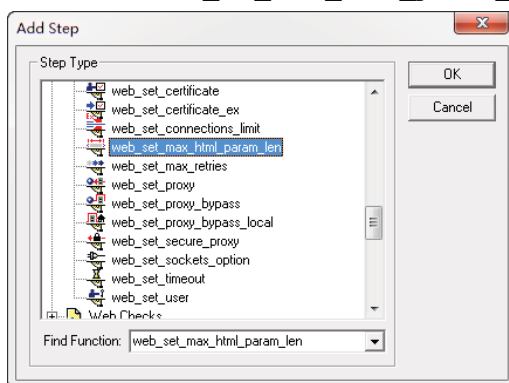


图 3.124 添加 `web_set_max_html_param_len` 函数

设置最大长度为 9999999，如图 3.125 所示。

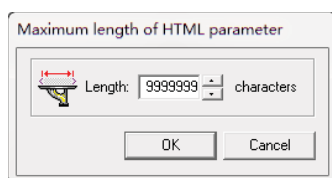


图 3.125 设置 `web_set_max_html_param_len` 函数长度

得到以下脚本：

```
web_set_max_html_param_len("9999999");
```

通过这个函数可以确保不会出现参数内容过长而无法存放的错误，不过这是以开销系统资源为代价的。

接着来看看 `web_reg_save_param_ex` 函数的选项，由于关联出来的内容存放在参数中，

所以还是建议打开日志中的 Parameter substitution 选项，以方便调试跟踪。

打开 Add Step 添加步骤，选择 web\_reg\_save\_param\_ex 函数，打开关联函数设置窗口，如图 3.126 所示。

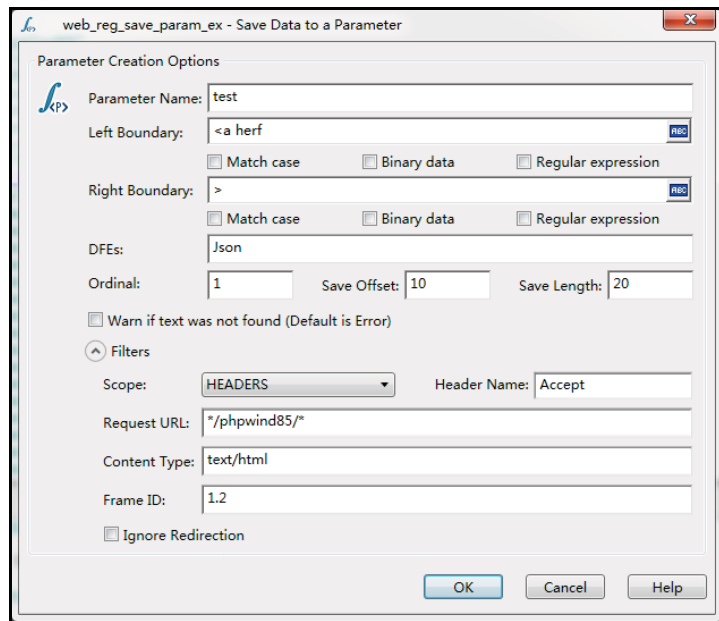


图 3.126 web\_reg\_save\_param\_ex 函数设置窗口

### 1. Parameter Name

此处设置存放参数的名称，关联出来的内容将会存放在该参数中。这里受到 Ordinal 选项的影响。

例如：

设置 Parameter Name 为 temp，当对应的 Ordinal 选项是任意一个数字的时候，只会关联一个匹配的记录，关联值将会存放在 temp 这个参数中。当 Ordinal 是 All 的时候，关联成功后的值将会依次存放在“temp\_数字”这样的参数数组中，并且还会添加一个 temp\_count 的参数存放关联出来的记录条数。

### 2. Left Boundary

此处设置左边界，这里是用来填写关联对于数据处理的左匹配内容规则。在左边界中存放的是一个字符串，例如，填写的内容为“左边界”会被转换成以下形式：

```
web_reg_save_param_ex(  
    "ParamName=test",  
    "LB=左边界",  
    "RB=",  
    SEARCH_FILTERS,  
    LAST);
```



#### 注意

如果输入的内容里面有双引号，那么需要通过转义符\来进行处理，例如：

```
web_reg_save_param_ex(  
    "ParamName=test",
```

```
"LB=\"左边界",  
"RB=",  
SEARCH_FILTERS,  
LAST);
```

### 3. Match case

默认情况下边界是 **Match case** 的，也就是检查大小写的，可以取消下面的选项来忽略大小写检查，会看到函数变为以下形式：

```
web_reg_save_param_ex(  
"ParamName=test",  
"LB/IC=左边界",  
"RB=",  
SEARCH_FILTERS,  
LAST);
```

### 4. Binary data

如果需要关联的内容是非 ASCII 字符的，那么需要使用该选项。选中该选项后可以看到函数变为以下形式：

```
web_reg_save_param_ex(  
"ParamName=test",  
"LB/BIN=\\x3F\\xDD",  
"RB=",  
SEARCH_FILTERS,  
LAST);
```

### 5. Regular expression

在 LR11 中关联函数提供了使用正则表达式的功能，但是在 LR11 Patch3 中取消了 `web_reg_save_param_ex` 函数对该功能的支持，而该功能由函数 `web_reg_save_param_regexp` 实现。

### 6. Right Boundary

此处设置右边界，这里是用来填写关联时对于数据处理的右匹配内容规则，选项同左边界。

### 7. DFEs

在录制选项和回放选项中我们提到过了 DFE 的功能，在关联这里也支持 DFE 的数据处理，我们先回到录制选项中的那个例子中，为其写一个普通关联看看返回（为了让返回结果看得更清楚，这里提前使用了 `Scope=BODY` 规则，该规则作用参考后面的 `Scope` 属性介绍）。

```
web_reg_save_param_ex(  
"ParamName=test",  
"LB=",  
"RB=",  
SEARCH_FILTERS,  
"Scope=BODY",  
LAST);
```

```
web_url("json.php",
"URL=http://localhost:8000/phpwind85/json.php",
LAST);
```

该代码运行后我们看到的返回结果是 JSON 数据:

```
Action.c(47): Notify: Saving Parameter "test = {"name":"Tom","age":18,"hobby":
"basketball"}".
```

接着我们使用关联的 DFE 功能, 在这里设置关联的 DFEs 格式为 JsonXml (这里的格式是指系统自带的 DFE 模块的 Tag 名, 参考图 3.35), 然后还要确保 Run-time settings 中的 DFE 功能启用, 代码变为:

```
web_reg_save_param_ex(
"ParamName=jsonresponse",
"LB=",
"RB=",
"DFEs=JsonXml",
"Ordinal=1",
SEARCH_FILTERS,
LAST);
web_url("json.php",
"URL=http://localhost:8000/phpwind85/json.php",
LAST);
```

代码运行后, 查看日志会发现 JSON 的数据格式变成了 XML 的数据格式了:

```
Action.c(47): Notify: Saving Parameter "jsonresponse = <HP_EXTENSION name=
"JsonXml"><object><pair name="name" type="str">Tom</pair><pair name="age" type=
"num">18</pair><pair name="hobby" type="str">basketball</pair></object></HP_
EXTENSION>".
```

现在是 XML 的数据格式了, 回忆在前面参数中介绍的对 XML 数据格式的处理函数, 你就知道接着可以干什么了。

DFE 是可以自行定义的, 除了系统自带的 5 种格式, 还能使用自行开发的格式。自定义 DFE 的代码写法参考安装目录为 HP\LoadRunner\samples\DataFormatExtension, 使用 VS2005 中的 VC++2005 进行开发。编译生成的 DLL 文件存放在安装目录下 HP\LoadRunner\bin 中, 配置文件存放在安装目录 HP\LoadRunner\DfeConfig\extensions 中。

## 8. Ordinal

这个关键字在很多函数里面都有应用, 在这里可以填写任意一个整数, 也可以填 All。如果填写数字, 那么说明从返回的记录中取出对应顺序的值, 而填写 All 的话将会返回所有的内容。

当使用 Ordinal =All 时, 关联函数会把所有匹配过滤策略的记录都抓出来, 由于参数只能存放一条记录, 所以关联函数会生成一个参数数组。被关联的记录会以{关联参数名\_关联 id}的形式生成参数列表, 并且在最后会有一个{关联参数名\_count}的参数来存放被关联到的记录条数。

例如, 上面写过的一个关联热搜关键字的例子, 代码如下所示:

```
web_reg_save_param_ex(
"ParamName=hotsearch",
```

```
"LB=<a href=\"searcher.php?keyword=\",
"RB=&type=thread\">",
SEARCH_FILTERS,
LAST);
```

当 Instance 设置为 All 时，代码变为：

```
web_reg_save_param_ex(
    "ParamName=hotsearch",
    "LB=<a href=\"searcher.php?keyword=\",
    "RB=&type=thread\">",
    "Ordinal=ALL",
    SEARCH_FILTERS,
    LAST);
```

运行代码后，关联将会返回所有匹配左右边界的内容，结果如下：

```
Action.c(20): Notify: Saving Parameter "hotsearch_1 = 结婚".
Action.c(20): Notify: Saving Parameter "hotsearch_2 = 母婴".
Action.c(20): Notify: Saving Parameter "hotsearch_3 = phpwind".
Action.c(20): Notify: Saving Parameter "hotsearch_4 = testing001".
Action.c(20): Notify: Saving Parameter "hotsearch_5 = 结婚".
Action.c(20): Notify: Saving Parameter "hotsearch_count = 5".
```

在使用 ALL 返回关联时，当关联的返回对象很长，就有可能导致参数长度溢出的问题。例如，我们关联一个板块中每一个帖子所在行的大多数信息，这里要关联的内容为：

```
<td class="subject" id="td_1882">

    <a href="read.php?tid=1882" name="readlink" id="a_ajax_1882" class=
"subject_t f14">testewt</a>&nbsp;
    </td>
    <td class="author"><a href="u.php?uid=1">admin</a><p>
2011-10-04</p></td>
    <td class="num"><em>0</em>/1</td>
    <td class="author"><a href="u.php?username=admin">admin</a><p>
<a href="read.php?tid=1882&page=e#a" title="2011-10-04 15:32">10-04 <span>
&raquo;</span></a></p></td>
```

这一段数据相对以前关联的内容要长很多，而且由于一个板块会有 20 个帖子，如果使用 ALL 关键字会返回 20 次这样长度的内容，代码如下：

```
web_reg_save_param_ex(
    "ParamName=test",
    "LB=<td class=\"subject\"\"",
    "RB=</a></p></td>",
    "Ordinal=ALL",
    SEARCH_FILTERS,
    LAST);
web_url("forum", "URL=http://localhost:8000/phpwind85/thread.php?fid=5", LAST);
```

当我们运行这样的关联时，会出现关联失败的情况。这个错误不是前面说到的关联边界设置导致关联失败错误，而是关联数据太长导致参数溢出而出现的关联失败错误。在这种情况下，我们就需要在关联前添加参数长度定义函数 `web_set_max_html_param_len("999999")` 来避免参数长度不够，无法将关联的结果保存，导致最终关联失败的情况。

## 9. Save Offset

设置关联的内容偏移量，从第几位开始进行关联操作。回到最开始的例子，我们抓取的是 `You have successfully installed XAMPP on this system!`，如果需要获得 `successfully installed XAMPP on this system!`这个字符串，则不用改变左边界，只需要设置 Save Offset 为 9 即可，代码为：

```
web_reg_save_param_ex(
    "ParamName=temp",
    "LB=Congratulations:<br>",
    "RB=</b><p>",
    "Ordinal=1",
    "SaveOffset=9",
    SEARCH_FILTERS,
    "ContentType=text/html",
    LAST);
```

对于一些定长度但是左边界比较难写的数据段用 Save Offset 可以很方便处理，不过在 LR11 中有了正则表达式这也不是什么问题了。

## 10. Save Length

关联出来的内容所需要保存的长度。在 Save Offset 的例子中我们写到如何获得 `successfully installed XAMPP on this system!`这个字符串，如果我们还希望获得这个字符串中的 `successfully installed XAMPP`，那么可以再添加 Save Length 为 22，代码变为：

```
web_reg_save_param_ex(
    "ParamName=temp",
    "LB=Congratulations:<br>",
    "RB=</b><p>",
    "Ordinal=1",
    "SaveOffset=9",
    "SaveLen=22",
    SEARCH_FILTERS,
    "ContentType=text/html",
    LAST);
```

通过 Save Length 和 Save Offset 的设置，我们就可以方便地抓取服务器返回的定长数据的任意一个部分了。

关联可以调整偏移量和长度，那么参数能做到吗？当然可以，如果需要对一个参数值进行偏移和长度设置，则需要使用 `lr_save_var` 函数，例如，下面的代码：

```
lr_save_string("I come from shanghai","city");
lr_save_var(lr_eval_string("{city}"),6,0,"result");
//从 city 这个参数中取 6 位长度的内容保存到 result 参数中
lr_save_var(lr_eval_string("{city}")+7,4,0,"result");
//从 city 这个参数的第 7 位开始取 4 个长度的内容保存到 result 参数中
```

可以看到运行的结果是：

```
Action.c(3): Notify: Saving Parameter "city = I come from shanghai"
Action.c(4): Notify: Saving Parameter "result = I come"
Action.c(5): Notify: Saving Parameter "result = from"
```

## 11. Warn if text was not found (Default is Error)

如果关联的对象不存在，又该如何进行处理呢？默认值为 **Error**，默认情况下如果没有关联到任何内容则提示错误。

```
Action.c(27): Error -26377: No match found for the requested parameter "temp".
Check whether the requested boundaries exist in the response data. Also, if the
data you want to save exceeds 256 bytes, use web_set_max_html_param_len to
increase the parameter size [MsgId: MERR-26377]
```

而选择该项，则只会简单提示没有抓到内容，不会产生错误。

```
Action.c(28): Warning -26377: No match found for the requested parameter
"temp". Check whether the requested boundaries exist in the response data. Also,
if the data you want to save exceeds 256 bytes, use web_set_max_html_param_len
to increase the parameter size [MsgId: MWAR-26377]
```

在很多时候关联不到内容也是正常的。例如，需要关联一个板块中的帖子编号，就存在没有帖子的情况。为了确保关联的正确性，在某些情况下可以选中该选项，出现的提示方式为 **WARNING**，并且为了确保没有帖子时不会对后面的操作产生影响，需要编写一个 **if** 语句进行判断，如下所示：

```
Action()
{
web_reg_save_param_ex(
    "ParamName=tid",
    "LB=<a href=\"read.php?tid=",
    "RB=&page=",
    SEARCH_FILTERS,
    LAST);
web_url("topiclist","URL=http://localhost:8000/phpwind85/thread.php?fid=
2",LAST);
web_url("read","URL=http://localhost:8000/phpwind85/read.php?tid={tid}",
LAST);
return 0;
}
```

上面这种写法是平常最通常的写法，接着我们添加分支判断，代码如下：

```
web_reg_save_param_ex(
    "ParamName=tid",
    "LB=<a href=\"read.php?tid=",
```



```

        "RB=&page=",
        "NotFound=warning",
        SEARCH_FILTERS,
        LAST);
web_url("topiclist","URL=http://localhost:8000/phpwind85/thread.php?fid=
64",LAST);
if (strcmp(lr_eval_string("{tid}"),"")==0)
{
    lr_output_message("there is no any topicid in the forum");
}
else
{
    web_url("read","URL=http://localhost:8000/phpwind85/read.php?tid={tid}",
LAST);
}

```

当我们添加了分支判断后，当板块没有帖子时，不会出现错误了，也不会去错误地访问后面的阅读帖子操作，转向执行信息输出。

如果关联选项 Ord 为 All 时，这里 if 语句检查的对象应该修改为 tid\_count 参数，代码变为：

```

web_reg_save_param_ex(
    "ParamName=tid",
    "LB=<a href=\"read.php?tid=",
    "RB=&page=",
    "NotFound=warning",
    "Ordinal=ALL",
    SEARCH_FILTERS,
    LAST);

web_url("topiclist","URL=http://localhost:8000/phpwind85/thread.php?fid=
2",LAST);

if (atoi(lr_eval_string("{tid_count}"))==0)
{
    lr_output_message("there is no any topicid in the forum");
}
else
{
    lr_save_string(lr_paramarr_random("tid"),"rtid");
    web_url("read","URL=http://localhost:8000/phpwind85/read.php?tid={rtid}"
, LAST);
}

```

这个代码可以实现在板块中随机选帖子，如果没帖子则不做操作。

## 12. Filters

下面的选项都是帮助关联返回限定的，通过这些设置可以进一步减少返回的范围。



### 13. Scope

该项设置关联查询的范围，在 LR11 中和以前的范围做了一些调整，应该算是更加强大了，这里 Scope 提供了 4 个选项：Body、Headers、Cookies、All。

#### 1 ) ALL

比较容易理解，就是让服务器的返回所有内容作为需要关联的目标来处理。

#### 2 ) Headers/Body/Cookies

这 3 个选项都是从请求返回的所有内容进行关联处理，包括图片、JavaScript 脚本等。区别在于对返回信息的分隔方式。在前面介绍 HTTP 的时候介绍过 HTTP 返回的内容其实是由 Header（HTTP 信息头）和 Body（HTTP 内容）组成的，而 Cookie 又是 Header 中的一部分，在 Tree 模式下的 HTTP View 视图可以清晰地看到 LR 如何定义各块内容。当我们对 Phpwind 登录返回做关联时，不同的 Scope 带来的效果如下。

#### 3 ) Header

指所关联的内容是所有服务器返回请求的 HTTP 头部分内容。可以通过查看服务器返回内容来了解，Body 之前的内容都属于 Header：

```
Action.c(16): Notify: Saving Parameter "temp = HTTP/1.1 200 OK\r\nDate: Wed,
05 Oct 2011 04:53:51 GMT\r\nServer: Apache/2.2.17 (Win32) mod_ssl/2.2.17
OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1\r\nX-Powered-By:
PHP/5.3.5\r\nSet-Cookie:
53d97_lastvisit=0%091317790431%09%2Fphpwind85%2Flogin.php; expires=Thu,
04-Oct-2012 04:53:51 GMT; path=/\r\nSet-Cookie: 53d97_lastpos=other; expires=
Thu, 04-Oct-2012 04:53:51 GMT; path=/\r\nSet-Cookie: 53d97_lastvisit=
0%091317790431%09%2Fphpwind85%2Flogin.php; expires=Thu, 04-Oct-2012 04:53:51
GMT; path=/\r\nSet-Cookie: 53d97_winduser=VgEDATtTUVZRVQ8HUFcEVg4JUFQBAQF
VVLRVUgBXUVIJAQ8AVT4; path=/; httponly\r\nSet-Cookie: 53d97_ck_info=%2F%09;
expires=Thu, 04-Oct-2012 04:53:51 GMT; path=/\r\nSet-Cookie: 53d97_lastvisit
=deleted; expires=Tue, 05-Oct-2010 04:53:50 GMT; path=/\r\nSet-Cookie: 53d97
regactivate=deleted; expires=Tue, 05-Oct-2010 04:53:50 GMT; path=/\r\nContent
-Encoding: gzip\r\nContent-Length: 7217\r\nKeep-Alive: timeout=5, max=100\r\
nConnection: Keep-Alive\r\nContent-Type: text/html\r\n\r\n".
```

在 Header 中还能进行二次过滤，填写 Header Name，默认值为 Undefined。如果我只想知道服务器返回 Header 中的 Server 服务器信息，那么在 Header Name 中填写 Server 即可：

```
Action.c(16): Notify: Saving Parameter "temp = Apache/2.2.17 (Win32)
mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1".
```

#### 4 ) Body

就是服务器返回在 Body 以后的内容：

```
Action.c(16): Notify: Saving Parameter "temp = <!doctype html>\r\n
<html>\r\n<head>\r\n<meta charset="utf-8" />\r\n<title>鎖悞ず淇℃炖 - Powered by
phpwind</title>\r\n<meta http-equiv="refresh" content="1;url=
http://localhost:8000/phpwind85/index.php">\r\n<SCRIPT type="text/javascript"
language="JavaScript" src="js/core/core.js"></SCRIPT>
...略
```

## 5 ) Cookie

指 Header 部分关于 Cookie 定义的部分内容。

```
Action.c(16): Notify: Saving Parameter "temp = 53d97_lastvisit=0%091317790561%09%2Fphpwind85%2Flogin.php; expires=Thu, 04-Oct-2012 04:56:01 GMT; path=/".
```

## 14. Request URL

这里提供了针对 URL 地址的过滤方式来减少关联范围，例如，我们可以填写\*.php 来说明只对 PHP 页面进行过滤。

## 15. Content Type

回顾我们在讲录制下载操作的时候，提到了 Content Type 这个概念，这样在录制的时候可以通过这个特性来过滤录制对象。而在关联这里，这个属性的效果是相同的，在网站应用中，我们要关联的内容一般都存放在 HTML 页面中，所以这里我们通常都是用 text/html 来作为 Content Type 过滤规则的。

## 16. Frame ID

这个选项是专门针对框架结构的网站设计的，有些时候需要关联的内容是在某个框架中的，这个时候就需要说明所关联的页面是框架中的哪一个了。比如在前面关联的处理上就可以使用这个属性，对比下面两个代码的关联返回：

```
web_reg_save_param_ex(  
    "ParamName=temp",  
    "LB=",  
    "RB=",  
    SEARCH_FILTERS,  
    "Scope=BODY",  
    "RelFrameID=1.2",  
    LAST);  
web_url("localhost:8000",  
    "URL=http://localhost:8000/",  
    LAST);  
  
web_reg_save_param_ex(  
    "ParamName=temp",  
    "LB=",  
    "RB=",  
    SEARCH_FILTERS,  
    "Scope=BODY",  
    "RelFrameID=1.3",  
    LAST);  
web_url("localhost:8000",  
    "URL=http://localhost:8000/",
```

LAST);

关联返回的内容是不同的,如果仔细观察返回代码会发现前一个关联返回的是左侧导航的内容,第二个关联返回的是右侧正文。

这里的 1.2 说明 XAMPP 网站是基于在第一个大的框架中的,所以如果想读取左下侧的页面,这个框架所属的编号就为 1.2,右侧框架编号为 1.3。

框架上栏 1.1	
左框架下栏 1.2	右框架下栏 1.3

某些复杂情况会出现多层框架嵌套问题,那么继续编号向下就可以了,例如:

框架上栏 1.1	
框架下栏 1.2	
下栏中的左栏 1.2.1	下栏中的右栏 1.2.2

通过 Frame ID 可以帮助我们在框架中细化关联数据范围,当某些框架中有多匹配边界的内容时用这个属性会更方便一些。

### 17. Ignore Redirection

在某些情况下系统会使用 HTTP 3xx 的重定向操作来完成页面跳转,该选项是用来忽略跳转页面信息的。如果选中该选项,通过这种重定向技术的页面将不会被关联。

在 LR11 中除了对 web\_reg\_save\_param 加强为 web\_reg\_save\_param\_ex,还提供了另外两个非常好用的函数 web\_reg\_save\_param\_regexp 和 web\_reg\_save\_param\_xpath。

### 3.7.6 关联函数 web\_reg\_save\_param\_regexp 详解

在介绍这个函数前先来处理一种情况。前面关联的左右边界都是静态的,如果左右边界是动态的,并且系统返回的 id 是不定长度的,那么如何使用关联函数将该 id 取出呢?这个问题在现实情况中会经常遇到,仅仅通过一个关联函数是无法处理的,这个时候还需要使用一个函数 strtok()来进行字符内容切割。

strtok()函数的作用是通过某个分隔符来切分内容的。

例如:

```
char city[1000];
char * token;
//这个函数是扩展的声明
extern char * strtok(char * string, const char * delimiters );
```

```
strcpy(city, "this is shanghai!");
token = (char *)strtok(city, " ");
lr_error_message(token);
token = (char *)strtok(NULL, " ");
lr_error_message(token);
token = (char *)strtok(NULL, " ");
lr_error_message(token);
```

通过这个函数可以得到 3 个字符：this、is、shanghai。通过空格来分隔字符串，可以得到第一个符合该条件的内容，如果需要继续分隔就使用 `strtok(NULL, " ");` 语句。如果关联出来的内容 `sessionid` 是变动长度的，如 `"sessionid=54321123&action=work"`，则如何获得这个变动长度的 `sessionid` 呢？使用下面的代码即可解决：

```
char temp[100];
char * token;
extern char * strtok(char * string, const char * delimiters );
lr_save_string("sessionid=54321123&action=work", "param");
strcpy(temp, lr_eval_string("{param}")); //取出参数值，并且赋值给变量 temp
token = (char *)strtok(temp, "&"); //使用&符号作为分隔符
```

这个时候 `token="sessionid=54321123"`，并且是根据 `&` 符号分隔的，所以 `id` 的长度可以任意变化，而 `token` 中的 `sessionid` 可以通过关联的时候 `Save Offset` 进行处理，或者使用 `strtok()` 函数对等号再次进行处理。

而在 LR11 中提供了 `web_reg_save_param_regexp` 正则表达式关联，上面的写法也可以退休了。打开 `Add Step` 添加步骤，选择 `web_reg_save_param_regexp` 函数，打开设置窗口，如图 3.127 所示。

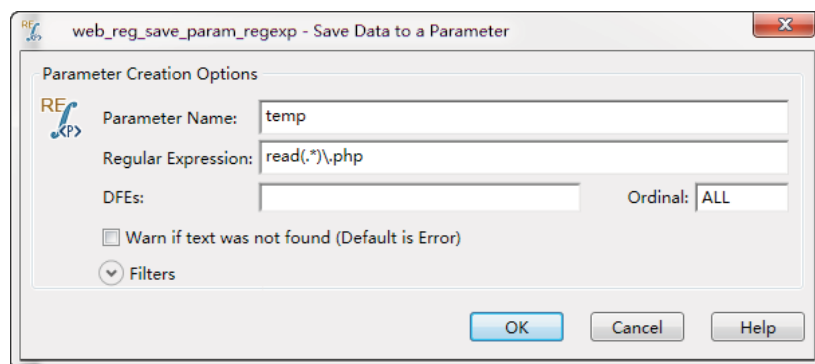


图 3.127 添加 `web_reg_save_param_regexp` 函数

在这个函数中关键就是在 `Regular Expression` 的写法上，在前面 XML 参数的 `lr_xml_find` 函数中我们提到过正则表达式的写法，在这里的写法唯一区别在于需要关联返回的内容需要用 `()` 圆括号标记。例如，这里的 `read(.*?)\.php` 就是指所有符合 `read` 开头 `.php` 结尾中间的任何内容都关联保存到参数 `temp` 中，这里的 `\` 是转义符，确保 `.` 能够正确地当做普通字符来匹配。

在 `Phpwind` 中如果我们要关联一个没有被回复过的帖子的发帖人是谁，这个在以前是比较难于实现的，我们先来看看一个帖子的 `HTML` 代码：

```

<tr class="tr3">
    <td class="icon tac"><input type="checkbox" autocomplete="off"
name="tidarray[]" id="tid_1884 value="1884" onclick="postManage.show
('postbatch','a_ajax_1884')" /><a title=" 开放主题 " href="read.php?tid=1884"
target="_blank"></a>
    </td>

    <td class="subject" id="td_1884">

        <a href="read.php?tid=1884" name="readlink" id="a_ajax_1884" class=
"subject_t_f14">需要使用正则表达式关联的例子</a>&nbsp;
        
    </td>
    <td class="author"><a href="u.php?uid=1">admin</a><p>2011-10-05
</p></td>

    <td class="num"><em>0</em>/2</td>
    <td class="author"><a href="u.php?username=admin">admin</a><p>
<a href="read.php?tid=1884&page=e#a" title="2011-10-05 13:43">4 秒前 <span>
&raquo;</span></a></p></td>
</tr>

```

在这个代码中我们需要关联的正文是 **admin**，验证的部分在于 `<em>0</em>`，这里的 0 代表没有回复，后面的 2 代表两次阅读。关联的难度在于如果用回帖数作为左边界那么右边界中的阅读数是动态数据。如果用回帖数作为右边界那么左边界中的发帖时间和用户 **uid** 是动态数据，导致这个关联在以前的写法中很难实现，必须要扩展关联后使用 **strtok** 来分离。现在使用正则表达式关联这个问题就非常简单了，代码如下：

```

web_reg_save_param_regexp(
    "ParamName=temp",
    "RegExp=<em>0</em>.*\r\n.*username=(.*)\">.*</a><p>",
    "Ordinal=1",
    SEARCH_FILTERS,
    LAST);

```

这里的过滤方式是使用 `<em>0</em>` 作为左边界然后拼接任意内容接回车换行符，再接任意字符至 **username=** 处，关联这串内容的右侧到 `>` 位置后面接任意字符，再接 `</a><p>`。通过这个关联就可以得到未回帖的发帖人名了，另外一种使用回帖数为 0 作为右边界的写法为：

```

web_reg_save_param_regexp(
    "ParamName=temp",
    "RegExp=uid=.*\">(.*)</a><p>.*\r\n.*<em>0</em>",
    "Ordinal=1",
    SEARCH_FILTERS,
    LAST);

```

刚开始写的时候大家会困惑在正则表达式的编写上，多多尝试（注意 `\r\n` 回车符、换行符及空格这是开始最难处理的东西），并且合理应用常见的正则表达式验证工具，就可以逐渐上手，成为你关联应用时的神器。

如果想要获得一个没有回复帖子的帖子编号，正则表达式为：

```
web_reg_save_param_regexp(
    "ParamName=topicid",
    "RegExp=ajax_(.*)\" class.*\r\n.*\r\n.*\r\n.*<em>0</em>",
    "Ordinal=1",
    SEARCH_FILTERS,
    LAST);
```

换成 `strtok` 的写法那么就要这样写了：

```
char tokstr[2000];
extern char * strtok(char * string, const char * delimiters );
char * token;
web_reg_save_param("string",
    "LB=a_ajax_",
    "RB=<em>0</em>",
    "Ord=1",
    "Search=NoResource",
    LAST);
//请求部分略
strcpy(tokstr,lr_eval_string("{string}"));
token = (char *)strtok(tokstr,"\"");
lr_output_message(token); //输出处理后得到的帖子编号
```

### 3.7.7 关联函数 `web_reg_save_param_xpath` 详解

如果大家用过一些自动化工具可能会对 Xpath 比较熟悉。Xpath 可以通过路径的方式访问到 XML、HTML 的任意节点位置，在关联里也可以使用这个技术来帮我们查找需要的元素。

打开 Add Step 添加步骤，选择 `web_reg_save_param_xpath` 函数，打开设置窗口，如图 3.128 所示。

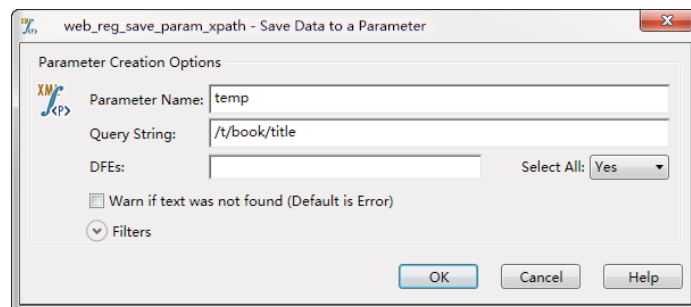


图 3.128 添加 `web_reg_save_param_xpath` 函数

在这里需要为 Query String 编写对应的 Xpath 查询语法，这里填写的 `/t/book/title` 是指一个 XML 格式中的结构。通过这个关联我们可以从：

```
<?xml version="1.0"?>
<t><book><author>cloud</author></book><book><author>cloudB</author></book></t>
```

这样的服务器返回中得到以下结果：

```
Action.c(14): Notify: Saving Parameter "temp_1 = A".
Action.c(14): Notify: Saving Parameter "temp_2 = B".
Action.c(14): Notify: Saving Parameter "temp_count = 2".
```



对于一些比较复杂的数据格式，那么怎么编写 Xpath 呢？这里使用 FireBug 来帮助我们，首先安装 Firefox 浏览器并且安装 FireBug 插件，接着在打开的页面中点击右下角的 FireBug 图标，切出该插件，如图 3.129 所示。

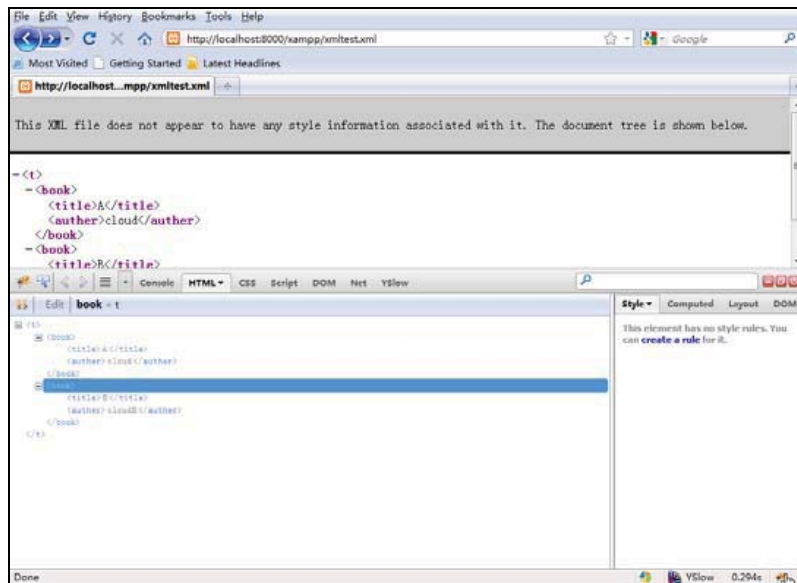


图 3.129 在 Firefox 中启动 FireBug

接着在浏览页面中找到自己想要的内容，通过右键菜单中的 Inspect Element 将这个元素定位，如图 3.130 所示。

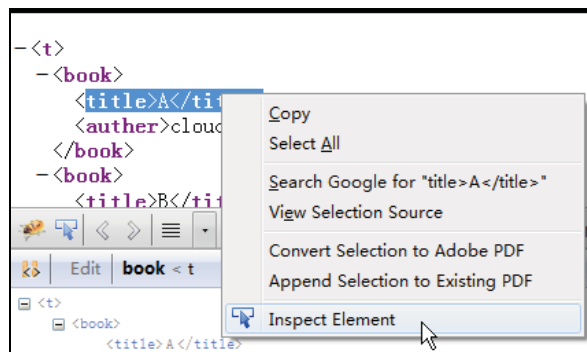


图 3.130 在 Firebug 中定位元素

接着将鼠标放到上面的工具条中，会看到对应的 Xpath 层次已经显示出来了，如图 3.131 所示。



图 3.131 在 Firebug 中观察 Xpath



这里可以通过右键菜单复制当前的 XPath 字符串，也可以在下面更加准确选择，如图 3.132 所示。

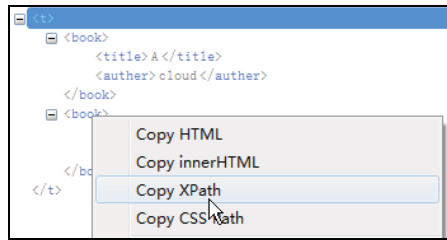


图 3.132 在 Firebug 中拷贝 XPath

通过这种方式不但可以得到 XML 的任意位置 XPath 写法，还能获得 HTML 的任意位置 XPath。在得到 XPath 后就可以直接复制到关联函数 `web_reg_save_param_xpath` 中了。但是在 LR11 中该关联函数只对 XML 数据格式有用，对于 HTML 格式无法使用 XPath 进行定位关联，所以，在处理 HTML 内容时还是推荐使用前面的两个关联函数来处理。

关于 XPath 的更多写法参考官方文档 <http://www.w3.org/TR/xpath/>。

## 6.2 搭建测试环境

在编写脚本的同时，执行场景之前需要完成测试环境的搭建工作，这里包括硬件和软件环境的搭建。根据性能测试计划中的测试环境规划，完成对整个测试环境的搭建（请协调开发人员及系统管理员等相关角色帮忙完成环境搭建的工作）。

### 6.2.1 测试平台评估

由于性能测试的特殊性，整个测试环境需要在严格的独立监控下管理，避免不受控的情况出现导致性能测试数据的偏差（类似于药品制造中的无菌室概念），而另一方面，在实际应用中很难得到真实的系统环境来完成性能测试。很多时候不可能在研发阶段就在公司搭建出和用户上线时完全相同的测试平台，那么得出来的数据就与真实数据有一定的差距，解决这个问题一般有以下两种解决方式（预估方式均有误差）。

#### 1. 通过建模的方式实现低端硬件对高端硬件的模拟

通过配置测试来计算不同配置下的硬件性能和系统处理能力的关系，从而推导出满足系统性能的真实配置情况，这种模拟需要精确的建模，模型的采样点越多，那么得到的结果越精确，从而将在低端配置下的性能指标通过该模型转化为高端配置下的最终预估性能指标。

例如，搭建一个低端环境，首先对这个环境的 CPU 或内存进行单独的性能基准测试。例如，使用 EVEREST 工具计算一下这个 CPU 的得分，再通过性能负载工具得到当前系统的 TPS 及响应时间。然后超频或者更换 CPU，再次计算相关分数，逐渐明确各种硬件资源对负载结果的影响关系，最后可以得到表 6.9。

表 6.9 硬件性能建模表

硬件配置	基准分数	TPS	响应时间
INTEL P4 1.8GB	1800	2	1.8s
INTEL P4 2GB	2000	2.2	1.8s
INTEL P4 3GB	3000	3	1.8s

通过这个表，可以简单认为每 900 基准分数约等于 1 个 TPS，而如果系统的最终需求是能够满足 20 个 TPS 的话，也就是说硬件配置的 CPU 至少需要达到 18000 分，反过来就可以知道服务器应该需要使用什么样的 CPU。

#### 2. 通过群集的方式计算

对于较大的系统来说，单台服务器的处理能力是有限的，通常都会采用群集的方式进行负载均衡，完成对海量请求的处理。虽然无法获得整个群集的测试环境，但是可以对群集上的一个节点进行性能测试，得出该节点的处理能力，再计算每增加一个节点的性能损失，同样也可以通过建模的方式得到大型负载均衡情况下的预估性能指标。

例如，首先在单台服务器上获得具体的性能指标，每台服务器能够承受 2000 人在线、

平均 TPS 为 80、响应时间为 2 秒。接着，添加负载均衡策略，再次测试负载策略下的数据损耗。得出数据后添加 1 台负载均衡服务器，测试在两台服务器下每台服务器的性能指标，依此类推，可以得到如表 6.10 所示的数据。

表 6.10 服务器负载性能建模表

负载服务器个数	平均每台负载服务器在线用户数	平均 TPS	响应时间
1	2000	80	2
1（添加负载均衡策略）	1950	78	2
2（使用负载均衡）	1950	78	2
3（使用负载均衡）	1900	76	2
4（使用负载均衡）	1900	75	2

随着负载均衡服务器的添加，平均每台服务器的处理能力会逐渐稳定，从而了解在什么样的情况下需要多少台负载均衡服务器。

受到资源限制，在这里搭建一个简易的测试环境，由一台配置较高的 PC 作为服务器，而另外一台笔记本作为负载生成来模拟用户行为，完成性能测试。除了搭建硬件环境外，还需要对软件环境进行搭建，其中最重要的是环境的备份。在进行性能测试时，为了保证测试结果数据的客观公正，在每次测试前，都需要保证相同的软件环境，所以对测试环境的备份工作是非常重要的。建议使用 Ghost 对测试环境进行镜像，每次测试后还原被测环境重启系统，避免由于磁盘碎片和前一次测试的缓存影响测试结果。本次测试由于使用了 VMWare 作为虚拟环境，通过其自带的 Snapshot 快照功能实现测试环境的备份还原。

对于整个测试环境的搭建，建议生成专门的文档进行管理，并进行配置管理，确保对测试环境做到基线控制。

6.2.2 数据生成

对于性能测试环境的搭建，还有一个麻烦的问题，就是性能测试数据如何生成？也就是大家经常遇到的容量问题。在性能测试方案中存在着容量为 500 万条论坛帖子，3 万注册会员的数据要求，如何快速有效地生成这些容量呢？

某些公司会将历史业务数据通过导入的方式实现容量的模拟，但是这种做法存在着几个问题：

- 某些数据内容敏感，是否会涉及信息泄露。
- 历史业务数据和新系统的结构不同，导出困难。

这样做不行，难道自己再编写个 LoadRunner 脚本专门来发那么多帖子么？500 万个帖子，如果用 LoadRunner 来进行数据生成效率太低，这里可以考虑使用编写 SQL 代码或者专门的数据生成工具来实现容量的生成。

通过编写一个存储过程来完成对数据的随机生成，只需要修改中间的 Insert 命令即可完成表格的适应，以及对生成记录条数的控制。这里以 T-SQL 为例，该存储过程可以实现随机生成 10~20 位大小写混合的字符串，最终通过 insertrandstr 存储过程，实现记录的添加。

```
--生成随机数据
```

```
if exists (select * from sysobjects where name = 'createsinglechar' and type
= 'p')
drop proc createsinglechar
go
if exists (select * from sysobjects where name = 'createsingleip' and type
= 'p')
drop proc createsingleip
go
if exists (select * from sysobjects where name = 'createwholestr' and type
= 'p')
drop proc createwholestr
go
if exists (select * from sysobjects where name = 'insertrandstr' and type =
'p')
drop proc insertrandstr
go
use [projcet]
go

create procedure createsingleip
@randip int output
as
declare @singleip int
set @singleip = cast(rand()*256 as int)
--print @singleip
while (@singleip>255 or @singleip<0)
begin
set @singleip = cast(rand()*256 as int)
--print @singleip
end
set @randip = @singleip
--print @randip
go

create procedure createsinglechar
@randchar varchar(3) output
as
declare @singlechar varchar(10)
set @singlechar = cast(rand()*123 as int)
while (@singlechar<65 or @singlechar>122 or (@singlechar>90 and
@singlechar<97))
begin
set @singlechar = cast(rand()*123 as int)
end
set @randchar = @singlechar
go
--97 122
--65 90
create procedure createwholestr
@wholestr varchar(20) output
as
```

```
declare @len varchar(10)
set @len = cast(rand()*20 as int)
while (@len>20 or @len<10)
begin
set @len = cast(rand()*20 as int)
end
declare @singlechar varchar(10)
declare @mystr varchar(20)
declare @i int
set @i=1
set @mystr = ''
while(@i<=@len)
begin
exec createsinglechar @singlechar output
--select @singlechar
set @mystr = @mystr + char(@singlechar)
set @i=@i+1
end
set @wholestr = @mystr
go
--exec createwholestr abc

create procedure insertrandstr
as
declare @data1 varchar(20)
declare @data2 varchar(20)
declare @ip1 int
declare @ip2 int
declare @ip3 int
declare @ip4 int
declare @ip varchar(15)

create table testtable(
id int not null,
username varchar(20) not null,
message varchar(20) not null,
logindate varchar(20) not null,
ip varchar(20) not null
)
declare @j int
set @j=1
while (@j<=10)
begin
exec createwholestr @data1 output
exec createwholestr @data2 output
exec createsingleip @ip1 output
exec createsingleip @ip2 output
exec createsingleip @ip3 output
exec createsingleip @ip4 output

set @ip=cast(@ip1 as varchar(3))+ '.'+cast(@ip2 as varchar)+ '.'+cast(@ip3
```

```
as varchar)+'.'+cast(@ip4 as varchar)

insert into testtable([id],[username],message,logindate,ip) values
(@j,@data1,@data2,getdate(),@ip)
set @j = @j + 1
end
select * from testtable
--drop table testtable
go

exec insertrandstr
```

下面我们简单地对存储过程进行解释。`Createsingleip` 存储过程生成 IP 地址格式的数字，每次调用都会返回一个 0~255 的随机数字；`Createsinglechar` 存储过程生成一个 a~z 或 A~Z 的随机字符；`Createwholestr` 存储过程调用 `createsinglechar` 存储过程生成 10~20 位长度的字符串；`Inserttrandstr` 存储过程新建一张 `testtable` 表，然后反复调用前面三个存储过程，进行插入操作，完成数据的添加。

无论如何通过 SQL 生成数据在维护和开发上总是比较麻烦的，这里推荐一个非常好用的数据生成工具 `DataFactory` 来解决环境搭建中需要进行大容量生成的问题。

`DataFactory` 是一种快速的、易用的、超强的数据产生器，它允许开发人员和 QA 很容易产生百万行有意义的、正确的测试数据库，`DataFactory` 首先读取一个数据库方案，用户随后单击鼠标产生一个数据库。该工具支持 DB2、Oracle、Sybase、SQL Server 数据库，并且还支持 ODBC 的连接方式，如果使用的系统是 MySQL 的话，那么是无法直接使用 `DataFactory` 来生成数据的。

那么如何使用 `DataFactory` 连接 MySQL 数据库呢？虽然 DF 没有提供对 MySQL 的直接连接，但是可以通过 ODBC 的方式来过渡，这里需要注意 MySQL 默认不支持 ODBC，所以先安装 `MyODBC` 工具来实现 ODBC 和 MySQL 的连接，再让 DF 通过 ODBC 的方式连接 `MyODBC`，从而实现曲线救国。

### 1. 为 Discuz NT 2.5 生成测试数据

我们来看看如何使用 `DataFactory` 快捷地完成构建 10 万条帖子记录数据的构建工作。

**STEP 01** 打开 `DataFactory`，新建一个数据库连接，设置 `DataFactory` 连接数据库类型为 SQL Server，如图 6.31 所示。



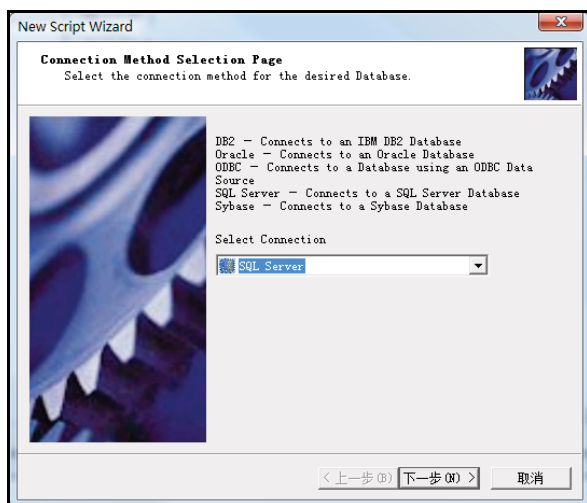


图 6.31 DF 中添加一个 SQL Server 数据库连接

**STEP 02** 在 DataFactory 中输入数据库的连接信息，如图 6.32 所示。

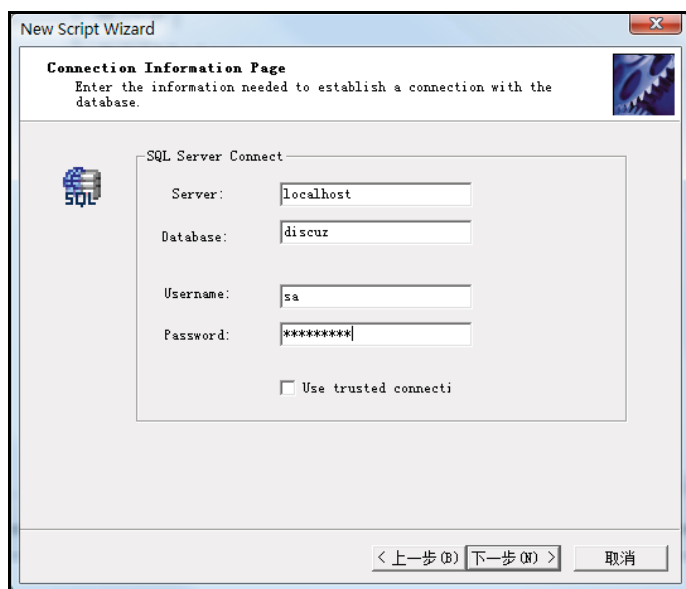


图 6.32 在 DataFactory 中设置 SQL Server 数据库连接信息

**STEP 03** 检测通过后，DataFactory 会列出该数据库中所有的表，选出需要注入数据的表后，就可以设置对于该表的数据插入数目和数据格式了。添加 dnt\_posts1 和 dnt\_topics 表，如图 6.33 所示。

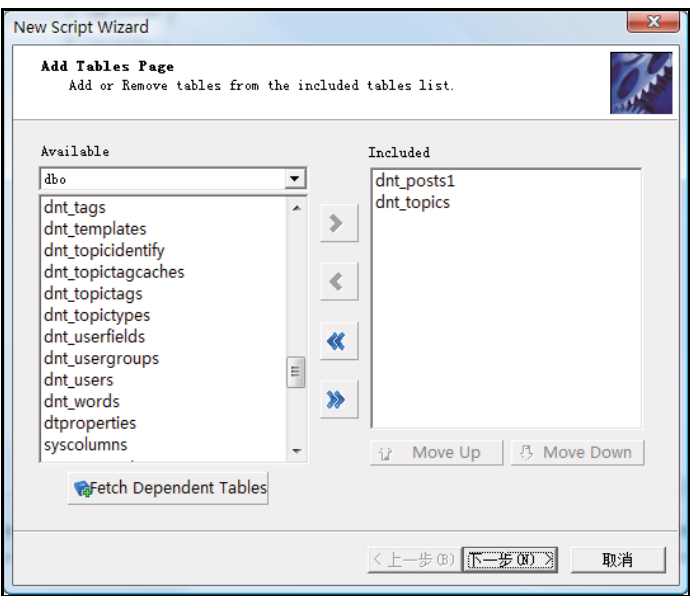


图 6.33 DataFactory 中选择需要生成数据的表名

首先根据 Discuz 论坛的数据字典了解保存帖子数据的格式，为我们通过 DataFactory 生成有效的论坛帖子记录提供支持。

根据数据字典，可以了解到当新增一个帖子时，数据库中的两张表会有直接的影响，其中，dnt\_post1 表记录了帖子的正文内容，而 dnt\_topics 表记录了每个板块的帖子列表，最终可以得到需要操作的属性列表，以及 DataFactory 的取值设置，如表 6.11 所示。

表 6.11 DataFactory 数据生成格式表 1

dnt_topics		
属性名	属性含义	数据规则
Fid	板块 id	随机 2~7
Poster	发帖人	Admin
Posterid	发帖人编号	1
Title	帖子标题	随机 40 长度字符串
Lastpostid	最后回帖主题编号	1
Lastposter	最后回复人	Admin
Tid	帖子标题 id	每次+1
dnt_posts1		
属性名	属性含义	数据规则
Pid	帖子正文 id	每次+1
Fid	板块 id	随机 2~7
Tid	帖子标题 id	每次+1
Parentid		1
Poster	发帖人	Admin
Posterid	发帖人 id	1
Title	帖子标题	随机 40 长度字符串
Message	帖子正文	随机 512 长度字符串

Htmlon		1
--------	--	---

接着按照每个字段的属性设置对应的取值方式。

例如，Fid 帖子所在的板块编号，论坛中有 6 个板块，编号为 2~7，现将这个值设置为随机取值，取值范围为“Between 2 and 7”，如 图 6.34 所示。

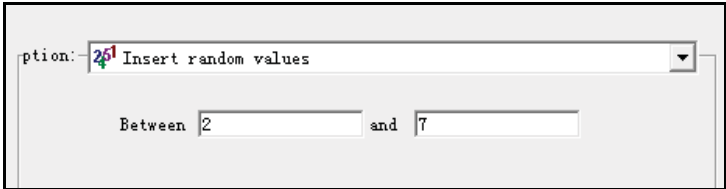


图 6.34 DataFactory 中设置数据生成成为 2~7 随机数

而 pid 及 tid 设置为从 1 开始每次值加 1，要注意的是 dnt\_topics.tid 在系统中设置为自动增长类型，需要强制修改该属性删除自动标识，才能确保生成的数据是有效的，如图 6.35 所示。

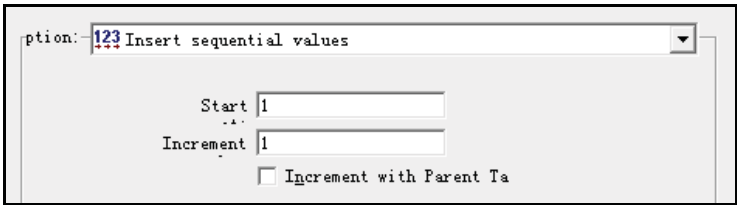


图 6.35 DataFactory 中设置数据生成成为从 1 开始，每次加 1

而对于发帖用户 Poster 可以设置为管理员发帖，如图 6.36 所示。

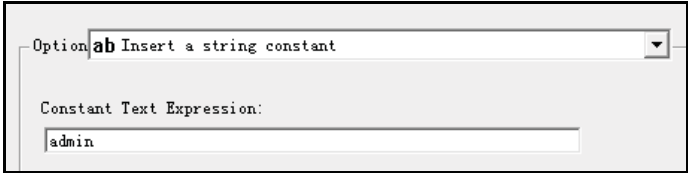


图 6.36 DataFactory 中设置数据格式为字符串 admin

将相关属性设置完成后，选择创建 100000 条记录，如图 6.37 所示。

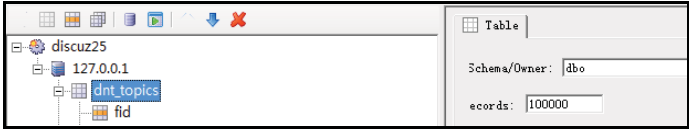


图 6.37 DataFactory 中设置对表生成的记录为 10 万条

单击 Run 按钮，稍等片刻，就在论坛中导入了 100000 条帖子记录，接着可以通过 SQL 查询一下该表的记录，如图 6.38 所示。

```

select top 100 * from dnt_posts1
select top 100 * from dnt_topics

```

!!!												
	pid	fid	tid	parentid	layer	poster	posterid	title			postdatetime	message
1	1	4	1	1	0	admin	1	Dhxgyrkjusvpnhugezdzqexajncrjbsnwhwicw			2009-04-05 20:36:00	EWLUCE
2	2	7	2	1	0	admin	1	Cxt dihdlaupgyydnwltvgnkfrdedlzfmmohxxgdx			2009-04-05 20:36:00	LCYPVI
3	3	4	3	1	0	admin	1	Umdkhvxpsnvnosxkpbwcufovvfxflsgllksuwvma			2009-04-05 20:36:00	00JPTD
	tid	fid	iconid	typeid	readperm	price	poster	posterid	title			
1	99998	2	0	0	0	0	admin	1	Mcudwvbsdcjjooahabacnnagxfpuscwtzhtwukuqn			
2	99941	2	0	0	0	0	admin	1	Ndynvndpphhfdpciemrrdutbrwhuntpeqqforunfv			
3	99927	2	0	0	0	0	admin	1	Lobduwzldbzezwxflpezisiysworyscipihfvcpvx			

图 6.38 检查数据库中的记录已经生成

可以看到记录都已加入到表中，再去论坛上检查一下，会看到每个板块都多了很多帖子，但是在统计上还存在一些问题，这是因为受论坛自身缓存机制的影响，在后台重置一下板块数据即可，最终效果如图 6.39 所示。

Discuz!NT 主题:100000, 帖子:100000				帖子标题
您上次访问是在: 2009/4/5 19:56:49 <a href="#">查看新帖</a>				今日:100000, 昨日:0, 最高日:100000(2009-4-5) <a href="#">精华区</a>
热门标签				
默认分类				
版块	主题	帖子	最后发表	
默认版块 (9955)	10010	9955	Hfipkqspfbjkbwxhjulrewzispulsfcyghghns by admin - 今天 20:37	
板块1 (19828)	19996	19828	Cfntjapfuuvvedkjovrgvjgztcbjqchihvkbpyj by admin - 今天 20:37	
板块2 (20108)	19965	20108	Tdphwxduevwnfnxuptbtdelphrrgmcmirbnkuwh by admin - 今天 20:37	
板块3 (20224)	19925	20224	Mcudwvbsdcjjooahabacnnagxfpuscwtzhtwukuqn by admin - 今天 20:37	
板块4 (20090)	20021	20090	Onwypmnlgyosnzpfsfemrlvgjncworatdclpzcet by admin - 今天 20:37	
板块5 (9795)	10083	9795	Neioxzdkwoczjhkqgkwidgfhnuhnrfxtatau by admin - 今天 20:37	

图 6.39 论坛中的数据已经生成

2. 为 Phpwind 8.5 生成测试数据

根据方案中的设定，这里需要为 Phpwind85 生成 500 万条帖子记录及 3 万注册会员数据。

由于在 Windows 2008 R2 下 MyODBC 无法正常与 DataFactory 正常工作，所以数据的生成是在 Windows 2003 下实现的。

添加帖子记录，这里需要操作的表包括 2 张，每张表 500 万条记录，如表 6.12 所示。

表 6.12 DataFactory 数据生成格式表 2

pw_threads		
属性名	属性含义	数据规则
Fid	板块 id	2
Icon	不详	0
Author	作者名	Admin
Authored	作者编号	1
Subject	帖子标题	随机 5-10 任意大写字符
Ifcheck	不详	1
Type	不详	0
Tid	帖子编号	从 1 开始每次自加 1
pw_tmsgs		

属性名	属性含义	数据规则
Userip	用户发帖 IP 地址	127.0.0.1
Tid	对应帖子编号	从 1 开始每次自加 1
overprint	不详	0
Ifwordsbf	不详	1
ifconvert	不详	1
contend	帖子正文	512 字符混合大小

添加注册用户记录这里也需要操作的表包括 2 张，每张表 3 万条记录，如表 6.13 所示。

表 6.13 DataFactory 数据生成格式表 2

pw_members		
属性名	属性含义	数据规则
username	用户名	混合模式 1.文本格式 cloud 2. 从 1 开始每次自加 1
password	密码	51testing 的 MD5 加密值 66c41f248c1375e6846b403175366f97
userstatus	不详	192
memberid	不详	8
groupid	不详	3
pw_memberdata		
属性名	属性含义	数据规则
Uid	用户编号	从 2 开始每次自加 1

如果没有办法很好地得到数据字典，也可以通过 Sybase PowerDesigner 逆向工程导出数据库中的数据字典，也可以直接通过 PowerDesigner 快速生成大量已注册用户，为后面的测试脚本提供相关数据。

容量生成后记得进行及时备份，方便测试后的还原工作。

6.2.3 测试环境搭建手册

搭建环境中还涉及很多细节内容，这里都编写在《测试环境搭建手册》中。

性能测试环境搭建手册

1. 虚拟机设置

网络设置为桥模式，2CPU X 2 核心，2GB 内存，启动 Virtualize Intel VT-x/EPT or AMD-V/RVI 支持。

2. 安装介质说明

- 官方 CentOS5.6 i386/x86\_x64



- Wdlinux.cn 发布的 Lanmp2.1 整合包

- Rpc.rstatd
- Nmon

### 3. 安装步骤

( 1 ) 先默认安装 CentOS5.6。

( 2 ) 安装选项为语言英文，区域上海，网卡 DHCP，密码 51testing，设置不选任何组件，再选择自定义安装：

- Development 中的 Development Libraries、Development Tools。
- Base System 中的 Administration Tools、Base、System Tools 附加 Dstat 和 sysstat。

( 3 ) 重启后使用 setup 命令关闭防火墙，然后关闭系统设置一个基础的 Snapshot，然后对该系统进行 VMWare Clone 操作，分别制作出 lamp、lnmp、lanmp 三套环境。

操作系统	平台	IP 地址/子网掩码均为 255.255.255.0
CentOS32 5.6	LAMP	192.168.11.20
CentOS32 5.6	LNMP	192.168.11.21
CentOS32 5.6	LANMP	192.168.11.22
CentOS64 5.6	LAMP	192.168.11.30
CentOS64 5.6	LNMP	192.168.11.31
CentOS64 5.6	LANMP	192.168.11.32

#### 3.1 安装 Lanmp

使用 Yum 默认安装可能需要的库文件：

```
yum install -y gcc gcc-c++ make autoconf libtool-ltdl-devel gd-devel freetype-devel  
libxml2-devel libjpeg-devel libpng-devel openssl-devel curl-devel patch libmcrypt-devel  
libmhash-devel ncurses-devel sudo bzip2
```

使用 tar xvf lanmp\_v2.1.tar.gz 解包，然后使用 sh lanmp.sh 启动安装，根据菜单选择依次安装，需要连网（安装时间较长）。

#### 3.2 安装 Rpc.rstatd

#### 3.3 根据情况安装 nmon

#### 4. Snapshot 制作规范及基础配置

(1) 安装完成关机，然后设置 Snapshot，减少空间开销。

(2) 使用 Unzip 命令完成 Phpwind\_UTF8\_8.5.zip 解压：

```
Unzip Phpwind_UTF8_8.5.zip
```

(3) 然后将 upload 目录中的内容通过 mv 命令移动到/www/web/default 下改名为

Phpwind85：

```
Mv Phpwind_UTF8_8.5/upload /www/web/default/Phpwind85
```

(4) 为该目录设置权限：

```
Chmod 777 -R /www/web/default/Phpwind85
```

(5) 接着就可以在浏览器中访问服务器 IP 地址/Phpwind85 启动安装了。安装使用 MySQL 作为数据库，数据库密码为 wdlinux.cn，数据库名称为 Phpwind85，管理员密码为 51testing。

(6) 安装完 Phpwind85，访问一次，关机后做一次 Snapshot。

(7) DF 数据生成帖子和用户后制作一次 Snapshot。

(8) 在使用 DF 前需要为数据库开启远程连接权限，在 Linux 命令行中输入：

```
Mysql -pwdlinux.cn mysql
```

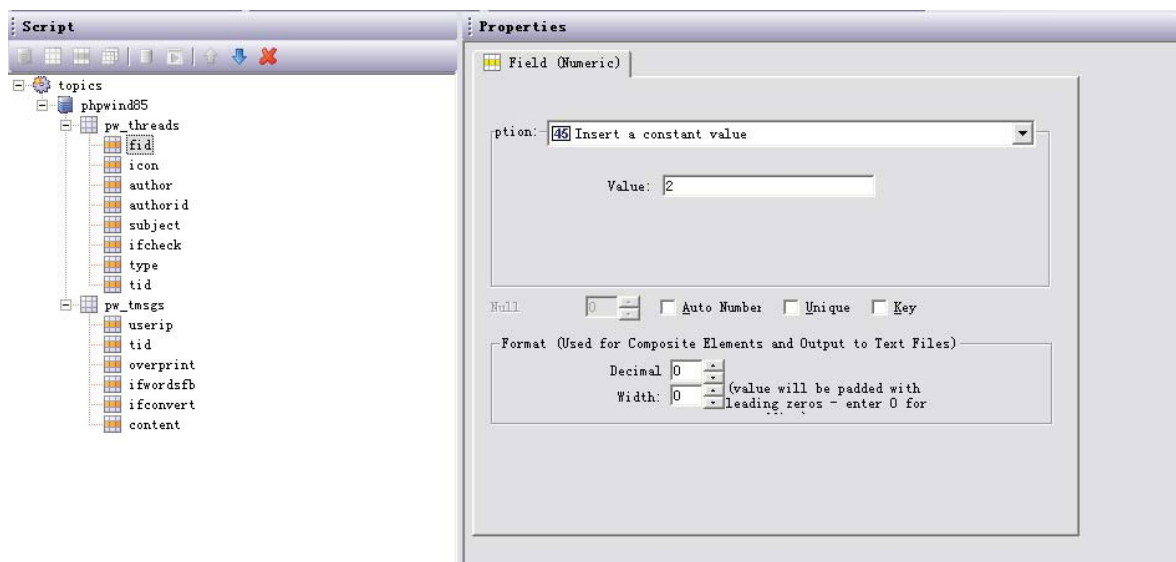
在 MySQL 命令行下输入下面的语句：

```
mysql> GRANT ALL PRIVILEGES ON *.* TO root@"%" IDENTIFIED BY "root";  
mysql> flush privileges;
```

为系统添加一个用户名为 root，密码为 root 可以远程连接的账户。flush privileges 表示从 MySQL 数据库的 grant 表中重新加载权限数据。因为 MySQL 把权限都放在了 Cache 中，所以在做完更改后需要重新加载。

## 5. Df 数据生成规则及 Snapshot

使用 DF 生成数据帖子数 500 万条（2 小时 50 分，数据库文件 2.53GB），可以登录的用户 3 万条（58 秒），后台刷新出现内容后，关机设置 Snapshot。



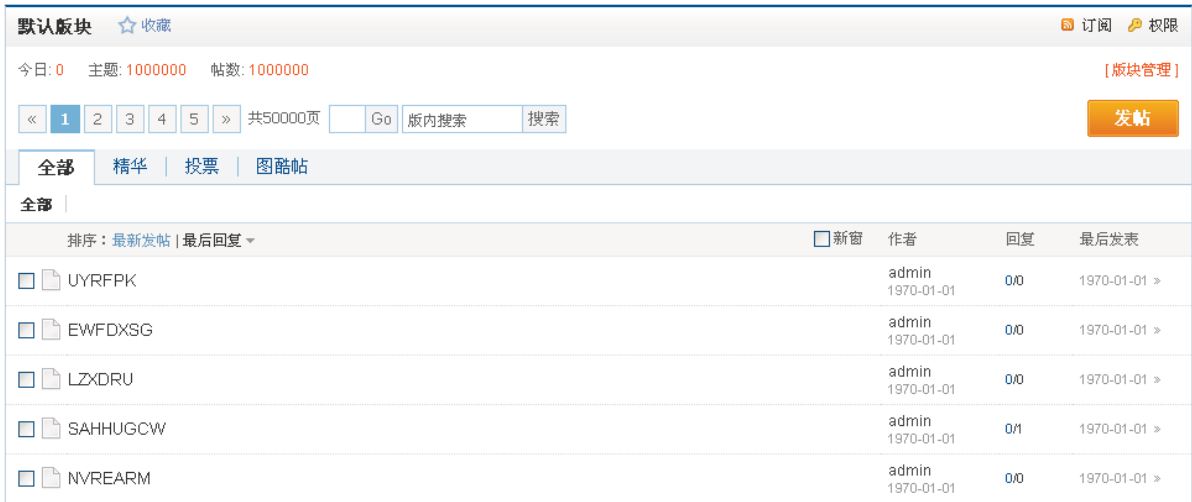
tid 用顺序加 1，设置中为了保证记录能进去，将每次提交的数据事务修改为 100，执行的时间比较长。



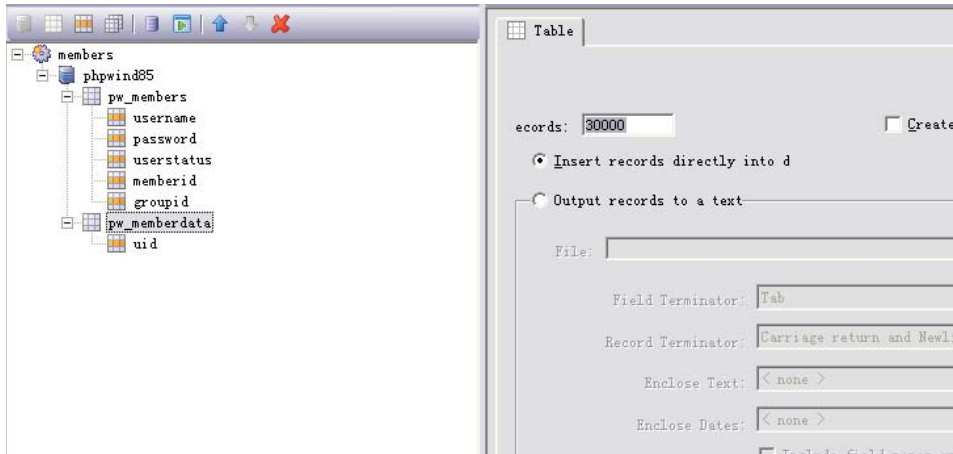
记录添加后需要在论坛后台运行缓存更新才能在系统上看到效果。



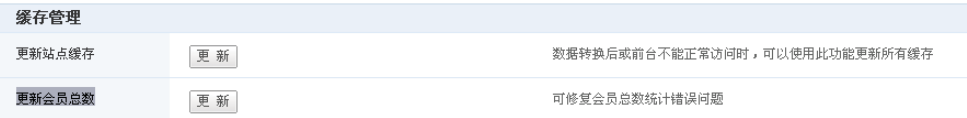
更新完毕后，回到论坛首页刷新就可以看到效果了。



用户的添加都用统一的密码，51testing 的 MD5 值为 66c41f248c1375e6846b403175366f97。









这样数据就生成完毕了，单击后台的“更新”按钮更新后台用户。



刷新用户数。



其余系统使用数据库同步的方式部署，将有数据的系统中的对应数据库文件通过 FTP 方式覆盖别的系统。

 pw_memberdata.MYD	586 KB	MYD 文件	2011-12-14 22:39	A
 pw_memberdata.MYI	605 KB	MYI 文件	2011-12-14 22:47	A
 pw_members.MYD	2,579 KB	MYD 文件	2011-12-14 19:44	A
 pw_members.MYI	1,149 KB	MYI 文件	2011-12-14 22:47	A
 pw_threads.MYD	189,453 KB	MYD 文件	2011-12-14 20:59	A
 pw_threads.MYI	351,908 KB	MYI 文件	2011-12-14 22:47	A
 pw_tmsgs.MYD	2,656,250 KB	MYD 文件	2011-12-14 22:34	A
 pw_tmsgs.MYI	50,110 KB	MYI 文件	2011-12-14 22:47	A

6. 最优对比平台

在安装 Phpwind85 的系统中通过 Snapshot 回到安装 Phpwind85 前 ,然后安装 DiscuzX2 系统，再制作一个 Snapshot。该对比平台 Phpwind85 及 discuzx2 均没有数据，提供接口性能测试评估用。出于生成数据环境搭建的复杂性，这里对 discuzx2 和 Phpwind85 使用全新空数据进行测试。

7. 基准测试数据

在主机上使用 EVEREST 5.50.2100 作为基准测试工具。所有数据只代表 HOST 主机，不代表虚拟机中的基准性能 ,由于所有虚拟机硬件环境设置相同 ,所以不影响平台公平性。主板中设置关闭 Intel Turbo Boost Tech 智能提速功能。

## 1) 降频测试

CPU 实际频率 2433MHz，内存实际频率 TripleDDR3-1079。

测试项目	测试结果
内存读取	11657MB/s
内存写入	9505MB/s
内存复制	13632MB/s
内存潜伏	65.6ns
CPU Queen	27697
CPU ZLib	81412KB

## 2) 标准测试

CPU 实际 133\*21 频率 2.8GHz，内存实际频率 TripleDDR3-1333。

测试项目	测试结果
内存读取	15005 MB/s
内存写入	14318 MB/s
内存复制	16341 MB/s
内存潜伏	61.3ns
CPU Queen	31901
CPU ZLib	93776KB/S

## 3) 超频测试

CPU 实际 166\*18 频率 3.0GHz，内存实际频率 TripleDDR3-1660。

测试项目	测试结果
内存读取	16811MB/s
内存写入	13705MB/s
内存复制	18726MB/s
内存潜伏	52.2ns
CPU Queen	34140
CPU ZLib	100864KB

注：超频后性能大概有接近 30%的提升

## 4) 极限超频测试

CPU 实际 200\*20 频率 4.0GHz，内存实际频率 TripleDDR3-1600。

测试项目	测试结果
内存读取	17225MB/s



内存写入	16498MB/s
内存复制	23368MB/s
内存潜伏	50.0ns
CPU Queen	45519
CPU ZLib	135186KB/s

注：极限超频后还能再提升 30%的性能。

5 ) 硬盘测试使用 HDTune 4.50

① 随机存取

测试项目	IOPS	平均存取时间	平均速度
4KB 读取	68	14ms	0.268MB/s
64KB 读取	62	16ms	3.881MB/s
1MB 读取	26	27ms	36.600MB/s

② 基准测试 ( 10gB 测试 )

测试项目	最低	最高	平均	存取时间	突发传输速度	CPU 占用率
基准读取	145.3MB/s	265.3MB/s	218.8MB/s	6.8ms	2464.9MB/s	3.8%

注：写入由于需要重新进行磁盘分区，所以没有进行测试。

虚拟机基准测试结果，只比较 32 位和 64 位平台性能，使用 LANMP 2.1 系统自带的 iproberv0.024 平台进行测试。为了确保相对公平，所有测试均执行 6 次，取最后一次值为准。

5 ) 降频测试

① CentOS64

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风 CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.167秒</b> TEST_1	<b>0.164秒</b> TEST_2	<b>0.049秒</b> TEST_3

② CentOS32

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.179秒</b> TEST_1	<b>0.177秒</b> TEST_2	<b>0.051秒</b> TEST_3

## 6) 标准频率测试

### ① CentOS64

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.179秒</b> TEST_1	<b>0.183秒</b> TEST_2	<b>0.069秒</b> TEST_3

### ② CentOS32

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.18秒</b> TEST_1	<b>0.18秒</b> TEST_2	<b>0.06秒</b> TEST_3

## 7) 超频测试

### ① CentOS64

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.134秒</b> TEST_1	<b>0.139秒</b> TEST_2	<b>0.048秒</b> TEST_3

### ② CentOS32

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.149秒</b> TEST_1	<b>0.143秒</b> TEST_2	<b>0.038秒</b> TEST_3

8 ) 极限超频测试

① CentOS64

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.104秒</b> TEST_1	<b>0.108秒</b> TEST_2	<b>0.05秒</b> TEST_3

② CentOS32

服务器性能检测			
检测对象	整数运算能力测试 (1+1运算300万次)	浮点运算能力测试 (开平方300万次)	数据I/O能力测试 (读取10K文件10000次)
Tahiti 的电脑(P4 1.7G 256M WinXP)	1.421秒	1.358秒	0.177秒
PIPNI免费空间(2004/06/28 02:08)	2.545秒	2.545秒	0.171秒
神话科技风CGI型(2004/06/28 02:03)	0.797秒	0.729秒	0.156秒
您正在使用的这台服务器	<b>0.156秒</b> TEST_1	<b>0.154秒</b> TEST_2	<b>0.063秒</b> TEST_3

可以看到 64 位系统普遍比 32 位系统能够更好地使用 CPU 进行整数和浮点计算，超频后会明显提升 64 位下的处理能力，但 32 位不是十分明显。

6.7.1 平台对比性能测试报告

Phpwind85 最佳平台性能测试报告			
性能测试目的			
评估 Phpwind85 在 LAMP、LNMP、LANMP 三大架构下的运行情况，评估 CentOS 5.5 32 位系统及 CentOS5.6 64 位系统优劣。			
测试环境			
设备名称	硬件配置	软件配置	备 注
性能测试进阶指南——LoadRunner 11 实战			

服务器	CPU: i7 930 2.80GHz 1333×21 内存: DDR3 1600 2GB×3 (三通道) 硬盘: ST 7200.12 1TB (7200 转\32MB 缓存) ×2 RAID 0 网卡: Intel10/100/1000 自适应	操作系统: Windows 7 X64 SP1 6.1.7601 Vmware8.0.0 build- 471780 CentOS 5.5 32 位 CentOS 5.6 64 位 Web 服务: Apache、Nginx 数据库: MySQL 监控工具: Rpc.rstatd	所有测试环境均在虚拟机下完成, 网络设置为桥模式, 2CPU×2 核心, 2GB 内存, 启动 Virtualize Intel VT-x/EPT 或 AMD-V/RVI 支持 关闭 CPU 中的智能核心加速功能
-----	---	--	---

注：软件环境安装为 wdlinux.cn 发布的 LANMP2.1 安装包，该包包含的组件版本为：

httpd-2.2.17  
nginx-0.8.54  
php-5.2.17  
mysql-5.1.56  
phpmyadmin-3.3.7  
zend-3.3.3  
eAccelerator-0.9.6.1  
pure-ftpd-1.0.32

在虚拟机下安装 CentOS5.6 32 位和 64 位系统 ,分别安装对应的 LAMP、LNMP、LANMP 环境。

操作系统	平台	IP 地址/子网掩码均为 255.255.255.0
CentOS32 5.6	LAMP	192.168.11.20
CentOS32 5.6	LNMP	192.168.11.21
CentOS32 5.6	LANMP	192.168.11.22
CentOS64 5.6	LAMP	192.168.11.30
CentOS64 5.6	LNMP	192.168.11.31
CentOS64 5.6	LANMP	192.168.11.32

每个环境分别安装 Phppwind85，并且为其生成 500 万随即帖子及 3 万会员数据。

测试工具及测试方法的说明

使用 LR 生成 4 个测试用例，包括注册用户、浏览帖子、查询、回帖 4 项。

序 号	业务名称	脚本编写要求	备 注
1	注册用户	不同的用户使用不同的注册数据,注册数据覆盖全部的业务规则, 在用户并发情况下, 每个用户注册时间	
2	查询	不同登录用户查询不同内容, 获得查询返回所需要的时间	查询内容需要随机
3	浏览帖子	游客浏览不同板块下的首页帖子, 在用户并发情况下, 打开每个帖子所需要的时间	用户会随机选择板块进行浏览, 这里的数

			据需要使用参数化
4	回帖	不同登录用户在板块中回帖,回帖提交所需要的响应时间和登录所需要花费的时间	用户随机选择板块进行发帖

在场景中分别按照下表进行负载。

序 号	测试场景描述
1	50 个并发注册用户,并发率为 10% 5 用户启动,每隔 60 秒增加 5 用户,持续 5 分钟,立即结束负载
2	500 个用户查询 10 用户启动,每隔 30 秒增加 25 用户,持续 5 分钟,立即结束负载
3	300 个游客浏览帖子 25 用户启动,每隔 30 秒增加 25 用户,持续 5 分钟,立即结束负载
4	100 个用户登录后发帖,回帖并发率为 10%,登录不设置并发 10 用户启动,每隔 30 秒增加 10 用户,持续 5 分钟,立即结束负载

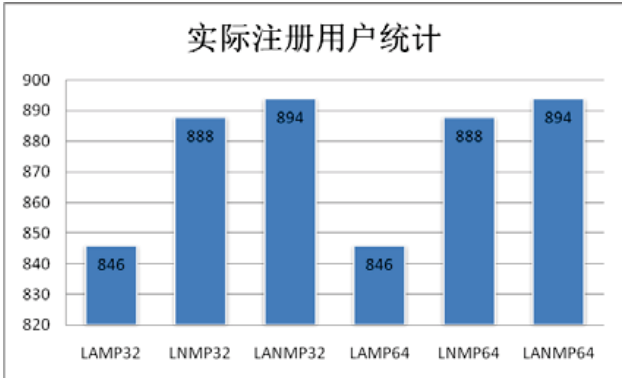
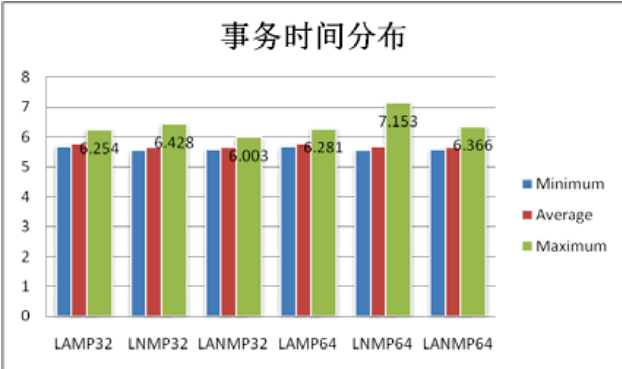
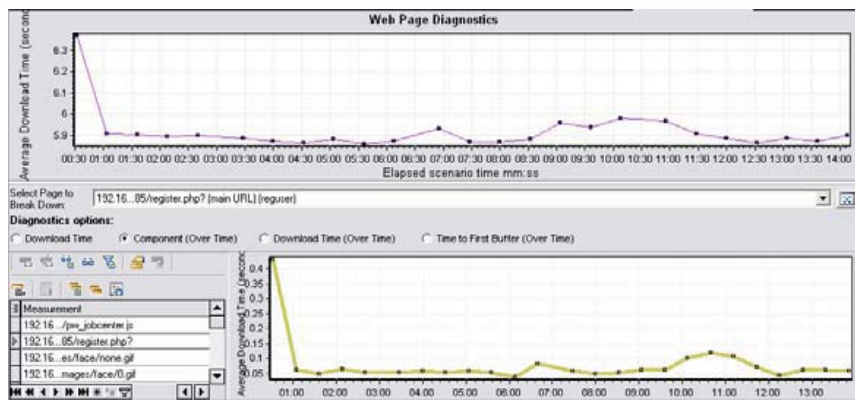
获得事务完成数及资源占用情况。

测试结果数据列表

### 1) 注册用户

Transaction Name	Platform	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop	实际生成
Reguser	LAMP32	5.708	5.782	6.254	0.047	5.829	846	0	0	846
Reguser	LNMP32	5.55	5.659	6.428	0.169	5.96	884	4	0	888
Reguser	LANMP32	5.587	5.658	6.003	0.053	5.71	893	1	0	894
Reguser	LAMP64	5.706	5.785	6.281	0.049	5.835	846	0	0	846
Reguser	LNMP64	5.553	5.684	7.153	0.228	5.911	888	0	0	888
Reguser	LANMP64	5.596	5.669	6.366	0.063	5.712	894	0	0	894

通过 WebPageBreakdown 可以看到注册的事务时间稳定在 5~6 秒内,而其中注册的请求稳定在 0.1 秒内,说明注册没有明显的负载瓶颈。再核对服务器资源监控均未发现明显资源瓶颈。



根据统计可以看到在低负载情况下，LANMP32 平台的响应时间表现是最好的（最长与最短时间差只有 0.1 秒（约 2% 的误差，可以忽略），而处理能力 LANMP32 和 LANMP64 都为最好（记录生成最多与最少误差 48 个用户（约 6% 误差））。

综合评定在注册用户这一项上 LANMP32 表现最好。但由于响应时间的误差非常小，而 64 位系统在扩展能力和资源管理上更胜一筹，所以推荐使用 LANMP64 平台。

Platform	平均响应时间排名	处理能力排名	事务失败排名	综合得分 (越小越好)	综合评分	备注
LAMP32	5	3	1	9		
LNMP32	2	2	3	7		
LANMP32	1	1	2	4	推荐☆☆	高效稳定
LAMP64	6	3	1	10		
LNMP64	4	2	1	7		
LANMP64	3	1	1	5	推荐☆☆	稳定、高效、

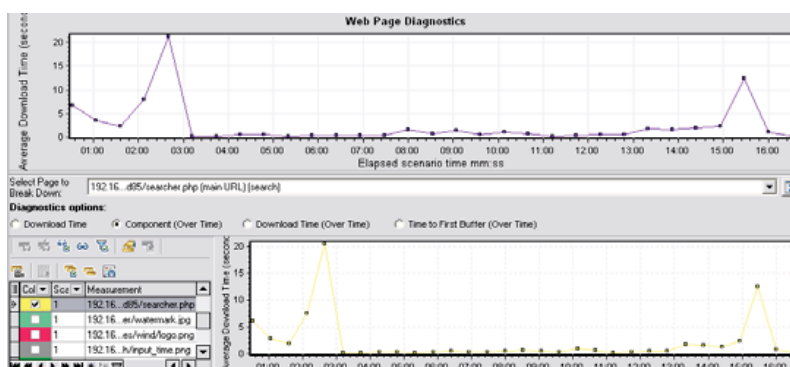


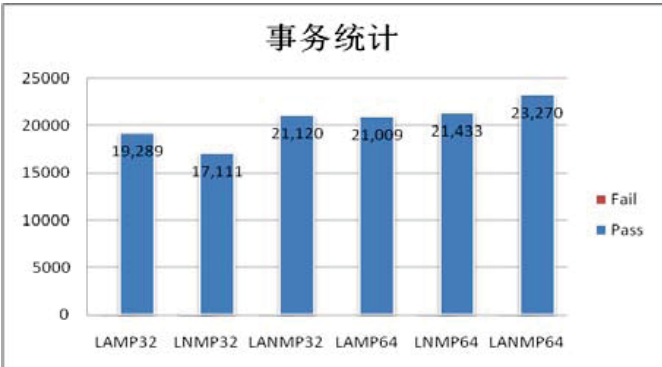
					☆	扩展能力强
--	--	--	--	--	---	-------

## 2) 查询帖子

Transaction Name	Platform	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop	实际生成
search	LAMP32	0.013	0.976	109.548	4.768	1.285	19,289	62	0	/
search	LNMP32	0.011	2.008	42.216	3.269	6.27	17,111	257	0	/
search	LANMP32	0.012	1.13	87.356	4.958	1.708	21,120	2	0	/
search	LAMP64	0.012	0.474	28.902	1.613	0.657	21,009	0	0	/
search	LNMP64	0.01	1.086	27.969	2.179	3.14	21,433	123	0	/
search	LANMP64	0.01	0.749	83.49	2.209	1.333	23,270	2	0	/

通过 WebPageBreakdown 可以看到查询的事务时间稳定在 1 秒内，但由于没有对该页面进行手工事务查询，加之单用户点击查询时响应时间超过 30 秒，所以这个结果可能存在问题，建议更换为管理员用户并且添加手工事务验证。再核对服务器资源监控可以发现在 2:40 秒时达到 CPU 瓶颈后，CPU 暂用率逐渐下降，说明时间点后的负载存在一定的问题，要么数据库缓存发挥了超高的效率，要么就是没有真正地对系统进行有效的负载查询。





根据统计可以看到在高负载情况下，LAMP64 平台的响应时间表现是最好的（最长与最短时间差有 1.6 秒(约 500%的误差)）。而处理能力 64 位平台下的 3 套环境与 LANMP32 都比较接近，其中 LANMP64 最好（相对第二名事务有 10%的提升）。在错误问题上 LAMP64 没有出现事务错误，而 LANMP64 排名第二（只有 2 个事务失败），LNMP 架构在这里全面溃败，出现了超过 100 个的事务失败，体现出在高负载下 CGI 模式运行 PHP 容易出错的缺陷。

综合评定在注册用户这一项上 LANMP64 表现最好，虽然事务中出现了 2 个事务失败，但是相对 2.3 万的整体事务数目这个错误率只有万分之一，最终推荐使用 LANMP64 平台。而没有错误的 LAMP64 由于处理能力相对低了 10%，所以作为第二推荐平台。

Platform	平均响应时间排名	处理能力排名	事务失败排名	综合得分 (越小越好)	综合评分	备注
LAMP32	3	4	3	10		
LNMP32	6	6	5	17		
LANMP32	5	3	2	10		
LAMP64	1	2	1	4	推荐☆☆	稳定不出错
LNMP64	4	5	4	13		
LANMP64	2	1	2	5	推荐☆☆☆	处理能力突出

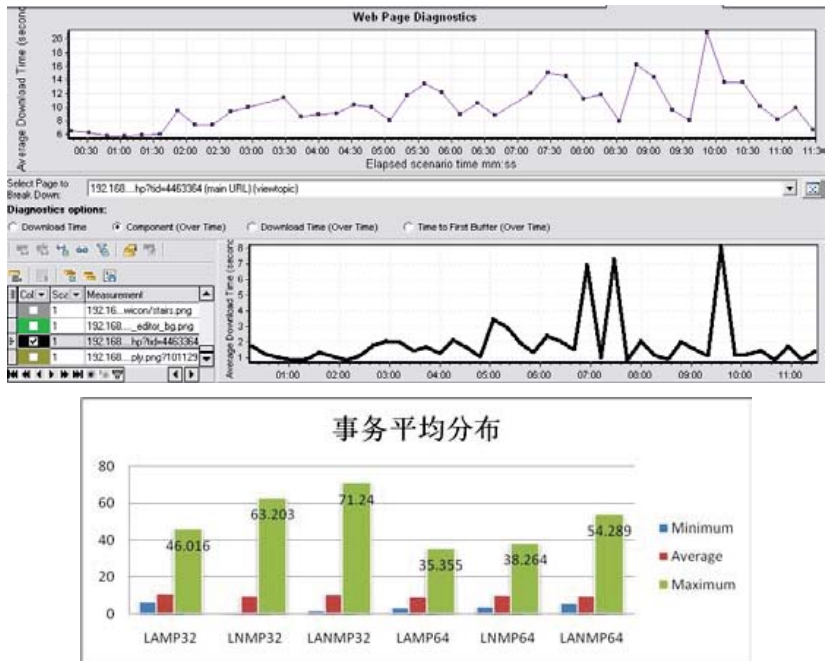
3 ) 随机看帖

Transaction Name	Platform	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	St op	实 际
------------------	----------	---------	---------	---------	----------------	------------	------	------	-------	-----

										生成
viewtopic	LAMP32	6.293	10.836	46.016	5.799	17.884	5,529	0	0	/
viewtopic	LNMP32	0.577	9.766	63.203	4.568	16.035	6,637	0	0	/
viewtopic	LANMP32	1.788	10.525	71.24	5.783	18.006	6,477	1	0	/
viewtopic	LAMP64	3.664	9.067	35.355	4.503	15.655	6,281	0	0	/
viewtopic	LNMP64	4.136	10.158	38.264	3.922	14.845	6,217	0	0	/
viewtopic	LANMP64	5.398	9.503	54.289	4.116	14.207	6,808	1	0	/
*viewtopic500	LAMP32	5.095	15.964	55.788	8.329	27.896	8,176	0	0	/
*viewtopic500	LNMP32	5.524	13.501	60.58	5.501	19.426	10,363	0	0	/

注：由于客户端负载处理能力不足，由 500 用户负载减少为 300，所以补充了 2 个 500 用户运行场景的数据

通过 WebPageBreakdown 可以看到看帖的事务时间在 20 秒内，平均稳定在 12 秒左右，在 1:30 秒后开始大幅上升（用户负载 80 个）。看帖的请求稳定在 4 秒内，中间出现了几次响应时间较大波动的情况。再核对服务器资源监控可以发现在 Context Switch Rate 计数器中就是 Process（Thread）的切换有明显上升，说明开始 CPU 忙于切换，从而导致影响吞吐量，而 CPU 本身的占用率并不是很高，说明 CPU 队列长度是影响响应时间的主要因素。





根据统计可以看到在大型的散列密集访问负载情况下，LAMP64 平台的响应时间表现是最好的（最长与最短时间差 10% 的误差）。而处理能力 LANMP64 最好（相对第二名事务有 2% 的提升）。在错误问题上 LANMP 都出现了一次事务错误，基本可以忽略，而其他平台均没有出现事务失败的情况。在最大响应时间上，64 位系统由于对 CPU 的支持更好，明显得到了接近 30%~40% 的响应时间下降。

综合评定在随机看帖这一项上 LANMP64 表现最好。而第二推荐平台是 LAMP64 和 LNMP32，这里比较奇怪的是 LNMP64 没有发挥出应该有的特长，可能与 64 位下 CGI 模式运行策略有关系。最终推荐使用 LANMP64 平台。

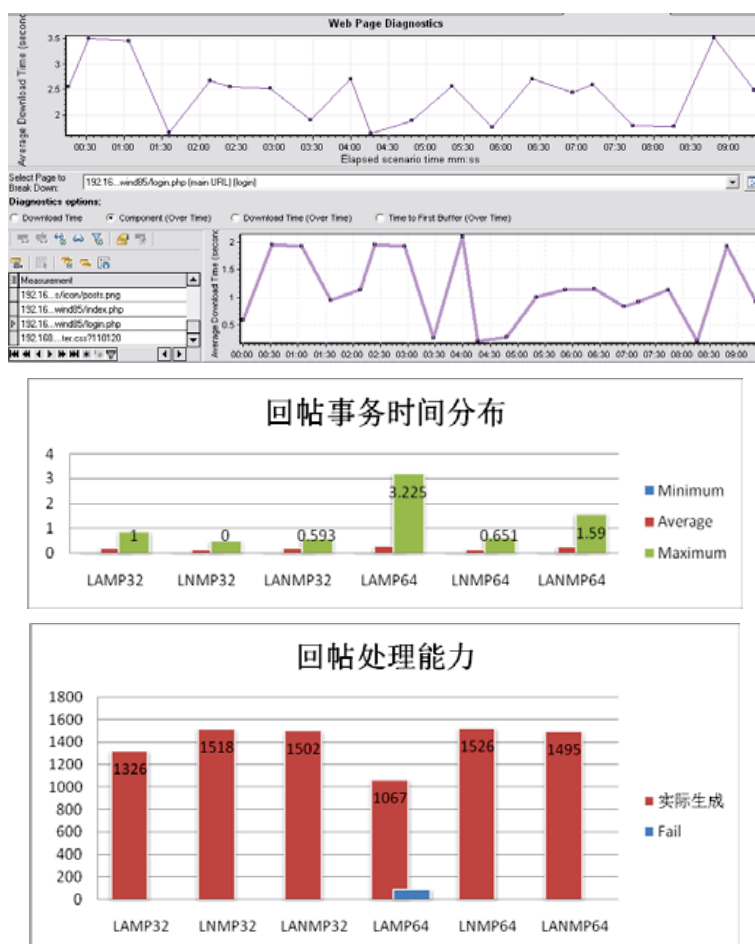
Platform	平均响应时间排名	处理能力排名	事务失败排名	综合得分 (越小越好)	综合评分	备注
LAMP32	6	6	1	13		
LNMP32	3	2	1	6	推荐☆☆	稳定不出错
LANMP32	5	3	2	10		
LAMP64	1	4	1	6	推荐☆☆	稳定不出错
LNMP64	4	5	1	10		
LANMP64	2	1	2	5	推荐☆☆☆	处理能力、响应时间总和突出

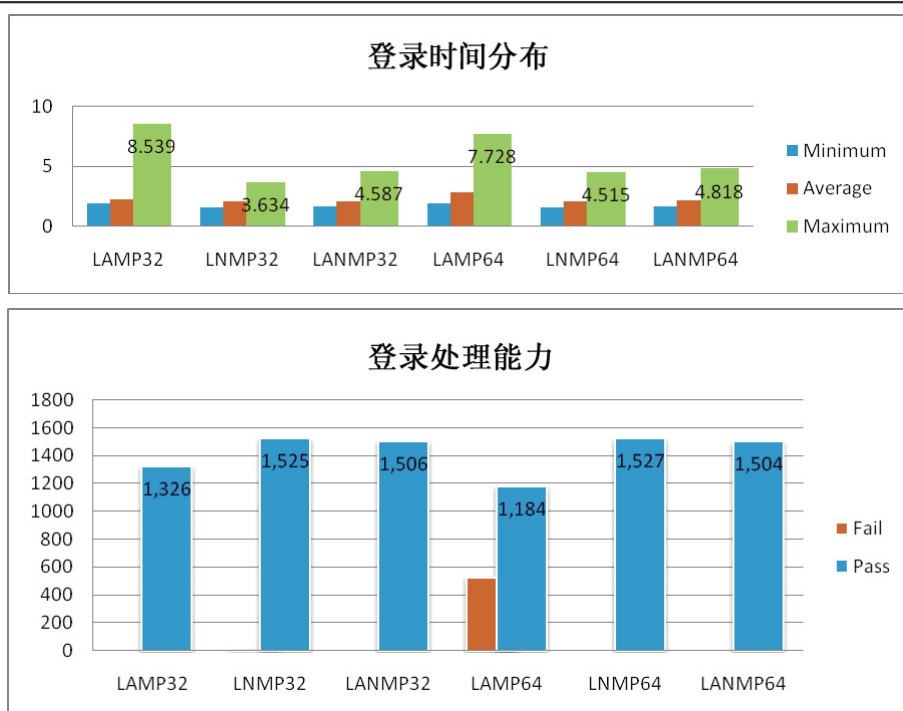
4 ) 随机回帖

Transaction Name	Platform	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop	实际生成
replytopic	LAMP32	0.075	0.211	1	0.092	0.284	1326	0	0	1326
replytopic	LNMP32	0.073	0.127	0	0.057	0.181	1518	0	0	1518
replytopic	LANMP32	0.073	0.202	0.593	0.077	0.292	1502	0	0	1502

replytopic	LAMP64	0.076	0.293	3.225	0.17	0.437	1067	95	0	1067
replytopic	LNMP64	0.074	0.143	0.651	0.071	0.204	1526	0	0	1526
replytopic	LANMP64	0.073	0.252	1.59	0.151	0.355	1495	0	0	1495
login	LAMP32	1.901	2.25	8.539	0.427	2.454	1,326	0	0	/
login	LNMP32	1.592	2.03	3.634	0.121	2.149	1,525	2	0	/
login	LANMP32	1.603	2.095	4.587	0.171	2.228	1,506	0	0	/
login	LAMP64	1.873	2.78	7.728	1.023	4.345	1,184	527	0	/
login	LNMP64	1.58	2.041	4.515	0.199	2.159	1,527	0	0	/
login	LANMP64	1.631	2.153	4.818	0.239	2.428	1,504	0	0	/

通过 WebPageBreakdown 可以看到登录的事务时间在 2 秒内，而回帖时间都稳定在 0.3 秒以内。再核对服务器资源监控可以发现在 Context Switch Rate 计数器中就是 Process ( Thread ) 的切换有明显上升，说明开始 CPU 忙于切换，从而导致影响吞吐量，而 CPU 本身的占用并不是很高，说明 CPU 队列长影响较大。





根据统计可以看到在大型的散列密集访问负载情况下，LNMP32 平台的登录回帖响应时间表现是最好的，从回帖平均响应时间来看最大和最小差距有 80%，而登录平均响应时间最大和最小差距很小（由于回帖是单请求事务，而登录是多请求事务，所以登录请求本身的误差会被页面刷新掩盖）。而处理能力 LANMP64 最好，相对垫底的 LAMP32 有 15% 的提升。LAMP64 出现了比较多的事务失败情况。整体上 64 位系统均略好于 32 位系统的表现，LNMP64 略胜于 LNMP32。

Platform	平均响应时间排名（登录）	处理能力排名（登录）	事务失败排名（登录）	平均响应时间排名（回帖）	处理能力排名（回帖）	事务失败排名（回帖）	综合得分（越小越好）	综合评分	备注
LAMP32	5	5	1	4	5	1	21		
LNMP32	1	2	2	1	2	1	9	推荐☆☆☆	处理能力、响应时间总和突出
LANMP32	3	3	1	3	3	1	14		
LAMP64	6	6	3	6	6	2	29		
LNMP64	2	1	1	2	1	1	8	推荐	处理能力



								☆☆☆	力、响应 时间总和 突出
LANMP64	4	4	1	5	4	1	19		

## 测试结果总结

Platform	注册用户 总分	随机查询 总分	随机看帖	随机登录 回帖	总分 (越小越好)	备注
LAMP32	9	10	13	21	53	查询存在部分 事务失败情况
LNMP32	7	17	6	9	39	注册用户存在 极个别事务失 败，查询存在 部分事务失败 情况
LANMP32	4	10	10	14	38	
LAMP64	10	4	6	29	49	回帖中存在 30%以上是事 务失败情况
LNMP64	7	13	10	8	38	查询存在部分 事务失败情况
LANMP64	5	5	5	19	34	

综上所述可以发现，传统的 LAMP 无论是 32 位还是 64 位都处在排名的最后，由于 Apache 的静态处理能力相对 Nginx 劣势太多，导致在系统真正使用中全面落败，甚至牵连 MySQL 降低了系统的处理能力。在大部分测试中 64 位系统都会比 32 位系统表现有所

提升，而从未来的扩展能力和内存管理的能力角度考虑，64 位系统为不二之选。而

LANMP 架构相对于 LNMP 从性能角度来说相差无几，但在大负载模式下 CGI 模式的 PHP 模块稳定性略差，最终测试结果为 LANMP64 平台为最佳系统运行平台。

## 8.6 Web Service

Web Service 是一种构建应用程序的普遍模型，可以在任何支持网络通信的操作系统中实施运行；它是一种新的 Web 应用程序分支，是自包含、自描述、模块化的应用，可以发布、定位、通过 Web 调用。Web Service 是一个应用组件，它逻辑性地为其他应用程序提供数据与服务。各应用程序通过网络协议和规定的一些标准数据格式(HTTP、XML、SOAP)来访问 Web Service，通过 Web Service 内部执行得到所需结果。

Web Service 可以执行从简单的请求到复杂商务处理的任何功能。一旦部署以后，其他 Web Service 应用程序可以发现并调用它部署的服务。在构建和使用 Web Service 时，主要用到以下几个关键的技术和规则。

- XML：描述数据的标准方法。
- SOAP：表示信息交换的协议。
- WSDL：Web 服务描述语言。
- UDDI (Universal Description, Discovery and Integration)：通用描述、发现与集成，它是一种独立于平台的、基于 XML 语言的、用于在互联网上描述商务的协议。

这里我们使用的案例是 [http://www.webxml.com.cn/zh\\_cn/index.aspx](http://www.webxml.com.cn/zh_cn/index.aspx) 提供的天气预报，该服务页面为 <http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx>。由于该服务是使用微软.NET 标准实现的，所以该页面里面直接提供了一些调用的说明。

调用 Web Service 的方式很多，这里我们介绍两种最常用的调用方式 WSDL 和 SOAP，另外再扩展介绍两个底层的做法，基于 HTTP/HTML 协议和 Windows Sockets 协议来实现 Web Service 调用。

### 8.6.1 基于 WSDL 的调用

新建一个基于 Web Service 协议的脚本，然后单击 SOA Tools 菜单下的 Manage Services。在弹出的窗口中单击 Import，导入我们需要的连接 WSDL 串。这里需要导入的 WSDL 串地址为 <http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx?WSDL>。

如果不知道 WSDL 的地址，可以询问一下开发人员，或者尝试在浏览器中访问服务页面并在页面地址后添加“?WSDL”关键字，如果返回一个 XML，一般说明这个数据包就是我们需要的 WSDL 包。

在导入窗口中选择 URL 方式导入，输入 WSDL 文件地址，单击 Import 后得到该服务信息，如图 8.23 所示。

在成功导入 Web Service 的连接信息后，我们需要开始进行服务调用了，单击 OK 按钮关闭 Manage Services 窗口。选择菜单 SOA Tools 下的 Add Service Call，弹出 New Web Service Call 窗口，如图 8.24 所示。

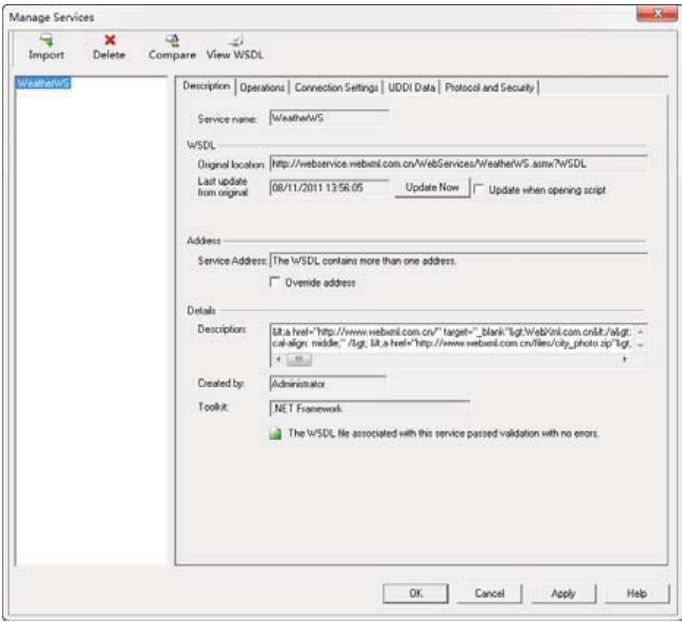


图 8.23 成功导入的服务信息

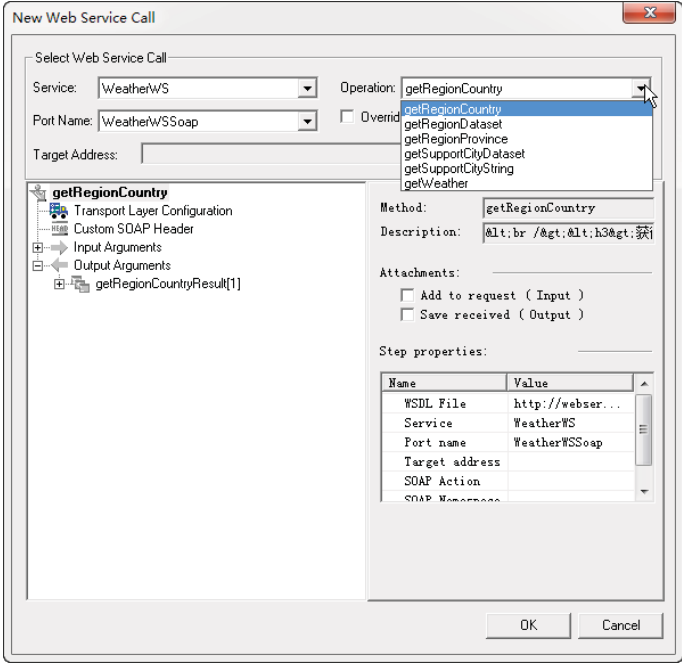


图 8.24 New Web Service Call

在 Operation 下拉列表中列出了该 Service 所提供的所有操作接口，而下方提供了该接口的输入/输出结构。

这里我们做一个简单的业务流程，用户首先查询自己所在城市的城市编号，再通过编号查询得到对应城市的天气情况。

方法 getSupportCityString 提供了对于城市查询返回对应编号的服务，这里选择该 Operation，并且在 Input Arguments 中输入我们需要查询的城市名称“上海”（可以参数化），如图 8.25 所示。

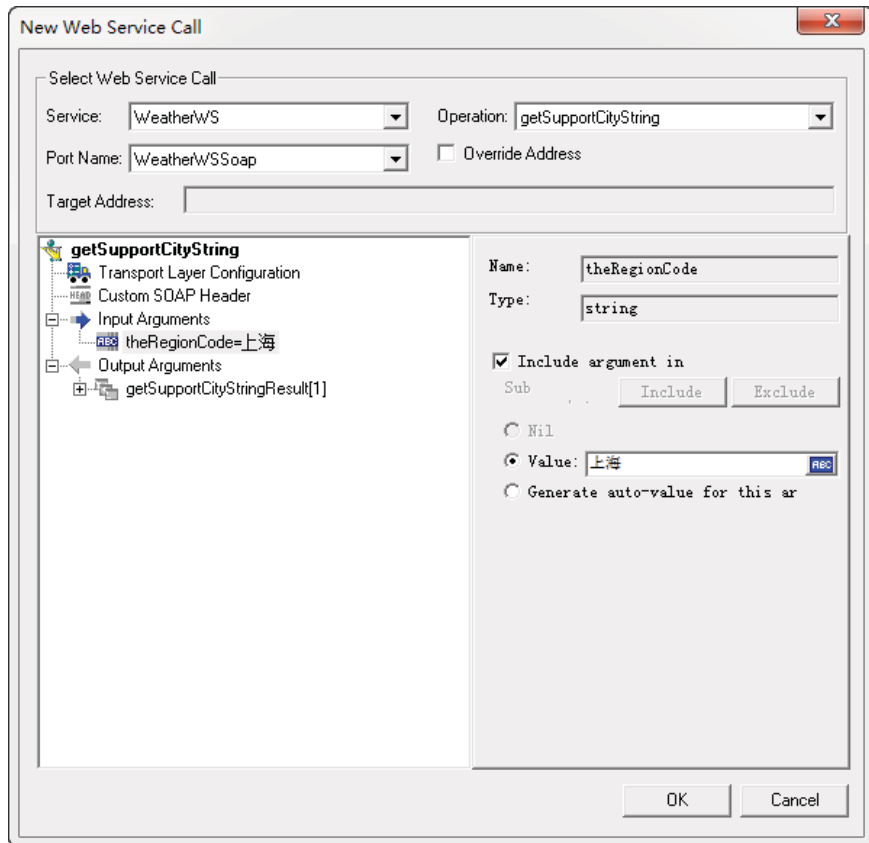


图 8.25 访问 getSupportCityString 服务

单击 OK 按钮完成调用，生成对应的代码：

```
web_service_call( "StepName=getSupportCityString_101",
    "SOAPMethod=WeatherWS|WeatherWSSoap|getSupportCityString",
    "ResponseParam=response",
    "Service=WeatherWS",
    "ExpectedResponse=SoapResult",
    "Snapshot=t1320732864.inf",
    BEGIN_ARGUMENTS,
    "theRegionCode=上海",
    END_ARGUMENTS,
    BEGIN_RESULT,
    END_RESULT,
    LAST);
```

对于 `Web_service_call` 函数，自身就提供了关联返回的功能，该请求的返回会保存在参数 `response` 和 `getSupportCitystring_101_response` 中。我们启动日志运行该函数检查服务器返回，代码如下：

```
Action.c(4): Notify: Saving Parameter "getSupportCityString_101_Response =
<getSupportCityString><getSupportCityStringResult>
<string>宝山,2009</string><string>崇明,2012</string><string>奉
贤,2063</string><string>嘉定,2011</string><string>金山,3530</string><string>闵
行,2008</string><string>南汇,2014</string><string>浦东,2015</string><string>青
浦,2061</string><string>上海,2013</string><string>松江,3413</string><string>徐
家汇,3643</string>
```

```
</getSupportCityStringResult></getSupportCityString>".
    Action.c(4): Notify: Saving Parameter "response = <?xml version="1.0"
encoding="utf-8"?><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:Body><getSupportCityStrin
gResponse xmlns="http://WebXml.com.cn/"><getSupportCityStringResult>
<string>淪濱北,2009</string><string>宕困罾,2012</string><string>濂爰
搯,2063</string><string>錫文鼎,2011</string><string>閱賤
北,3530</string><string>闌佃 ,2008</string><string>錫格
联,2014</string><string>嫫一篋,2015</string><string>闌椒
鄣,2061</string><string>涓嫫搗,2013</string><string>鍬炬
曠,3413</string><string>寰愬 姘\x87,3643</string>
</getSupportCityStringResult></getSupportCityStringResponse></soap:Body></so
ap:Envelope>".
```

这里可以看到服务器的返回仍然是XML,如果有前面的XML函数、Flex或者Silverlight脚本开发的基础,这个时候处理返回已经游刃有余了(XML处理部分参考3.6.9节)。我们需要的上海编号为2013,得到这个值需要通过lr\_get\_xml\_value()函数,编写代码:

```
lr_xml_get_values("XML={response}",
"FastQuery=/Envelope/Body/getSupportCityStringResponse/getSupportCityStringR
esult/string[10]", "ValueParam=ParamValue_string", LAST);
//如果需要多返回随机获得城市编号代码修改为
//代码最前定义整形变量 citycount
citycount=lr_xml_get_values("XML={response}",
"Query=/Envelope/Body/getSupportCityStringResponse/getSupportCityStringResul
t/string", "ValueParam=ParamValue_string", "SelectAll=yes", LAST);
lr_save_int(citycount, "ParamValue_string_count");
lr_save_string(lr_paramarr_random("ParamValue_string"), "ParamValue_string"
);
```

参数ParamValue\_string中返回了“上海,2013”字符串,我们需要进一步得到2013,这个时候有两种方式来处理。一种方法是通过strtok来根据逗号分隔,后面的就是2013;第二种方法是2013的编号应该是定长度4位的,那么整个字符串长度的最后4位就是我们需要的内容。这里使用后面这种方式来解决该问题,代码为:

```
//代码最前定义指针字符 citystring 和整形变量 stringlen
citystring=lr_eval_string("{ParamValue_string}");
stringlen=strlen(citystring)-4;
lr_save_var(citystring+stringlen,4,0,"citycode");
```

这样在参数citycode中就存放了我们需要的2013城市编号。接着我们要查看该城市的天气情况,查看城市天气需要调用GetWeather服务,继续添加一个新的Service Call,调用时传递生成的城市编号,代码如下:

```
web_service_call( "StepName=getWeather_101",
    "SOAPMethod=WeatherWS|WeatherWSSoap|getWeather",
    "ResponseParam=response",
    "Service=WeatherWS",
```

```
"ExpectedResponse=SoapResult",  
"Snapshot=t1320734126.inf",  
BEGIN_ARGUMENTS,  
"theCityCode={citycode}",  
END_ARGUMENTS,  
BEGIN_RESULT,  
END_RESULT,  
LAST);
```

运行该代码我们可以看到该城市编号的天气已经返回保存在参数中了。

### 8.6.2 基于 SOAP 的调用

简单对象访问协议（SOAP）是一种轻量的、简单的、基于 XML 的协议，它被设计在 Web 上交换结构化的和固化的信息。

SOAP 可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（HTTP）、简单邮件传输协议（SMTP）、多用途网际邮件扩充协议（MIME），它还支持从消息系统到远程过程调用（RPC）等大量的应用程序。

SOAP 是另一种常用的 Web Service 调用方式，在使用的时候也需要先导入 SOAP 信息。相对于 WSDL 来说，SOAP 的导入更为麻烦，首先你需要知道请求地址及 SOAP 请求结构，如果不知道可以通过 WSDL 来生成 SOAP，这里推荐使用工具 SoapUI 来帮助我们生成 SOAP。

安装 SoapUI 后使用试用版本，然后新建一个项目，在项目中输入我们要访问的 Web Service WSDL 地址，如图 8.26 所示。

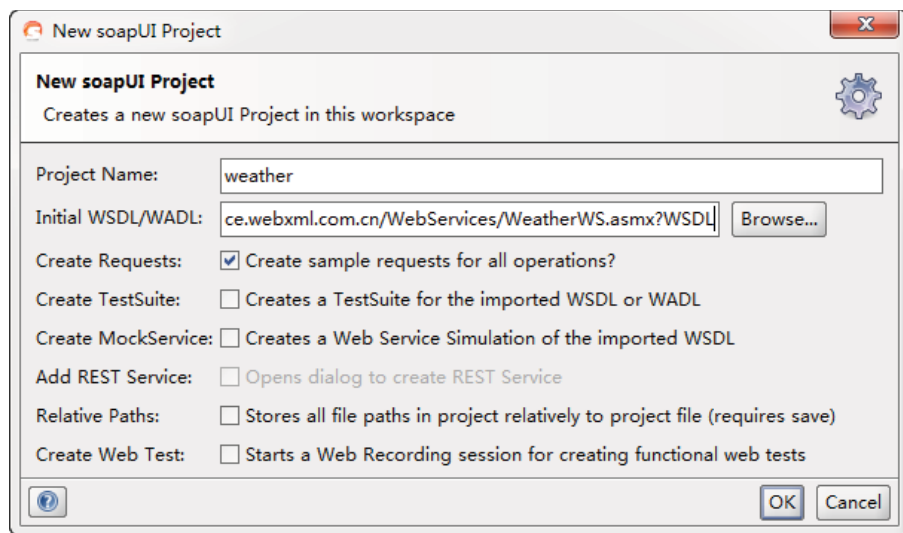


图 8.26 使用 SoapUI 载入 Web Service

确定后双击我们需要访问的 `getSupportCityString` 方法，在右侧的 XML 中可以看到完整的 SOAP 请求，如图 8.27 所示。

这里将该 XML 文件保存下来，接着我们在 Vugen 中导入这个 XML 文件。在 Vugen 中新建一个 Web Service 脚本，选择 SOA Tools 菜单下的 Import SOAP，将刚才从 SoapUI 中导出的 XML 文件导入，如图 8.28 所示。



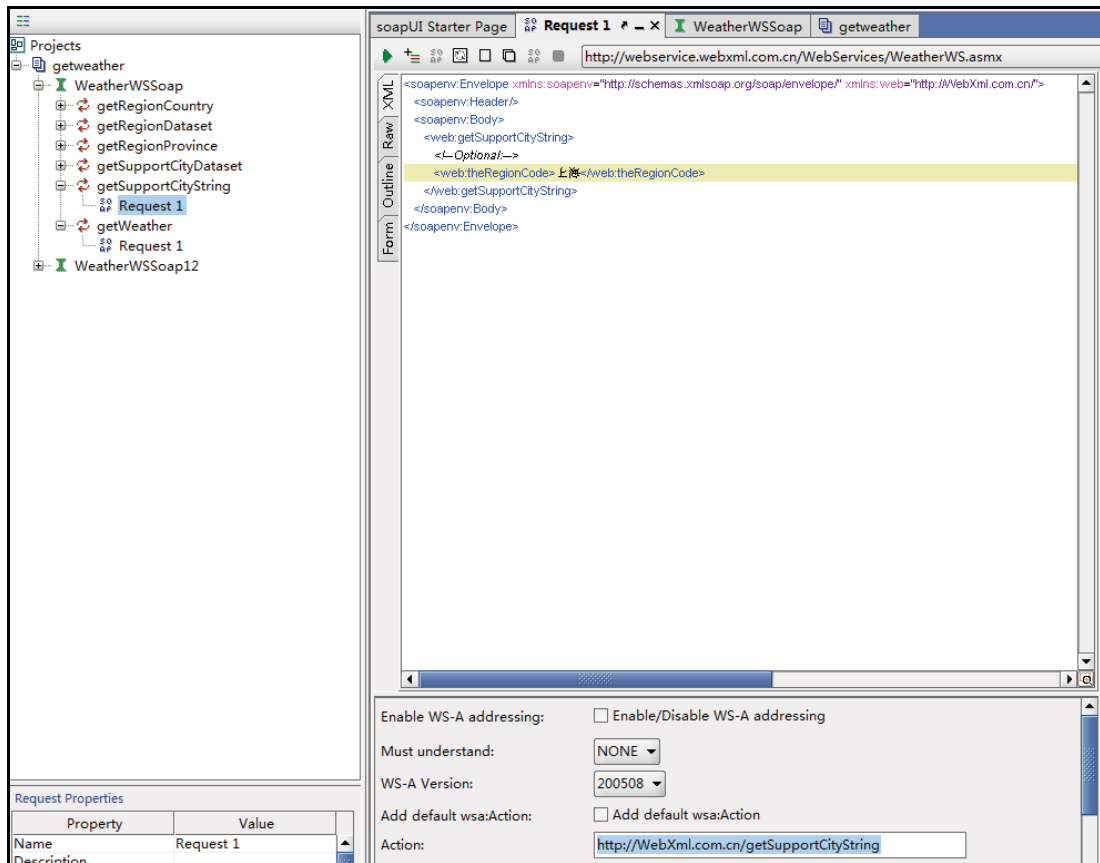


图 8.27 调用 getSupportCityString 的 SOAP

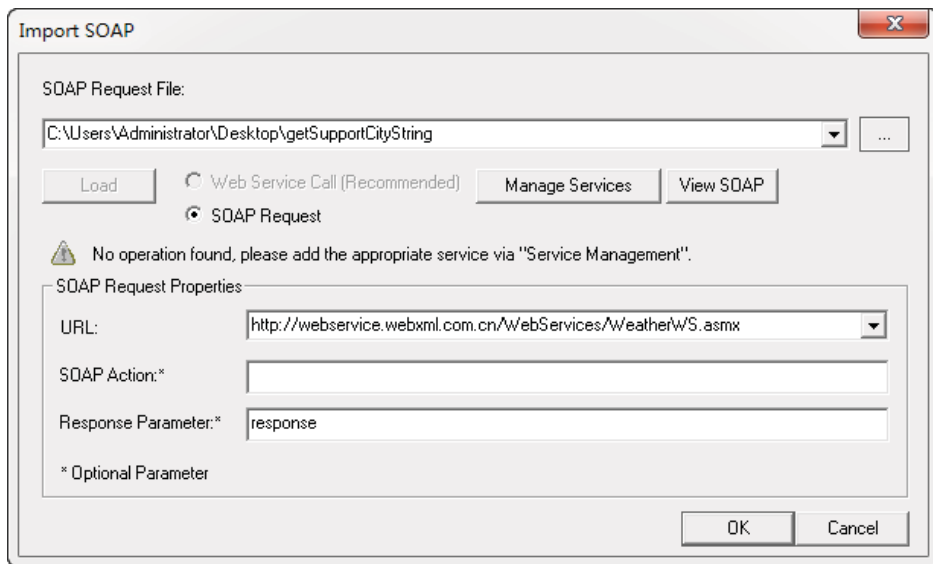


图 8.28 在 Vugen 中导入 SOAP

这里的 URL 需要自己填写，地址就是 SoapUI 中地址栏显示的内容，SOAP Action 可以填也可以不填，如果需要填写，需要在 SoapUI 中查看该请求的 Action，并且复制到这里。单击 OK 按钮后形成 SOAP 请求函数代码：

```
soap_request("StepName=SOAP Request",
    "URL=http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx",
```

```

        "SOAPEnvelope="
        "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:web=\"http://WebXml.com.cn/\">\"
        "<soapenv:Header></soapenv:Header>\"
        "<soapenv:Body>\"
            "<web:getSupportCityString>\"
                "<web:theRegionCode?></web:theRegionCode>\"
            "</web:getSupportCityString>\"
        "</soapenv:Body>\"
    "</soapenv:Envelope>\",
    "SOAPAction=",
    "ResponseParam=response",
    "Snapshot=t1320736948.inf",
    LAST);

```

这里的<web:theRegionCode?></web:theRegionCode>是需要输入的内容，也就是我们需要查询的城市编号，所以修改问号为“上海”，运行代码检查返回：

```

Action.c(4): Notify: Saving Parameter "response = <?xml version="1.0"
encoding="utf-8"?><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:Body><getSupportCityStrin
gResponse xmlns="http://WebXml.com.cn/"><getSupportCityStringResult>
    <string>鏼犺煶甯\x82,000000</string>
    </getSupportCityStringResult></getSupportCityStringResponse></soap:Body>
</soap:Envelope>".

```

我们会发现返回并没有出现前面标准的城市信息，而是返回了“鏼犺煶甯\x82,000000”，转码后是无城市\x00,000000，也就是说我们发送的内容服务器能够接受但是没有匹配记录。在 XML 数据包中由于 LR 无法按照 XML 说明中的格式进行转化，经常会出现由于错误转码而无法完成业务的情况，在这里也是因为这个问题。

```

    lr_convert_string_encoding("鏼犺煶甯\x82",
LR_ENC_UTF8 ,LR_ENC_SYSTEM_LOCALE,"test");
    lr_convert_string_encoding("上海",
LR_ENC_SYSTEM_LOCALE,LR_ENC_UTF8,"test");

```

这两行代码在我们开发 XML 请求时会经常用到，帮助我们把 UTF-8 的格式换成系统格式，或把系统当前格式转换为 UTF-8 格式。

这里我们需要得到“上海”的 UTF-8 编码，然后将编码后的内容作为函数 soap\_request 的一部分发送给服务器，修改代码为：

```

    soap_request("StepName=SOAP Request",

        "URL=http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx",

        "SOAPEnvelope="
        "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"

```

```

xmlns:web=\"http://WebXml.com.cn/\">
    "<soapenv:Header></soapenv:Header>"
    "<soapenv:Body>"
        "<web:getSupportCityString>"
            "<web:theRegionCode>涓涓涓涓</web: theRegionCode>\"//上海
        "</web:getSupportCityString>"
    "</soapenv:Body>"
"</soapenv:Envelope>",
// "SOAPAction=http://WebXml.com.cn/WeatherWSSoap/getSupportCityStr
ingRequest",
"ResponseParam=response", "Snapshot=t1320736948.inf", LAST);

```

检查返回和前面 WSDL 的方式结果相同，同样的方式可以完成后面的数据分离，该城市的天气情况查询，具体内容这里就不详述了。

在这两种 Web Service 脚本开发中，WSDL 导入方式是开发最为简单的方式，基于 SOAP 的方式就相对来说自己要处理很多东西，但是当没有 WSDL 文件时，SOAP 基本是唯一的实现方式。这两种模式都是需要对应的 License。如果只有 Web 的 License 怎么办呢？既然 SOAP 是用 HTTP 协议的 XML 数据格式，那么我们使用 HTTP 协议也是可以模拟的。接着我们来看看如何使用 HTTP 协议完成 Web Service 调用。

### 8.6.3 基于 HTTP 的调用

基于 HTTP 协议的 Web Service 调用也有两种模式，第一种是强制发送 XML 数据包模拟 SOAP 请求格式，另一种需要 Web Service 自身支持 HTTP 调用模式。

#### 1. 使用 HTTP 协议进行 SOAP 模拟

对于 SOAP 协议我们前面有所研究，但是并没有说透，对于查询城市编号的这个操作，SOAP 的请求格式其实是这个样子的：

```

POST /WebServices/WeatherWS.asmx HTTP/1.1
Host: webservice.webxml.com.cn
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
    <soap12:Body>
        <getSupportCityString xmlns="http://WebXml.com.cn/">
            <theRegionCode>string</theRegionCode>
        </getSupportCityString>
    </soap12:Body>
</soap12:Envelope>

```

也就是说我们只需要使用 `web_custom_request()` 函数往服务器上的这个地址抛一个 XML 数据包就行了。新建一个 HTTP/HTML 协议的脚本，在里面通过 `web_custom_request()` 函数发送对应的数据包，代码如下：

```

web_custom_request("web_custom_request",

    "URL=http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx",
    "Method=POST",
    "TargetFrame=",
    "Resource=0",
    "Referer=",
    "Body=<?xml version=\"1.0\" encoding=\"utf-8\"?>"
    "<soap12:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema"
    "-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" xmlns:soap12="
    "\"http://www.w3.org/2003/05/soap-envelope\">"
    "    <soap12:Body>"
    "        <getSupportCityString xmlns=\"http://WebXml.com.cn/\">"
    "            <theRegionCode>string</theRegionCode>"
    "        </getSupportCityString>"
    "    </soap12:Body>"
    "</soap12:Envelope>",
    LAST);

```

但是这个数据包直接运行后会得到服务器返回的错误:

```

Action.c(17): Error -26616: HTTP Status-Code=415 (Unsupported Media Type)
for "http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx" [MsgId:
MERR-26616]

```

这个错误的原因是我们发送的数据包没有明确格式类型, 服务器在校验格式时出现错误, 所以我们需要手动添加一个 Header 头部分, 使用函数 `web_add_header()` 添加 `Content-Type: application/soap+xml; charset=utf-8` 头信息:

```

web_add_header("Content-Type", "application/soap+xml; charset=utf-8");

```

添加这行代码后再次运行, 请求成功发送, 返回的内容可以通过 `web_reg_save_param_*` 关联函数解决, 将 `<theRegionCode>string</theRegionCode>` 中的 `string` 替换为上海的 UTF-8 编码, 请求前添加头信息, 编写关联函数即可完成该 Web Service 的调用, 完整代码如下:

```

web_add_header("Content-Type",
    "application/soap+xml; charset=utf-8");

web_reg_save_param_ex(
    "ParamName=getcityid",
    "LB=",
    "RB=</string>",
    "Ordinal=ALL",
    SEARCH_FILTERS,
    "Scope=BODY",
    LAST);

web_custom_request("web_custom_request",

```

```

        "URL=http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx",
        "Method=POST",
        "TargetFrame=",
        "Resource=0",
        "Referer=",
        "Body=<?xml version=\"1.0\" encoding=\"utf-8\"?>"
        "<soap12:Envelope
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:soap12=\"http://www.w3.org/2003/05/soap-envelope\">"
        "  <soap12:Body>"
        "    <getSupportCityString xmlns=\"http://WebXml.com.cn/\">"
        "      <theRegionCode>涓涓掳搯</theRegionCode>//上海 UTF-8
        "    </getSupportCityString>"
        "  </soap12:Body>"
        "</soap12:Envelope>",
        LAST);

```

检查返回可以看到成功关联到和上海有关的 12 个区域编号，后续开发略。

## 2. 使用 HTTP 协议直接访问 Web Service

模拟 SOAP 的方式其实还是比较复杂的，当 Web Service 自身支持的时候我们可以使用更简单的调用方式来解决。HTTP 支持可以通过 GET 或 POST 完成对 Web Service 的调用，例如调用城市编号这个方法时我们可以通过这两种方式来实现（GET 方式相对来说更简单点）：

```

//GET 写法
web_url("getscs","URL=http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx/getSupportCityString?theRegionCode=涓涓掳搯",LAST);
//POST 写法
web_submit_data("getscs",
"Action=http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx/getSupportCityString",
    "Method=POST",
    "Mode=HTTP",
    ITEMDATA,
    "Name=theRegionCode", "Value=上海", ENDITEM,
    LAST );

```

这里需要注意的是，当使用 `web_submit_data` 函数时，如果系统回放选择支持 UTF-8 转码（参考 3.5.9 节中的运行设置），那么这里是可以写中文的，否则还是要像 GET 写法，使用转码后的内容。通过 `web_reg_save_param_*` 关联函数同样可以对返回进行处理，后续开发略。

使用 HTTP 协议模式完成对 Web Service 的调用并没有什么神奇，只是逐渐深入，越来越涉及本质协议而已。但在实际项目中，使用 WSDL 方式进行脚本开发的效率是最高的，如果无法解决 License 问题必须使用 HTTP 协议时，需要对数据包结构有很深的了解，还可能需要在开发上提供一些支持，便于直接使用 GET 模式调用。

上面提到了三大类调用方式，接着来看一下更加本质的做法，基于 Windows Sockets 调用开发。这里只是给大家再拓展一下知识，并没有什么实用价值，而 Sockets 开发请先熟悉了 8.7 节的内容后再来看本章节。

#### 8.6.4 基于 Windows Sockets 的调用

使用 Sockets 的开发其实并不复杂，在了解了可以使用 HTTP 协议完成模拟后，我们可以使用针对浏览器直接录取 Sockets 数据包的方式来完成脚本开发。

新建一个 Windows Sockets 的脚本录制浏览器访问 Web Services 中的操作，请求地址为：

```
http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx/getSupportCityString?theRegionCode=涓涓涓
```

由于 LR11 的 Windows Sockets 协议不支持 Windows 2008 R2 下的 IE 9，有数据包但是无法生成脚本，这里的代码是在 Windows 2003 下的 IE 8 中录制的（回放在 Windows 2008 R2 下测试通过），得到的代码及 Data.ws 文件分别是：

```
vuser_init()
{
    lrs_startup(257);

    lrs_create_socket("socket0", "TCP", "LocalHost=0", "RemoteHost=61.147.124.120:80", LrsLastArg);
    lrs_send("socket0", "buf0", LrsLastArg);
    lrs_receive("socket0", "buf1", LrsLastArg);
    return 0;
}

;WSRData 2 1
send buf0 391
    "GET
/WebServices/WeatherWS.asmx/getSupportCityString?theRegionCode=涓涓涓 "
    "HTTP/1.1\r\n"
    "Accept: */*\r\n"
    "Accept-Language: zh-cn\r\n"
    "User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Trident/4.0"
    "; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR "
    "3.5.30729; .NET4.0C; .NET4.0E)\r\n"
    "Accept-Encoding: gzip, deflate\r\n"
    "Host: webservice.webxml.com.cn\r\n"
    "Connection: Keep-Alive\r\n"
    "\r\n"

recv buf1 819
    "HTTP/1.1 200 OK\r\n"
    "Date: Tue, 08 Nov 2011 13:45:26 GMT\r\n"
    "Server: Microsoft-IIS/6.0\r\n"
    "X-Powered-By: ASP.NET\r\n"
```



```

        "X-AspNet-Version: 2.0.50727\r\n"
        "Cache-Control: private, max-age=0\r\n"
        "Content-Type: text/xml; charset=utf-8\r\n"
        "Content-Length: 589\r\n"
        "\r\n"
        "<?xml version=\"1.0\" encoding=\"utf-8\"?>\r\n"
        "<ArrayOfString
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xml"
        "ns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns=\"http://WebXml.com.cn/\">
        >\r\n"
        "  <string>淪漬北,2009</string>\r\n"
        "  <string>宕困罾,2012</string>\r\n"
        "  <string>濂爰搗,2063</string>\r\n"
        "  <string>緡爰晶,2011</string>\r\n"
        "  <string>閱賤北,3530</string>\r\n"
        "  <string>闌佃魑,2008</string>\r\n"
        "  <string>錦格朕,2014</string>\r\n"
        "  <string>嫻へ篋,2015</string>\r\n"
        "  <string>闌椒郇,2061</string>\r\n"
        "  <string>涓娉搗,2013</string>\r\n"
        "  <string>鍬炬睔,3413</string>\r\n"
        "  <string>寰愬揆姘"
        "\x87"
        ",3643</string>\r\n"
        "</ArrayOfString>"

```

-1

我们发送的请求是 BUF0，而返回的数据包为 BUF1。通过参数化 BUF0 中的“theRegionCode=涓娉搗”段，然后发送即可完成对服务器不同参数的调用，而服务器返回的部分需要使用 lrs\_save\_searched\_string()函数来解决，在代码中添加关联函数：

```

lrs_save_searched_string("socket0",LRS_LAST_RECEIVED,"param1","LB=<string>","RB=</string>",1,0,-1);

```

这样就可以得到第一个符合边界的城市区域及编号，如果需要多返回并且随机，则要修改为：

```

lrs_save_searched_string("socket0",LRS_LAST_RECEIVED,"param1","LB=http://WebXml.com.cn/\">\r\n  <string>","RB=</string>\r\n</ArrayOfString>",1,0,-1);

```

通过上面这种关联（注意边界中的空格），可以得到一个这样的数据包：

```

淪漬北,2009</string>\r\n<string>宕困罾,2012</string>\r\n<string>濂爰搗,2063</string>\r\n<string>緡爰晶,2011</string>\r\n<string>閱賤北,3530</string>\r\n<string>闌佃魑,2008</string>\r\n<string>錦格朕,2014</string>\r\n<string>嫻へ篋,2015</string>\r\n<string>闌椒郇,2061</string>\r\n<string>涓娉搗,2013</string>\r\n<string>鍬炬睔,3413</string>\r\n<string>寰愬揆姘\x87,3643

```

接着我们使用函数 splitchartoparamarr()（该函数为自定义函数，使用方法参考 8.7 节）来分离该字符串，这里不能使用 XML 的处理方式，在 Sockets 协议中不能使用

lr\_xml\_get\_values()函数，代码如下：

```
lr_convert_string_encoding(lr_eval_string("<param1>"),LR_ENC_UTF8,LR_ENC_SYSTEM_LOCALE,"temp");

splitchartoparamarr(lr_eval_string("<temp>"), "</string>\\r\\n <string>", "myarr");
```

这里首先将 lrs\_save\_searched\_string()关联得到的 param1 参数进行变化转换，从 UTF-8 转换为系统编码，再对转码后的字符串按照</string>\\r\\n <string>这个字符串进行分隔（\\n 和<string>之间有两个空格），最终就可以得到一个叫做 myarr 的参数列表，便于我们进行随机选择并且进一步分离了（在这个例子中最后一个参数取值是错误的，本来应该是“徐家汇,3643”，但是在 UTF-8 转码中错误的转化为“徐家?\x87,3643\x00”），后面的开发就和普通的多返回随机选择再发送类似了，这里就不详细介绍了。

也可以使用 HTTP+Sockets 协议来解决无法使用 lr\_xml\_get\_values()的问题，首先需要使用 Sockets 录制得到标准的 data.ws 和请求脚本，接着新建一个空的 HTTP+Sockets 的多协议脚本。再将前面 Sockets 脚本目录中的 data 目录复制到这个新脚本目录中，这个时候 Tools 菜单下的 Regenerate Script 变为可用模式，点击重新构建脚本。构建完成后就会出现 Sockets 脚本必须的 data.ws 文件引用，然后我们把前面 Sockets 代码贴入到对应的文件中即可。

由于 HTTP 协议中的参数边界是左右花括弧，所以在后面的代码编写中需要注意当前的边界符号，避免参数取值的错误，代码如下：

```
char *xml;

lrs_startup(257);

lrs_create_socket("socket0", "TCP", "LocalHost=0",
"RemoteHost=61.147.124.120:80", LrsLastArg);

lrs_send("socket0", "buf0", LrsLastArg);

lrs_receive("socket0", "buf1", LrsLastArg);

lrs_save_searched_string("socket0",LRS_LAST_RECEIVED,"param1","LB=
http://WebXml.com.cn/\">\\r\\n\", \"RB=\\r\\n</ArrayOfString>\",1,0,-1);

xml=lr_eval_string("<?xml version=\\\"1.0\\\"
encoding=\\\"utf-8\\\"?><root>{param1}</root>");

lr_save_string(xml,"param1");

lr_xml_get_values( "XML={param1}",
"ValueParam=OutputParam",
"Query=/root/string[1]",
```

```
LAST);
```

在这个代码中，使用了 `lr_eval_string()` 函数直接构建 XML 结构的做法，为 `param1` 参数值弥补了根节点 `root` 和 XML 编码版本说明信息，然后再使用 `lr_xml_get_values` 取值（在使用 XML 的处理规则时仍然会出现“徐家汇”节点编码错误的问题，导致取值失败）。

这里如果需要本质解决“徐家汇”节点编码错误的问题，最有效的方法是修改系统区域设置中的非 Unicode 程序使用的语言，把中文修改为英文，这样所有汉字都会变成十六进制的双字节编码，“徐家汇”变为 `\xe5\xbe\x90x\xe5xae\xb6\xe6\xb1\x87`，这样再去关联就不会有任何问题了，如果需要从十六进制 Unicode 转回汉字那就比较复杂了，这里提供一个 PHP 的代码便于快速查询，但在真实工作中尽量避免这种复杂的转换策略带来的成本开销。

Unicode.php 代码如下：

```
$code=$_GET['string'];
echo urldecode(str_replace('/x', '%', $code));
$string = '';
foreach (str_split($code, 4) as $char) {
    $string .= chr(hexdec(substr($char, 2)));
}
echo $string;
```

使用该代码时在浏览器中访问 `unicode.php?string=\xe5\xbe\x90x\xe5xae\xb6\xe6\xb1\x87` 即可，显示的文本就是 UTF-8 编码的中文。

### 8.6.5 扩展 Oracle 数据库性能测试

在 Web Service 协议中，可以使用 LR 内置的数据库连接函数 `lr_db_connect()` 来完成对各种常见数据库的连接，从而进一步通过场景并发进行数据库性能测试。

`lr_db_connect()` 函数默认支持四种数据库连接模式：

- SQL（原生 MS SQL Server）。
- OLEDB（使用 OLEDB 连接数据库）。
- ODBC。
- ORACLE。

例如我们需要完成一个对 Oracle 数据库的连接，则可以通过 OLEDB 的方式来完成，代码如下：

```
lr_db_connect("StepName=Connect",
"ConnectionString=Provider=OraOLEDB.Oracle.1, Data Source=ORCL; Server=
127.0.0.1;Persist Security Info=True;User ID=cloudchen;Password=123456",
"ConnectionName=db1",
"ConnectionType=OLEDB",
LAST );
```

在这里 `Data Source` 需要填写连接 Oracle 时需要的 Net Service 名称，而用户名密码需

要填写在 User ID 和 Password 属性中，这里使用的连接方式是 OLEDB。

而通过 `lr_db_executeSQLStatement()` 函数可以对数据库进行 SQL 指令执行。如果想在数据库中执行某条语句查询数据集，并且获得该返回数据集中的部分属性，对其进行响应时间分析从而获得系统数据库的性能时，我们可以这样写：

```
int NumRows=0;
int i;
lr_db_connect("StepName=Connect",
              "ConnectionString=Provider=OraOLEDB.Oracle.1, Data
Source=ORCL; Server=172.168.11.40 ;Persist Security Info=True;User
ID=cloudchen;Password=123456",
              "ConnectionName=db1",
              "ConnectionType=OLEDB",
              LAST );

lr_start_transaction("SQL");

    NumRows = lr_db_executeSQLStatement("StepName=PerformQuery",
    "ConnectionName=db1",
    "SQLStatement=select * from USERS",
    "DatasetName=MyDataset",
    LAST );

    lr_end_transaction("SQL", LR_AUTO); //获得查询记录所开销的时间

    lr_output_message("The query returned %d rows.", NumRows);

while(i<NumRows) {
    lr_db_getvalue("StepName=GetValue",
    "DatasetName=MyDataset",
    "Column=USER_NAME",
    "Row=next",
    "OutParam=MyOutputParam",
    LAST);

    lr_output_message("The value is: %s", lr_eval_string("{MyOutputParam}"));

    i=i+1;
}

lr_db_disconnect("StepName=Disconnect",
                 "ConnectionName=db1",
                 LAST);
```

通过上面的代码可以得到下面的结果：

```
Action.c(25): Notify: Transaction "SQL" ended with "Pass" status (Duration:
0.2789 Wasted Time: 0.2672).
```

整个查询的时间开销是  $0.2789 - 0.2672 = 0.0117$  秒，而在 PL/SQL Developer 中的该 SQL 执行时间（执行 SQL 前先执行 `set timing on` 即可看到执行时间）为 0.016 秒，如图 8.29 所示。

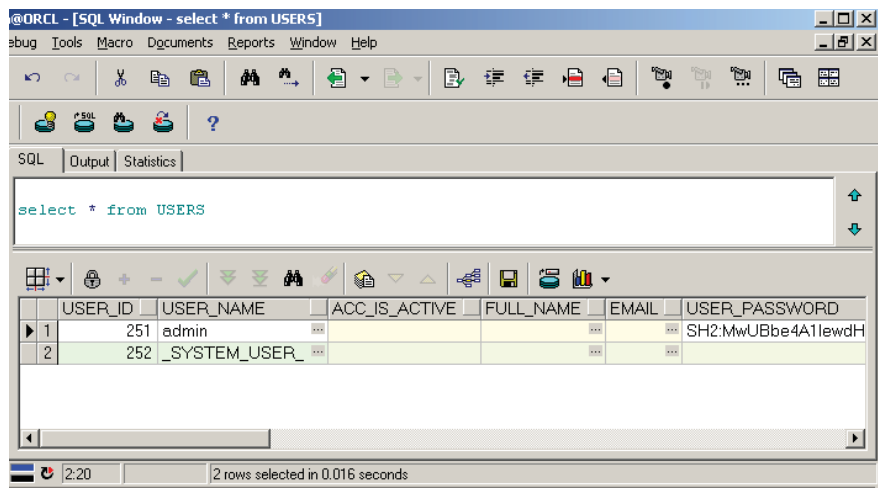


图 8.29 PL/SQL Developer 中的执行计时

如果需要直接使用 ODBC 的方式来完成数据库性能测试，那么可以这样编写：

```
lr_db_connect("StepName=Connect",
"ConnectionString=Driver={SQL
Server};Server=Localhost;Database=cloudsample;Trusted_Connection=yes;",
"ConnectionName=db1",
"ConnectionType=ODBC",
LAST);
```

这里可以按照标准的 ODBC 或 OLEDB 连接写法修改 ConnectionString 串完成对各种数据对象的连接，并且通过添加事务完成响应时间的监控。各种常见 ODBC 及 OLEDB 连接串写法如表 8.1 所示。

表 8.1 常见 ODBC 及 OLEDB 连接串写法

DB	Connection String
Access	<b>Access ODBC Connection String Driver</b> {Microsoft Access Driver (* .mdb) }; Dbq=C:\demo.mdb;Uid=Admin;Pwd=;
	<b>Access OLEDB Connection String Driver</b> Provider=Microsoft.Jet.OLEDB.4.0;Data Source=\directory\demo.mdb;User Id=admin;Password=;
DB2	<b>DB2 ODBC Connection String</b> driver={IBM DB2 ODBC DRIVER}; Database=demodb;hostname=myservername;port=myPortNum;prot ocol=TCPIP; uid=myusername; pwd=mypasswd
	<b>DB2 OLEDB Connection String</b> Provider=IBMDADB2; Database=demodb; HOSTNAME=myservername; PROT OCOL=TCPIP; PORT=50000; uid=myusername; pwd=mypasswd;
DBase	<b>DBase ODBC Connection String</b> Driver={Microsoft dBASE Driver (* .dbf) }; DriverID=277; Dbq=c:\directory;
	<b>DBase OLEDB Connection String</b> Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\directory; Extended Properties=dBASE IV; User ID=Admin; Password=

续 表

Excel	<b>Excel ODBC Connection String</b> Driver={Microsoft Excel Driver (*.xls)};DriverId=790;Dbq=C:\CloudExcel.xls;DefaultDir=c:\directo ry; <b>Excel OLEDB Connection String</b> Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\CloudExcel.xls;Extended Properties="Excel 8.0;HDR=Yes;IMEX=1"
Exchange	<b>Exchange OLEDB Connection String</b> oConn.Provider = "EXOLEDB.DataSource" oConn.Open = "http://myServerName/myVirtualRootName"
Firebird	<b>Firebird ODBC Connection String</b> DRIVER=Firebird/InterBase(r) driver;UID=SYSDBA;PWD=mypasswd;DBNAME=c:\directory\demo.fdb <b>Firebird OLEDB Connection String</b> User=SYSDBA;Password=mypasswd;Database=demo.fdb;DataSource=lo calhost;Port=3050;Dialect=3;Charset=NONE;Role=;Connection lifetime=15;Pooling=true;MinPoolSize=0;MaxPoolSize=50;Packet Size=8192;ServerType=0
FoxPro	<b>FoxPro ODBC Connection String</b> Driver={Microsoft Visual FoxPro Driver};SourceType=DBC;SourceDB=c:\demo.dbc;Exclusive=No;NULL=NO; Collate=Machine;BACKGROUNDFETCH=NO;DELETED=NO <b>FoxPro OLEDB Connection String</b> Provider=vfpoledb.1;Data Source=c:\directory\demo.dbc;Collating Sequence=machine
Informix	<b>Informix ODBC Connection String</b> Driver={Informix-CLI 2.5 (32 Bit)};Server=demoservername;Database=demodb;Uid=myusername;Pwd=my passwd <b>Informix OLEDB Connection String</b> Provider=Ifxoledbc.2;User ID=myusername;password=mypasswd;Data Source=demodb@demoservername;Persist Security Info=true
MySQL	<b>MySQL ODBC Connection String</b> DRIVER={MySQL ODBC 3.51 Driver};SERVER=myservername;PORT=3306;DATABASE=mydemodb; USER=myusername;PASSWORD=mypasswd;OPTION=3; <b>MySQL OLEDB Connection String</b> Provider=MySQLProv;Data Source=mydemodb;User Id=myusername;Password=mypasswd;
Oracle	<b>Oracle ODBC Connection String</b> Driver={Microsoft ODBC for Oracle};Server=myservername;Uid=myusername;Pwd=mypassword; <b>Oracle OLEDB Connection String</b> Provider=msdaora;Data Source=mydemodb;User Id=myusername;Password=mypasswd; <b>Oracle .NET Connection String</b> Data Source=mydemodb;User Id=myusername;Password=mypasswd;Integrated Security=no;



续 表

SQL Server	<b>SQL Server ODBC Connection String - Database Login</b> Driver={SQL Server}};Server=myservername;Database=mydemodb;Uid=myusername;Pwd= mypasswd; <b>SQL Server ODBC Connection String - Trusted Connection</b> Driver={SQL Server}};Server=mysername;Database=mydemodb;Trusted_Connection=yes ; <b>SQL Server OLEDB Connection String - Database Login</b> Provider=sqloledb;Data Source=myservername;Initial Catalog=mydemodb;User Id=myusername;Password=mypasswd; <b>SQL Server OLEDB Connection String - Trusted Connection</b> Provider=sqloledb;Data Source=myservername;Initial Catalog=mydemodb;Integrated Security=SSPI; <b>SQL Server .NET Connection String - Database Login</b> Server=myservername;Database=mydemodb;User ID=myusername;Password=mypasswd;Trusted_Connection=False <b>SQL Server .NET Connection String - Trusted Connection</b> Server=myservername;Database=mydemodb;Integrated Security=SSPI;
Sybase	<b>Sybase ODBC Connection String</b> Driver={SYBASE ASE ODBC Driver}};Srvr=myservername;Uid=myusername;Pwd=mypasswd <b>Sybase OLEDB Connection String</b> Provider=Sybase.ASEOLEDBProvider;Server Name=myservername,5000;Initial Catalog=mydemodb;User Id=myusername;Password=mypassword

除了使用 LR 自带数据库处理函数完成数据库性能测试的方法外,我们也可以使用 Java Vuser 或者 .NET Vuser 来完成数据库的连接性能测试,相关内容参考 8.10.1 和 8.11.1 节。

## 8.10 .NET Vuser

当我们测试基于微软 .NET 技术的网站或者应用时都是通过协议的方式对服务器产生负载的,但是这种方式不足以支持代码级别的定位,当使用 .NET Vuser 协议时,我们可以使用该语言与 LR 进行代码的负载,从本质上可以解决很多问题。

在 LR 11 的 Patch2 中提供了对 VS 2010 的支持,我们在 Vugen 中创建的 .NET Vuser 可以在 VS 2010 中开发了。在安装过程中需要注意安装顺序,否则可能会导致 VS 2010 转换过的代码无法在 Vugen 中运行的问题。个人推荐先安装 VS 2010 英文版,接着安装 LR 11,再依次安装 Patch1、Patch2 补丁,避免不必要的问题。安装盘中提供了 VS 2008 IDE 插件,这个插件必须要在 VS 2008 存在的情况下才能使用,这里可以不用安装,安装后可以在 VS 中直接创建 LR Vuser 脚本。

在 Vugen 中新建一个 Microsoft .NET 协议脚本,然后直接保存。在 VS 2010 中打开脚本目录中的 script.sln 项目文件,VS 2010 会提示该项目需要升级为新的项目格式,确认完

成，接着我们就可以看到在 VS 中已经可以对脚本进行开发了，如图 8.35 所示。

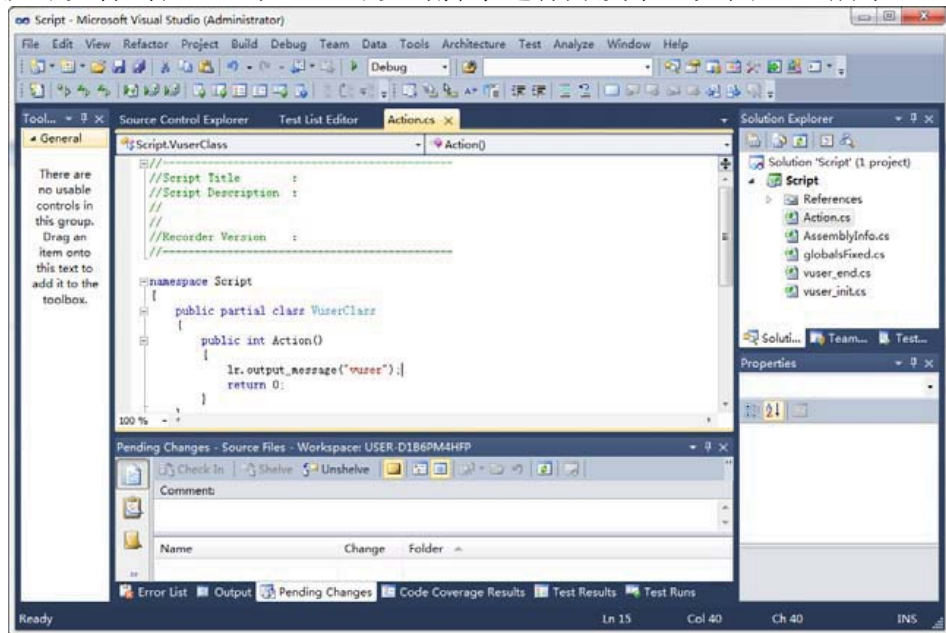


图 8.35 在 VS 2010 中开发.NET 用户脚本

我们在 Action 中编写一个函数，在 C#语法中所有的函数首关键字要用点号说明，所以以前在 C 语言中的 lr\_output\_message()函数现在要写成 lr.output\_message()。代码编写完成编译后单击“保存”按钮，接着切回 Vugen 会提示代码已经更新，我们确认重新加载所有内容，这个时候就能在 Vugen 中看到在 VS 中编写的代码，接着运行这个代码，可以看到出现了我们期待的日志信息。

.NET 的最大作用就是扩展了脚本编写的空间，我们可以通过 C#语言完成很多以前无法完成的操作，而使用 LR 来完成负载和响应时间监控。比如这里需要测试一下在 C#中如果做一个整型数据从 1 累加到 10000 到底需要多少时间的性能测试，我们现在就可以轻松实现。在 VS 2010 中编写以下代码：

```
public int Action()
{
    int i,t;
    t = 0;
    lr.start_transaction("for");
    for (i = 0; i < 10000; i++)
    {
        t = t + i;
    }
    lr.end_transaction("for", lr.AUTO);
    lr.output_message(t.ToString());
    return 0;
}
```

回到 Vugen 中运行可以看到事务 for 的事务时间：

```
Notify: Transaction "for" started.
Notify: Transaction "for" ended with "Pass" status (Duration: 0.0006).
```

在整个脚本开发中 VS 提供了开发工具及编译支持，所以我们必须在 VS 中完成编译工

作，脚本会被编译成一个.dll 动态链接库的形式存放在脚本目录的 bin 目录下。

在 Vugen 中运行 Microsoft .NET 脚本经常会出现 MSBuild 不支持版本的错误，就是因为没有在 VS 中完成编译导致的。

### 8.10.1 使用.NET Vuser 测试 SQL Server 2008 数据库性能

我们使用 .NET 来测试一下使用 ADO.NET 连接 SQL Server 2008 读取一张表的响应时间。

使用 VS 2010 开打 Vugen 新建的项目，完成转换格式后，首先在项目上选择属性，修改编译目标版本从 2.0 升级到 3.5（4.0 的编译对象无法在 Vugen 中运行）。接着为该项目添加 system、system.data、system.xml 的对象引用。编写下面的代码：

```
using System;
using System.Data.SqlClient;
using System.Data;
namespace Script
{
    public partial class VuserClass
    {
        public int Action()
        {
            string title;
            title="cloud";

            lr.start_transaction("ado");

            SqlConnection con = new SqlConnection("server='(local)';
database='cloudsample';uid='sa';pwd='51testing'");
            con.Open();    //打开数据连接

            string strsql = "select * from webservicetable where chartitle like
'" + title + "%'";    //生成 SQL 语句

            lr.start_transaction("search");

            SqlDataAdapter da = new SqlDataAdapter(strsql, con);
            //创建适配器

            lr.end_transaction("search", lr.AUTO);

            DataSet ds = new DataSet();    //创建数据集

            int i = da.Fill(ds, "mytable");    //填充数据集

            con.Close();

            lr.end_transaction("ado", lr.AUTO);

            return 0;
        }
    }
}
```

```
}  
}
```

在数据库的表中有 10000 条随机记录，当我们运行该脚本时就可以得到准确的 SQL 执行时间和.NET 部分 ADO 连接时间，Vugen 日志如下：

```
Notify: Transaction "ado" started.  
Notify: Transaction "search" started.  
Notify: Transaction "search" ended with "Pass" status (Duration: 0.0006).  
Notify: Transaction "ado" ended with "Pass" status (Duration: 0.0984).
```

可以看到查询数据库的时间只有 0.0006 秒，而 ADO 连接是整个查询中开销最大的部分，同时我们还可以在 SQL Server Management Studio 中对该语句做一个单独的查询分析，结果如图 8.36 所示。

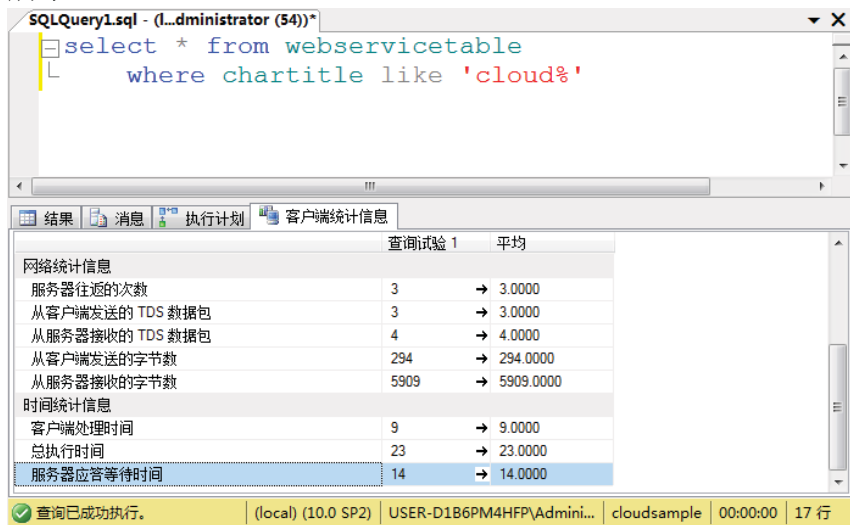


图 8.36 查看 SQL 语句的客户端统计

通过 SQL 执行客户端统计信息可以看到执行一次这样的 SQL 语句需要开销的时间大概在 14 毫秒左右；和我们通过 ADO 连接得到的事务时间基本接近。在确认脚本的正确性后我们可以使用 Controller 进行多用户负载，得到系统在面临多用户 ADO 连接时数据库端和应用端的响应时间，从而定位性能瓶颈。

所以使用.NET Vuser 用户开发脚本相当于我们自己重写了一个富客户端，从而可以方便地在其中插入各种探针和分离各种干扰因素。

### 8.10.2 使用.NET Vuser 测试 C# 类库

最后我们来看一个如何对一个已经开发完成的模块进行接口级别的单元性能测试。首先创建一个 ClassLibrary 脚本，在脚本里面新建一个自己的方法，接着将这个脚本编译成为一个 DLL 文件，等待后边的.NET Vuser 调用。

接着新建一个自己的.NET Vuser 脚本，然后使用 VS 打开转换格式后编译保存。然后我们在引用中添加需要调用的 DLL 库文件，如图 8.37 所示。

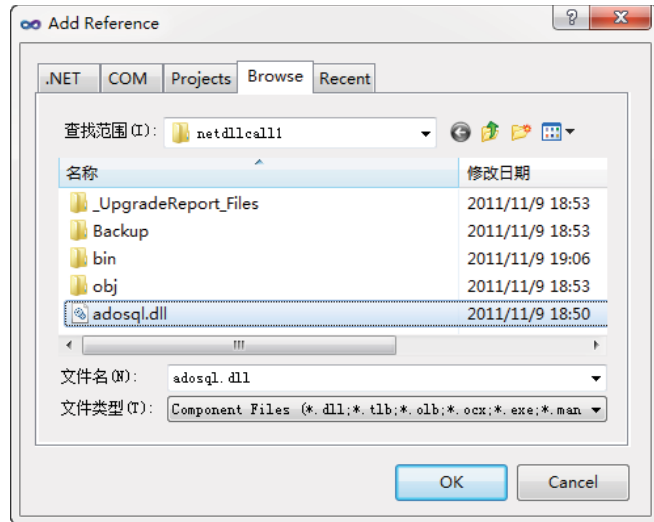


图 8.37 添加引用 DLL 库

在代码中添加对该库中类的实例化，如果不知道 DLL 中的类名或者方法名，那么可以在 VS 中双击引用的对象名，在弹出的对象浏览器中可以看到该 DLL 中的类及方法，如图 8.38 所示。

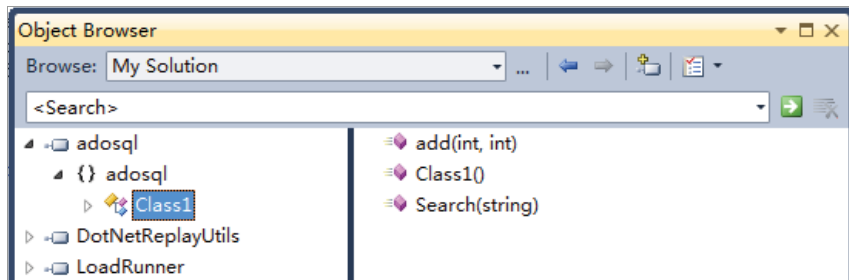


图 8.38 被调用的 adosql 库中的类及方法名

接着编写代码实例化（还需要添加 system、system.xml、system.data 引用），代码如下：

```
using System.Data;
using adosql;
namespace Script
{
    public partial class VuserClass
    {
        public int Action()
        {
            DataSet ds = new DataSet();
            Class1 lrtest = new Class1();
            lr.start_transaction("dllcall");
            ds = lrtest.Search("cloud");
            lr.end_transaction("dllcall", lr.AUTO);
            return 0;
        }
    }
}
```

将该代码编译后，切回 Vugen 运行得到 Class1 类下调用 search 方法的执行时间：

Starting action Action.

```
Notify: Transaction "dllcall" started.
Notify: Transaction "dllcall" ended with "Pass" status (Duration: 0.0737).
Ending action Action.
```

代码写到这里大家应该明白对于.NET 开发的项目如何进行进一步的性能测试了吧。以前做的性能测试往往都是系统级别的，很难定位到应用层还是数据层的问题，就算定位到了应用层，由于调用过于复杂很难定位。而使用.NET Vuser 可以帮助我们针对每一个类、方法进行更加底层的性能测试，从而抽丝剥茧最终定位到病因。能够完成同样工作的还有很多开发工具，如何选择就看你自己了。

## 8.11 Java Vuser

在上面的章节中我们发现使用.NET 协议可以帮助我们解决很多问题，既然.NET 可以这样写，Java 当然也能这样写。

我们接着来看看如何使用 Java Vuser 来解决 Java 项目开发中的测试问题。在 Vugen 中创建一个 Java Vuser 脚本，在这个新建的脚本中可以直接编写 Java 代码了，由于还没有 Eclipse 的 IDE 插件，Java Vuser 的脚本还不能像.NET Vuser 脚本一样在开发工具中进行开发。

我们在代码的 Action 中编写一个基本的输入语句，代码如下：

```
public int action() throws Throwable {

    System.out.println("java Vuser");

    return 0;
} //end of action
```

运行该代码可以看到正确地输出了 java Vuser 字符串。接着我们继续测试一下在 Java Vuser 中做一个从 1 累加到 10000 的操作到底要多少时间的性能测试。新建一个新的脚本，编写下面的代码：

```
public int action() throws Throwable {
    int i,t;
    t=0;
    lr.start_transaction("for");
    for(i=0;i<10000;i++)
    {
        t=t+i;
    }
    lr.end_transaction("for", lr.AUTO);
    lr.output_message(""+t);
    return 0;
} //end of action
```

这里模拟了前面.NET Vuser 中的测试案例，运行之后得到响应时间为：

```
Notify: Transaction "for" started.
Notify: Transaction "for" ended with "Pass" status (Duration: 0.0016).
```

有时候在 Java Vuser 中跑出的时间相对于前面.NET Vuser 的响应时间会长很多。这个



数据并不是非常正确，当加载到场景中运行时，数据会回到正常的情况。

### 8.11.1 使用 Java Vuser 测试 MySQL 数据库性能

这里我们通过 JDBC 编写一个连接 MySQL 数据库查询的 Java Vuser 脚本，用来测试数据库的性能，同样的写法可以应用在 SQL Server 或 Oracle 中。

这里要使用 MySQL JDBC 驱动程序 `mysql-connector-java-*.jar`，并加入到 ClassPath 中。打开 Run-time Settings，在 Classpath 中添加该 Jar 包，如图 8.39 所示。

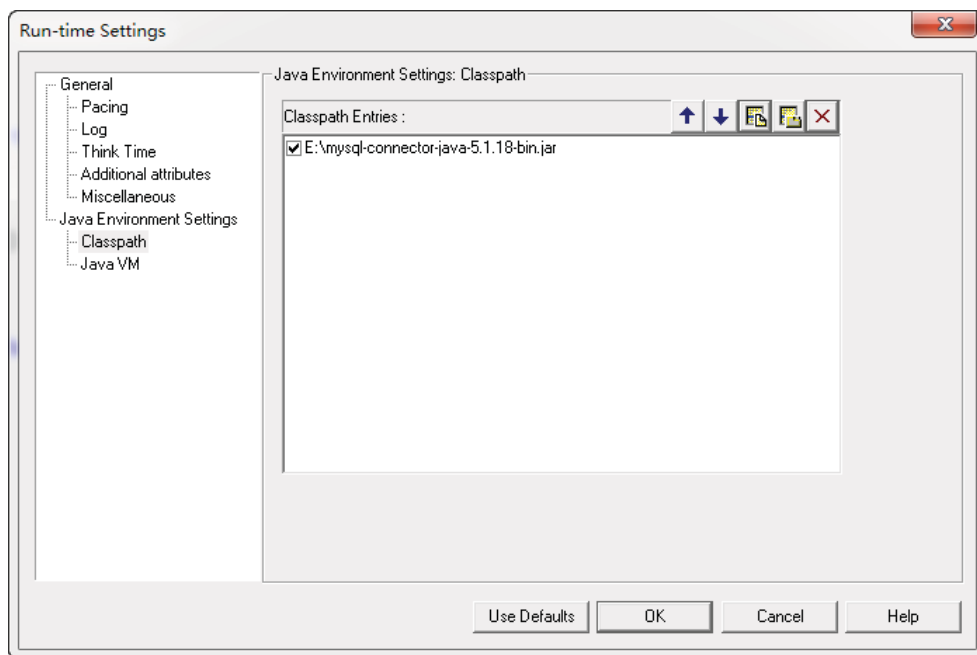


图 8.39 添加 Jar 包

接着我们编写 JDBC 连接代码（由于在 Vugen 中开发代码比较麻烦，推荐先在 Eclipse 中调试成功后，再复制到 Vugen 代码中），完整代码如下（为了排版略作格式调整）：

```
import lrap1.lr;
import java.sql.DriverManager;
import java.sql.*;
import com.mysql.jdbc.Connection;

public class Actions
{

    public int init() throws Throwable {
        return 0;
    } //end of init

    public int action() throws Throwable {
        int ColumnCount;
        int RowCount;
        String driver = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://localhost:3306/Phpwind85";
        String user = "root";
```

```

String password = "";
try {
    Class.forName(driver);
    lr.start_transaction("jdbc");
    Connection conn = (Connection) DriverManager.getConnection(url, user,
password);
    if(!conn.isClosed())
        System.out.println("数据库连接成功!");
    Statement stat=conn.createStatement();

    lr.start_transaction("search");

    ResultSet rs = stat.executeQuery("select * from pw_members");

    lr.end_transaction("search", lr.AUTO);

    ResultSetMetaData rsmd = rs.getMetaData();
    ColumnCount = rsmd.getColumnCount();
    rs.last();
    RowCount=rs.getRow();
    System.out.println("结果集的列数:" + ColumnCount);
    System.out.println("结果集的行数:" + RowCount);
    rs.close();
    conn.close();

    lr.end_transaction("jdbc", lr.AUTO);

}
catch(ClassNotFoundException e) {
    System.out.println("找不到驱动程序");
    e.printStackTrace();
}
catch(SQLException e) {
    e.printStackTrace();
}
return 0;
} //end of action

public int end() throws Throwable {
    return 0;
} //end of end
}

```

运行之后可以看到查询 **Phpwind85** 论坛中用户表所有记录的事务时间开销:

```

Notify: Transaction "jdbc" started.
System.out: 数据库连接成功!           Notify:
Notify: Transaction "search" started.
Notify: Transaction "search" ended with "Pass" status (Duration: 0.0637).
System.out: 结果集的列数:41           Notify:
System.out: 结果集的行数:2625         Notify:
Notify: Transaction "jdbc" ended with "Pass" status (Duration: 0.8890).

```

接着我们在 PhpMyAdmin 上执行相同的 SQL 语句核对一下查询的时间开销,如图 8.40 所示。

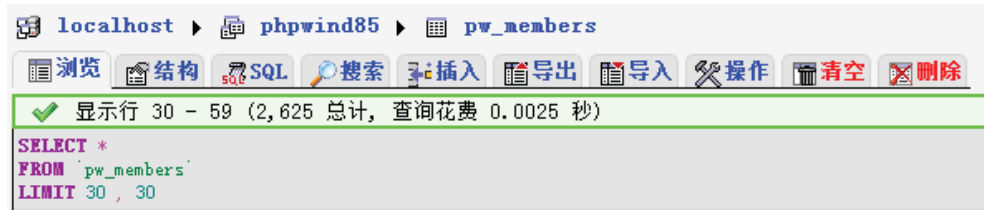


图 8.40 PhpMyAdmin 中的查询开销

可以看到 JDBC 执行查询的时候相对 PhpMyAdmin 得到的时间还是略长的,而整个 JDBC 的开销时间也可以得到。

通过 JDBC 还能实现读取 Excel 文件,代码如下:

```
public Connection getJdbcOdbcConnection(String filePath)
{
    String url = "jdbc:odbc:Driver={Microsoft Excel Driver
(*.xls)};DBQ="
        + filePath;
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection conn = DriverManager.getConnection(url);
        return conn;
    }
    catch (Exception e)
    {
        System.out.print(e.getMessage());
    }
    return null;
}
```

### 8.11.2 使用 Java Vuser 测试 JAR 包

最后我们来看一下如何使用 Java Vuser 调用已经开发完成的 JAR 包,测试对应 JAR 包中的方法性能。

这里在 Eclipse 中编写一个简单的功能,代码如下:

```
public double n(int x)
{
    double sum;
    sum=1;
    for(int i = 1; i<=x; i++){
        sum *= i;
    }
    return sum;
}
```

这是一个普通的阶层算法，我们把这个方法放在 `lrjar` 类中，编译通过后通过 Eclipse 的 `Export` 功能导出为 JAR 文件。接着新建一个 Java Vuser 脚本，通过运行设置 Classpath 加载这个 JAR 包，然后编写下面的代码完成调用：

```
import lrapi.lr;
import jar.lr.cloud.jarlr;

public class Actions
{
    public int init() throws Throwable {
        return 0;
    } //end of init

    public int action() throws Throwable {
        jarlr jarcall=new jarlr();
        double result;
        lr.start_transaction("jarcall");

        result=jarcall.n(10);

        lr.end_transaction("jarcall", lr.AUTO);
        System.out.println(result);

        return 0;
    } //end of action

    public int end() throws Throwable {
        return 0;
    } //end of end
}
```

通过运行这个代码可以看到当我们做 10 的阶层计算时，所需要开销的事务时间为：

```
Notify: Transaction "jarcall" started.
Notify: Transaction "jarcall" ended with "Pass" status (Duration: 0.0362).
System.out: 3628800.0
```

如果我们修改为计算 150 的阶层时，所需要开销的事务时间变为：

```
Notify: Transaction "jarcall" started.
Notify: Transaction "jarcall" ended with "Pass" status (Duration: 0.0440).
System.out: 5.7133839564458505E262
```

由于计算时间非常短，通过 LR 已经很难保证测试结果的精度了。当我们将脚本加载到场景中多用户并发时就可以得到 JAR 包中的方法执行响应时间，帮助我们定位代码级别的性能问题。

# B

## 附录B

### 几款性能测试工具入门速成

#### B.1 VSTS2010

VSTS2010 可以帮助我们简便地进行性能测试，在 VSTS2010 中新建一个 Web 性能测试，如图 B.1 所示。

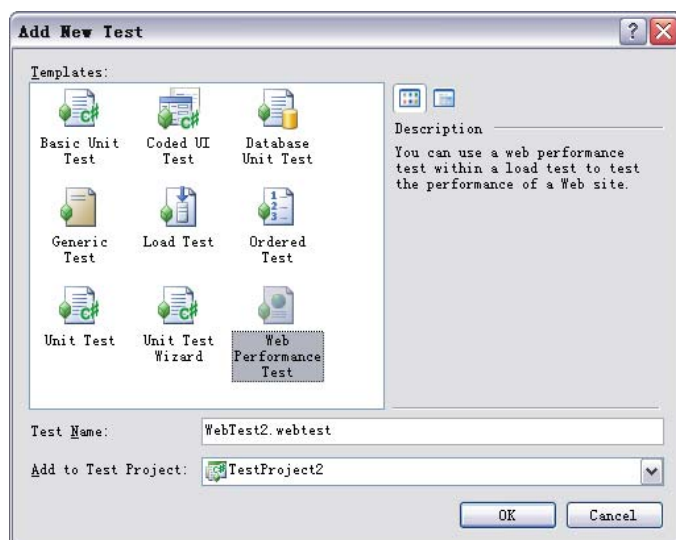


图 B.1 在 VSTS 2010 中新建 Web Performance Test

新建脚本后会启动一个 IE（IE 的加载项中确保有 Microsoft Web Test Recorder），在浏览器的左侧会出现录制工具条，我们只需要在 IE 中进行对应的操作，完成录制时单击“录制停止”即可生成脚本。

录制后可以通过直接在需要参数化的地方修改对应属性即可（支持的数据类型有 db、

xml、csv），如图 B.2 所示。

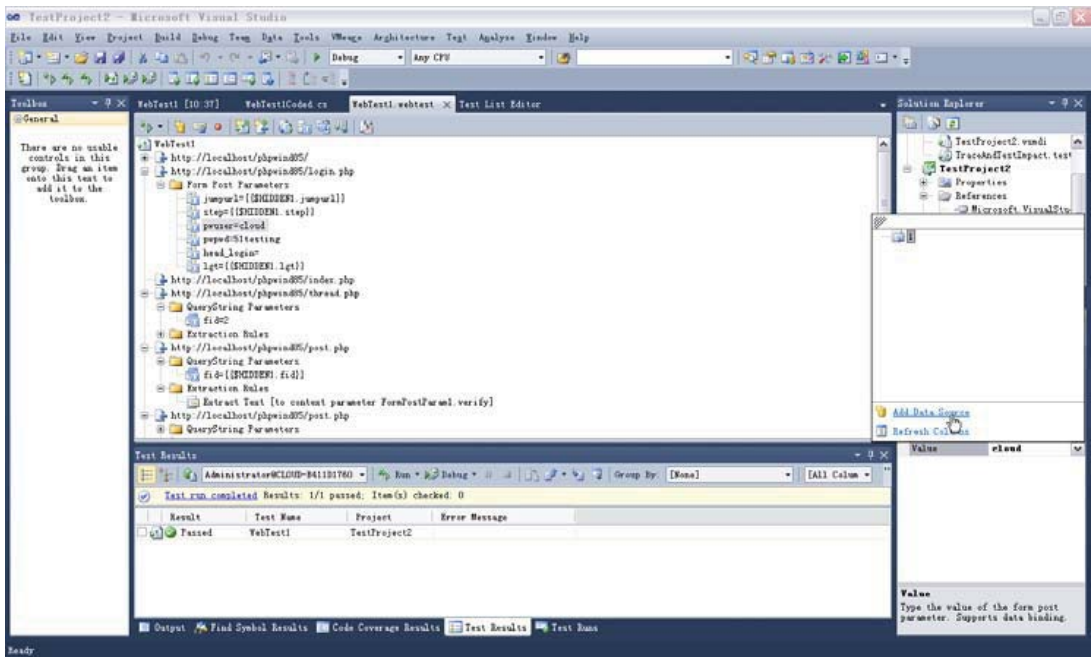


图 B.2 在 VSTS2010 中启动参数化

关联的做法是在 Login 请求上单击鼠标右键，在弹出的菜单中选择添加一个扩展规则（Add Extraction Rule），如图 B.3 所示。

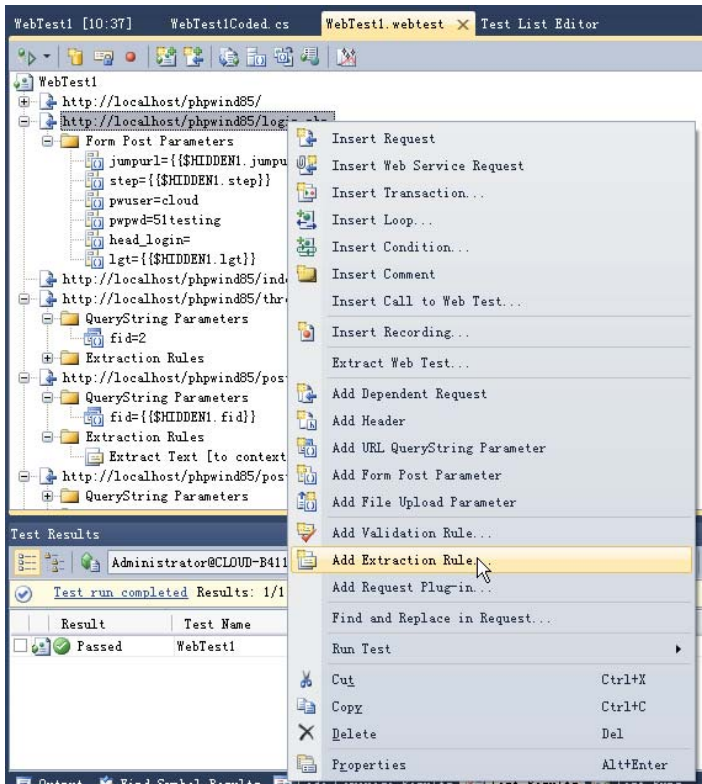


图 B.3 在 VSTS2010 中添加请求关联

弹出扩展规则，我们在规则中选择 Extract Text 规则，在这个规则中我们可以设置需要获得文本的左右边界，如图 B.4 所示。



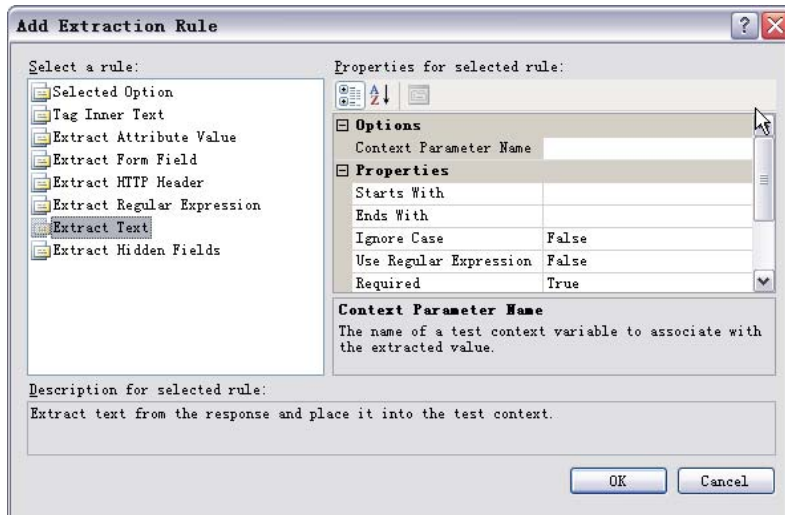


图 B.4 添加一个 Extract Text 规则

针对 Phpwind85, 这里按照关联 verify 的方式添加左边界为 lue=", 右边界为" nam, 如图 B.5 所示。

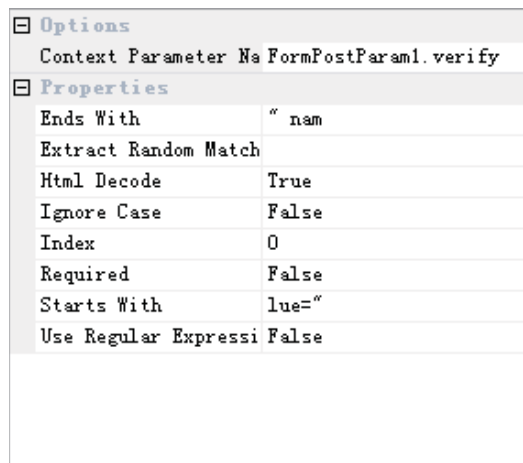


图 B.5 添加一个包含左右边界的 Extract Text 规则

关联返回的结果会被保存在 FormPostParam.verify 变量中, 然后在需要使用的时候用代码 `this.Context["FormPostParam1.verify"].ToString()` 来获得关键后的变量值即可, 在下面的提交帖子脚本中我们插入了动态 verify 变量值。

```
WebTestRequest request6 = new
WebTestRequest("http://localhost/phpwind85/post.php");
request6.Method = "POST";
request6.QueryStringParameters.Add("fid", "2", false, false);
request6.QueryStringParameters.Add("nowtime", "1318818570453",
false, false);
request6.QueryStringParameters.Add("verify", "f5015965", false,
false);

FormPostHttpBody request6Body = new FormPostHttpBody();
request6Body.FormPostParameters.Add("magicname", "");
request6Body.FormPostParameters.Add("magicid", "");
request6Body.FormPostParameters.Add("verify",
this.Context["FormPostParam1.verify"].ToString());
```

```

request6Body.FormPostParameters.Add("cyid", "0");
request6Body.FormPostParameters.Add("ajax", "1");
request6Body.FormPostParameters.Add("iscontinue", "0");
request6Body.FormPostParameters.Add("atc_title", "vsts");
request6Body.FormPostParameters.Add("atc_content",
"vststest");

request6Body.FormPostParameters.Add("atc_usesign", "1");
request6Body.FormPostParameters.Add("atc_autourl", "1");
request6Body.FormPostParameters.Add("atc_convert", "1");
request6Body.FormPostParameters.Add("digest", "0");
request6Body.FormPostParameters.Add("topped", "0");
request6Body.FormPostParameters.Add("replayorder", "0");
request6Body.FormPostParameters.Add("atc_money", "0");
request6Body.FormPostParameters.Add("atc_credittype",
"money");

request6Body.FormPostParameters.Add("atc_rvrc", "0");
request6Body.FormPostParameters.Add("atc_tags", "");
request6Body.FormPostParameters.Add("step", "2");
request6Body.FormPostParameters.Add("pid", "");
request6Body.FormPostParameters.Add("action", "new");
request6Body.FormPostParameters.Add("fid", "2");
request6Body.FormPostParameters.Add("tid", "0");
request6Body.FormPostParameters.Add("article", "0");
request6Body.FormPostParameters.Add("special", "0");
request6Body.FormPostParameters.Add("_hexie", "99fc0eb4");
request6.Body = request6Body;
yield return request6;
request6 = null;

```

接着我们为脚本添加事务获得发帖的响应时间，在请求上单击鼠标右键，在弹出的菜单中选择 Insert Transaction，如图 B.6 所示。

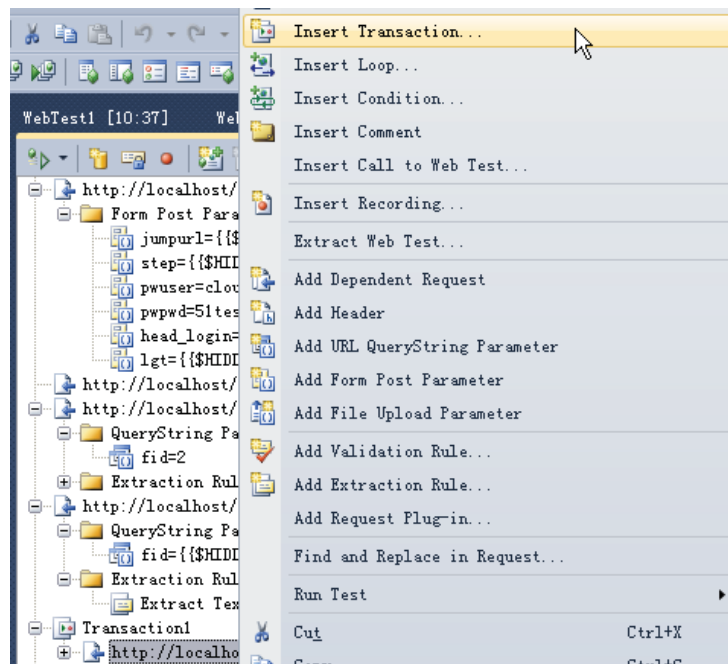


图 B.6 添加事务

VSTS 2010 中事务是独立的概念，自动将请求放在事务下。将请求生成代码后可以看到对应的代码为：

```
this.BeginTransaction("Transaction1");
this.EndTransaction("Transaction1");
```

当对该代码运行时可以看到对应的响应时间数据，如图 B.7 所示。

Request	Status	Total Time	Request Time	Request B...	Response ...
http://localhost/phpwind85/thread.php	200 OK	0.145 sec	0.088 sec	0	186,964
http://localhost/phpwind85/post.php	200 OK	0.109 sec	0.089 sec	0	247,880
Transaction1		0.053 sec	-	297	94
http://localhost/phpwind85/post.php	200 OK	0.053 sec	0.053 sec	297	94
http://localhost/phpwind85/read.php	200 OK	0.150 sec	0.083 sec	0	339,346

图 B.7 事务时间

脚本开发完成，最后新建一个 Load Test 来完成负载和监控操作，如图 B.8 所示。

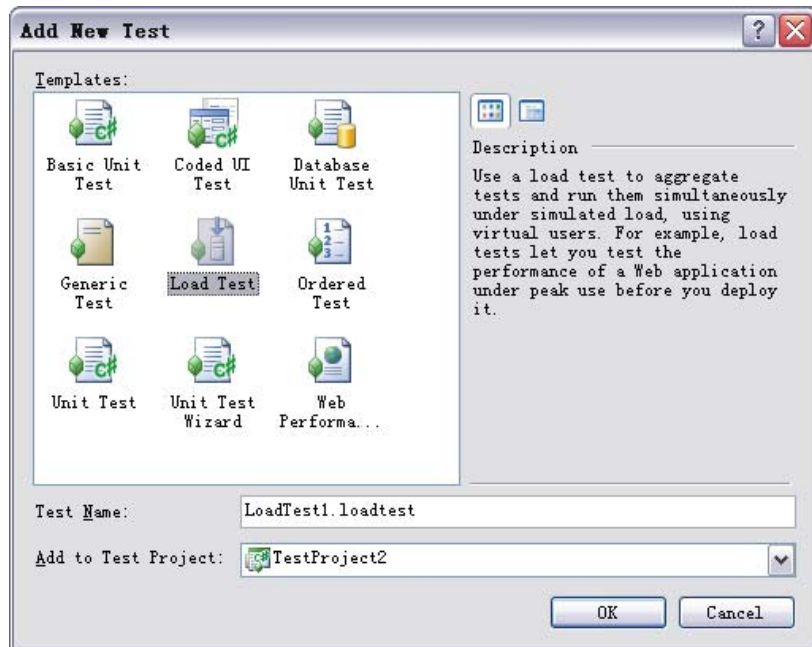


图 B.8 新建负载测试

## B.2 Apache AB

AB 的全称是 ApacheBench，是 Apache 附带的一个小工具，专门用于 HTTP Server 的 benchmark testing，可以同时模拟多个并发请求。

我们先编写一个这样的命令来对 www.google.com 进行一个简单的性能测试。

```
ab -n 10 -c 10 http://www.google.com/
//启动 AB，向 www.google.com 发送 10 个请求(-n 10)，并每次发送 10 个请求(-c 10)
```

运行后我们可以看到下面的 AB 输出测试报告。

```
www/web/Apache/bin/ab -n 10 -c 10 http://www.google.com/
This is ApacheBench, Version 2.0.40-dev <$Revision: 1.146 $> apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright 1997-2005 The Apache Software Foundation, http://www.apache.org/
```

```

Benchmarking www.google.com (be patient).....done
Server Software:      GWS/2.1
Server Hostname:      www.google.com
Server Port:          80
Document Path:        /
Document Length:      230 bytes
Concurrency Level:     10

/*整个测试持续的时间*/
Time taken for tests:   3.234651 seconds

/*完成的请求数量*/
Complete requests:     10

/*失败的请求数量*/
Failed requests:       0
Write errors:          0
Non-2xx responses:     10
Keep-Alive requests:   10

/*整个场景中的网络传输量*/
Total transferred:     6020 bytes

/*整个场景中的 HTML 内容传输量*/
HTML transferred:      2300 bytes

/*相当于 LR 中的每秒事务数，后面括号中的 mean 表示这是一个平均值*/
Requests per second:   3.09 [#/sec] (mean)

/*每个请求的响应时间，相当于 LR 中的平均事务响应时间，后面括号中的 mean 表示这是一个平均值*/
Time per request:      3234.651 [ms] (mean)

/*每个请求的响应时间，基于并行计算*/
Time per request:      323.465 [ms] (mean, across all concurrent requests)

/*平均每秒网络上的流量，可以帮助排除是否存在网络流量过大导致响应时间延长的问题*/
Transfer rate:         1.55 [Kbytes/sec] received

/*网络上消耗的时间的分解*/
Connection Times (ms)
      min mean[+/-sd] median max
Connect:    20  318 926.1    30 2954
Processing:  40 2160 1462.0  3034 3154
Waiting:    40 2160 1462.0  3034 3154
Total:      60 2479 1276.4  3064 3184

/*下面的内容为整个场景中所有请求的响应情况。在场景中每个请求都有一个响应时间，其中 50%
的用户响应时间小于 3064 毫秒，60 % 的用户响应时间小于 3094 毫秒，最大的响应时间小于 3184
毫秒*/
Percentage of the requests served within a certain time (ms)
50%    3064

```

```

66%    3094
75%    3124
80%    3154
90%    3184
95%    3184
98%    3184
99%    3184
100%   3184 (longest request)

```

AB 不像 LR 那么强大，但是它足够轻便，如果只是在开发过程中想检查一下某个模块的响应情况，或者做一些场景比较简单的测试，AB 还是一个不错的选择——至少不用花费很多时间去学习 LR 那些复杂的功能。

下面是 AB 的详细参数解释：

```

ab [ -A auth-username:password ] [ -c concurrency ] [ -C cookie-name=value ]
[ -d ] [ -e csv-file ] [ -g gnuplot-file ] [ -h ] [ -H custom-header ] [ -i ]
[ -k ] [ -n requests ] [ -p POST-file ] [ -P proxy-auth-username:password ] [ -q ]
[ -s ] [ -S ] [ -t timelimit ] [ -T content-type ] [ -v verbosity ] [ -V ] [ -w ]
[ -x <table>-attributes ] [ -X proxy[:port] ] [ -y <tr>-attributes ] [ -z
<td>-attributes ] [http://]hostname[:port]/path

```

**-A auth-username:password**  
Supply BASIC Authentication credentials to the server. The username and password are separated by a single : and sent on the wire base64 encoded. The string is sent regardless of whether the server needs it (i.e., has sent an 401 authentication needed).

**-c concurrency**  
Number of multiple requests to perform at a time. Default is one request at a time.

**-C cookie-name=value**  
Add a Cookie: line to the request. The argument is typically in the form of a name=value pair. This field is repeatable.

**-d**  
Do not display the "percentage served within XX [ms] table". (legacy support).

**-e csv-file**  
Write a Comma separated value (CSV) file which contains for each percentage (from 1% to 100%) the time (in milliseconds) it took to serve that percentage of the requests. This is usually more useful than the 'gnuplot' file; as the results are already 'binned'.

**-g gnuplot-file**  
Write all measured values out as a 'gnuplot' or TSV (Tab separate values) file. This file can easily be imported into packages like Gnuplot, IDL, Mathematica, Igor or even Excel. The labels are on the first line of the file.

**-h**  
Display usage information.

**-H custom-header**  
Append extra headers to the request. The argument is typically in the form of a valid header line, containing a colon-separated field-value pair (i.e., "Accept-Encoding: zip/zop;8bit").

**-i**  
Do HEAD requests instead of GET.

`-k`  
Enable the HTTP KeepAlive feature, i.e., perform multiple requests within one HTTP session. Default is no KeepAlive.

`-n requests`  
Number of requests to perform for the benchmarking session. The default is to just perform a single request which usually leads to non-representative benchmarking results.

`-p POST-file`  
File containing data to POST.

`-P proxy-auth-username:password`  
Supply BASIC Authentication credentials to a proxy en-route. The username and password are separated by a single `:` and sent on the wire base64 encoded. The string is sent regardless of whether the proxy needs it (i.e., has sent an 407 proxy authentication needed).

`-q`  
When processing more than 150 requests, ab outputs a progress count on stderr every 10% or 100 requests or so. The `-q` flag will suppress these messages.

`-s`  
When compiled in (ab `-h` will show you) use the SSL protected https rather than the http protocol. This feature is experimental and very rudimentary. You probably do not want to use it.

`-S`  
Do not display the median and standard deviation values, nor display the warning/error messages when the average and median are more than one or two times the standard deviation apart. And default to the min/avg/max values. (legacy support).

`-t timelimit`  
Maximum number of seconds to spend for benchmarking. This implies a `-n 50000` internally. Use this to benchmark the server within a fixed total amount of time. Per default there is no timelimit.

`-T content-type`  
Content-type header to use for POST data.

`-v verbosity`  
Set verbosity level - 4 and above prints information on headers, 3 and above prints response codes (404, 200, etc.), 2 and above prints warnings and info.

`-V`  
Display version number and exit.

`-w`  
Print out results in HTML tables. Default table is two columns wide, with a white background.

`-x <table>-attributes`  
String to use as attributes for `<table>`. Attributes are inserted `<table here >`.

`-X proxy[:port]`  
Use a proxy server for the requests.

`-y <tr>-attributes`  
String to use as attributes for `<tr>`.

`-z <td>-attributes`  
String to use as attributes for `<td>`.



## B.3 WebBench

WebBench 是有名的网站压力测试工具，它是由 Lionbridge 公司开发的。

WebBench 能测试处在相同硬件上、不同服务的性能，以及不同硬件上同一个服务的运行状况。WebBench 的标准测试可以向我们展示服务器的两项内容：每秒钟相应请求数和每秒钟传输数据量。WebBench 不但能具有静态页面的测试能力，还有对动态页面（ASP、PHP、JAVA、CGI）进行测试的能力。还有就是它支持对含有 SSL 的安全网站（例如，电子商务网站）进行静态或动态的性能测试。

在官方网站下载 `webbench-1.5.tar.gz` 源代码包，然后将其解压编译。

```
tar zxvf webbench-1.5.tar.gz
cd webbench-1.5
make && make install
```

使用的时候通过参数 `c`（表示并发数）和 `t`（表示时间单位为秒）完成对某一页面的请求模拟：

```
webbench -c 500 -t 30 http://192.168.11.32/phpwind85/
```

运行后得到日志：

```
Webbench - Simple Web Benchmark 1.5
Copyright (c) Radim Kolar 1997-2004, GPL Open Source Software.
Benchmarking: GET http://192.168.11.32/phpwind85/
500 clients, running 30 sec.
Speed=2765 pages/min, 14314212 bytes/sec.
Requests: 2413 succeed, 0 failed.
```

## B.4 HTTP\_Load

HTTP\_Load 以并行复用的方式运行，用以测试 Web 服务器的吞吐量与负载。但是它不同于大多数压力测试工具，它可以以一个单一的进程运行，一般不会把客户机搞死，还可以测试 HTTPS 类的网站请求。

下载地址：[http://soft.vpser.net/test/http\\_load/http\\_load-12mar2006.tar.gz](http://soft.vpser.net/test/http_load/http_load-12mar2006.tar.gz)，程序非常小，解压后也不到 100KB。

同样解压后编译：

```
tar zxvf http_load-12mar2006.tar.gz
cd http_load-12mar2006
make && make install
```

命令格式：

```
http_load -p 并发访问进程数 -s 访问时间 需要访问的 URL 文件
```

参数其实可以自由组合，参数之间的选择并没有什么限制。比如你可以写成：

```
http_load -parallel 5 -seconds
```

300 urls.txt 也是可以的。

我们把参数给大家简单说明一下。

- `-parallel`（简写 `-p`）：含义是并发的用户进程数。
- `-fetches`（简写 `-f`）：含义是总计的访问次数。

- **-rate** (简写-p)：含义是每秒的访问频率。
- **-seconds** (简写-s)：含义是总计的访问时间。

准备 URL 文件: `urllist.txt`, 文件格式是每行一个 URL, URL 最好超过 50~100 个测试效果比较好, 文件格式如下:

```
http://www.vpser.net/uncategorized/choose-vps.html
http://www.vpser.net/vps-cp/hypervm-tutorial.html
http://www.vpser.net/coupons/diavps-april-coupons.html
http://www.vpser.net/security/vps-backup-web-mysql.html
```

例如:

```
http_load -p 30 -s 60 urllist.txt
```

参数了解了, 我们运行一条命令来看看它的返回结果

命令: `% ./http_load -rate 5 -seconds 10 urls` 说明执行了一个持续时间 10 秒的测试, 每秒的频率为 5。

```
49 fetches, 2 max parallel, 289884 bytes, in 10.0148 seconds5916 mean
bytes/connection4.89274
fetches/sec, 28945.5 bytes/secmsecs/connect: 28.8932 mean, 44.243 max,
24.488 minmsecs/first
-response: 63.5362 mean, 81.624 max, 57.803 minHTTP response codes: code 200
- 49
```

结果分析:

- 49 fetches, 2 max parallel, 289884 bytes, in 10.0148 seconds

说明在上面的测试中运行了 49 个请求, 最大的并发进程数是 2, 总计传输的数据是 289884bytes, 运行的时间是 10.0148 秒。

- 5916 mean bytes/connection 说明每一连接平均传输的数据量  $289884/49=5916$

- 4.89274 fetches/sec, 28945.5 bytes/sec。

说明每秒的响应请求为 4.89274, 每秒传递的数据为 28945.5 bytes。

- msecs/connect: 28.8932 mean, 44.243 max, 24.488 min

说明每次连接的平均响应时间是 28.8932 ms。

最大的响应时间为 44.243 为 ms, 最小的响应时间 24.488 为 ms。

- msecs/first-response: 63.5362 mean, 81.624 max, 57.803 min

- HTTP response codes: code 200 — 49

说明打开响应页面的类型, 如果 403 的类型过多, 那可能要注意是否系统遇到了瓶颈。

特殊说明:

测试结果中主要的指标是 `fetches/sec`、`msecs/connect`, 即服务器每秒能够响应的查询次数, 用这个指标来衡量性能。似乎比 Apache 的 `Ab` 准确率要高一些, 也更有说服力一些。`Qpt` 表示每秒响应用户数和 `response time`, 每二次连接响应用户时间。

测试的结果主要也是看这两个值。当然仅有这两个指标并不能完成对性能的分析, 我们还需要对服务器的 CPU、mem 进行分析, 才能得出结论。

## B.5 Siege

一款开源的压力测试工具，可以根据配置对一个 Web 站点进行多用户的并发访问，记录每个用户所有请求过程的响应时间，并在一定数量的并发访问下重复进行。

在官方网站 <http://www.joedog.org/> 下载安装包 `siege-2.67.tar.gz`

解压后配置编译：

```
tar -zxf siege-2.67.tar.gz
cd siege-2.67/
./configure
make
make install
```

然后运行命令：

```
siege -c 200 -r 10 -f example.url
```

`-c` 是并发量，`-r` 是重复次数。`url` 文件就是一个文本，每行都是一个 url，它会从里面随机访问的。

`example.url` 内容：

```
http://www.licess.cn
http://www.vpser.net
http://soft.vpser.net
```

结果说明：

```
Lifting the server siege... done.
Transactions: 3419263 hits          //完成 419263 次处理
Availability: 100.00 %             //100.00 % 成功率
Elapsed time: 5999.69 secs         //总共用时
Data transferred: 84273.91 MB      //总共数据传输 84273.91 MB
Response time: 0.37 secs           //响应用时 1.65 秒：显示网络连接的速度
Transaction rate: 569.91 trans/sec //平均每秒完成 569.91 次处理：表示服务器处理后
Throughput: 14.05 MB/sec           //平均每秒传送数据
Concurrency: 213.42               //实际最高并发数
Successful transactions: 2564081  //成功处理次数
Failed transactions: 11           //失败处理次数
Longest transaction: 29.04        //每次传输所花的最长时间
Shortest transaction: 0.00        //每次传输所花的最短时间
```

## B.6 JMeter

Apache JMeter 是 Apache 组织开发的基于 Java 的压力测试工具。用于对软件做压力测试，它最初用于 Web 应用测试，但后来扩展到其他测试领域。它可以用于测试静态和动态资源（例如静态文件、Java 小服务程序、CGI 脚本、Java 对象、数据库，FTP 服务器等）。JMeter 可以用于对服务器、网络或对象模拟巨大的负载，来在不同压力类别下测试它们的强度和分析整体性能。

在官方网站 [http://jmeter.apache.org/download\\_jmeter.cgi](http://jmeter.apache.org/download_jmeter.cgi) 可以下载最新版本。

### 1. JMeter 的脚本

JMeter 的脚本最基本来源有以下三种方式：

- 使用 Badboy 录制。
- 使用 JMeter 自带的 HTTP 代理服务器组件进行录制。
- 手动编写。

## 2. 使用 Badboy 进行录制

Badboy 的界面如图 B.9 所示，录制操作类似于 LoadRunner。



图 B.9 Badboy 界面

这里 step 的功能就相当于 LoadRunner 里的事务的概念，大家在录制的时候，每一步都可以添加一个新的 step，从而方便导入到 JMeter 之后进行相应的脚本编辑工作。

## 3. 导入到 JMeter

具体操作如图 B.10 所示。

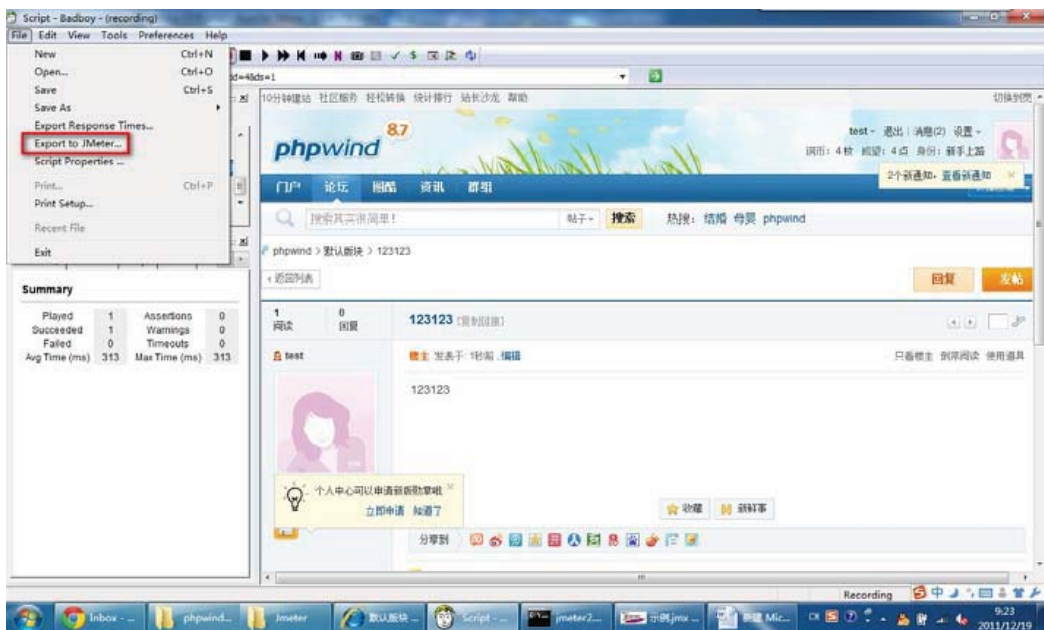


图 B.10 导入到 JMeter

#### 4. 使用 JMeter 导入 Badboy 录制的脚本

具体操作如图 B.11 所示。



图 B.11 使用 JMeter 导入 Badboy 录制的脚本

Badboy 录制的时候已经帮我们建立好了统一的 Cookie 管理器 and 头文件管理器，必要的时候，可以自己在每个 HTTP 请求下建立单独的 Cookie 管理器和头文件管理器。

#### 5. 编辑脚本

录制的脚本还是需要编辑才能进行有效工作的，需要添加必要的控制器，思考时间和验证。

##### 1) 控制器

如果不是为了测试登录操作，一般而言登录操作是希望只运行一次的，因此我们需要添加“仅一次控制器”，如图 B.12 所示。



图 B.12 选择“仅一次控制器”



如图 B.12 所示, JMeter 提供了很多的逻辑控制器给大家使用, 大家可以根据需要选择。添加控制器之后, 需要把那些只运行一次的步骤都复制到该控制器之下。JMeter 允许我们进行复制、粘贴及拖曳操作。

添加“仅一次控制器”, 处理后如图 B.13 所示。

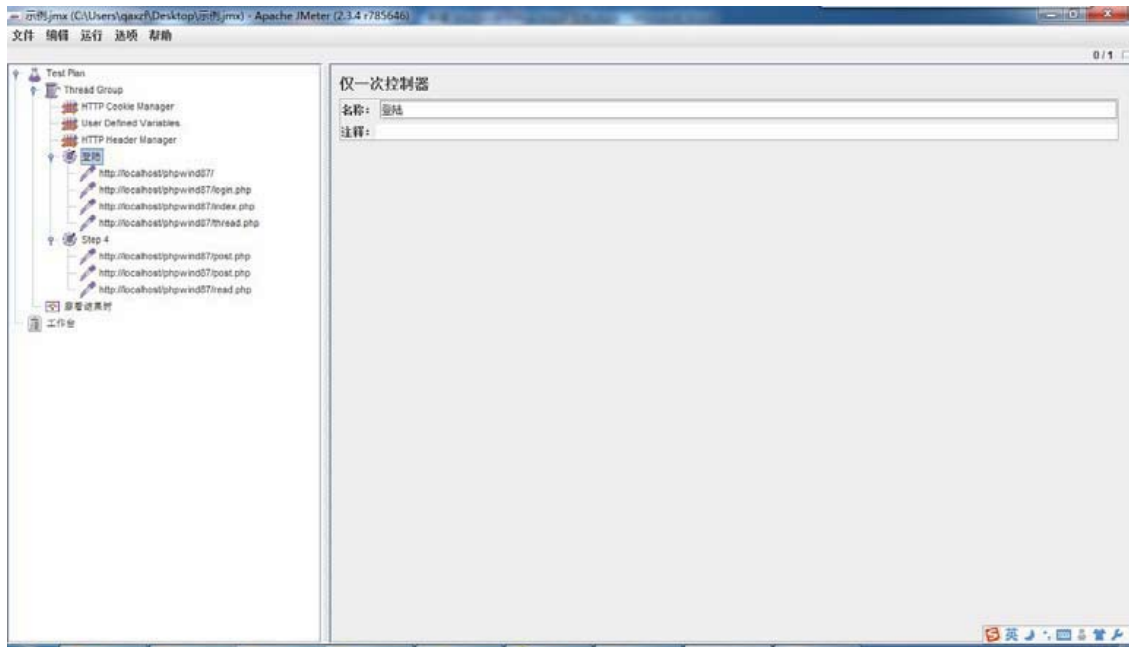


图 B.13 添加完成“仅一次控制器”

## 2) 添加定时器

定时器类似于 LoadRunner 中的 ThinkTime 概念, 但又有所区别。因为 JMeter 中的定时器有一个作用域的概念, 如图 B.14 所示。

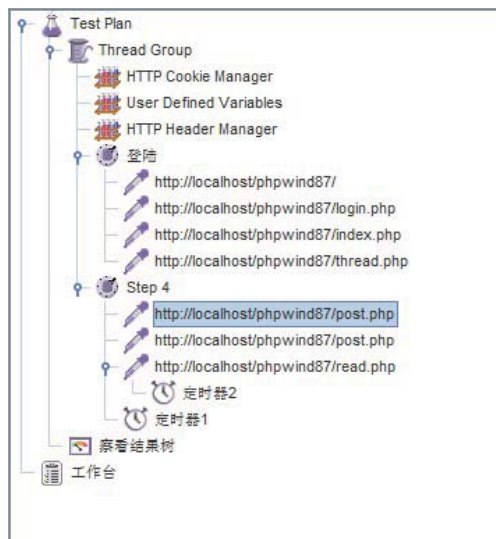


图 B.14 定时器

在 Step4 中包含 3 个请求, 定时器 1 是作用于这三个请求的, 定时器 2 只作用最后一个请求。也就是说运行时, Step4 中的 3 个请求运行前都会先执行一次定时器 1, 而定时器 2 在执行最后一个 read 操作的时候才会执行。



### 3) 添加验证

首先用鼠标右键单击测试计划，添加监控组件“查看结果树”，该组件是给我们调试脚本用的，在正式运行压力测试时请记得删除该组件。

通过结果树组件，我们可以看到执行每项操作时，服务器的返回结果，从而确定检查点需要设置的值。

JMeter 这里提供的检查机制是断言，一般采用响应断言，如图 B.15 所示。

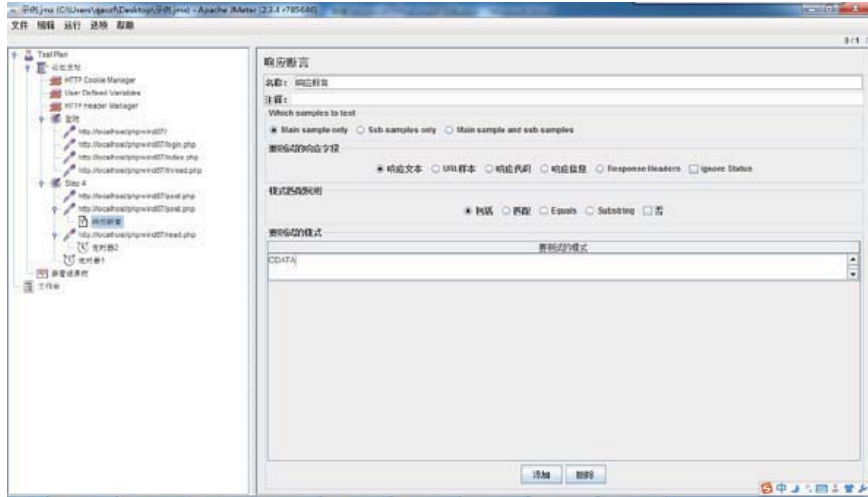


图 B.15 响应断言

单击“添加”按钮，将我们需要验证的值填入文本框即可。

示例中的发帖操作，成功时会返回一个字符串，包含 CDATA 的字段，因此这里的断言就设置要测试的模式为 CDATA。

### 4) 关联

JMeter 提供了很多后置的处理器，可以做关联用，一般关联采用最多的就是正则表达式提取器。

如图 B.16 所示，在提交帖子之后系统会返回一个贴子的 ID，并在之后的 read 步骤中使用这个 ID。

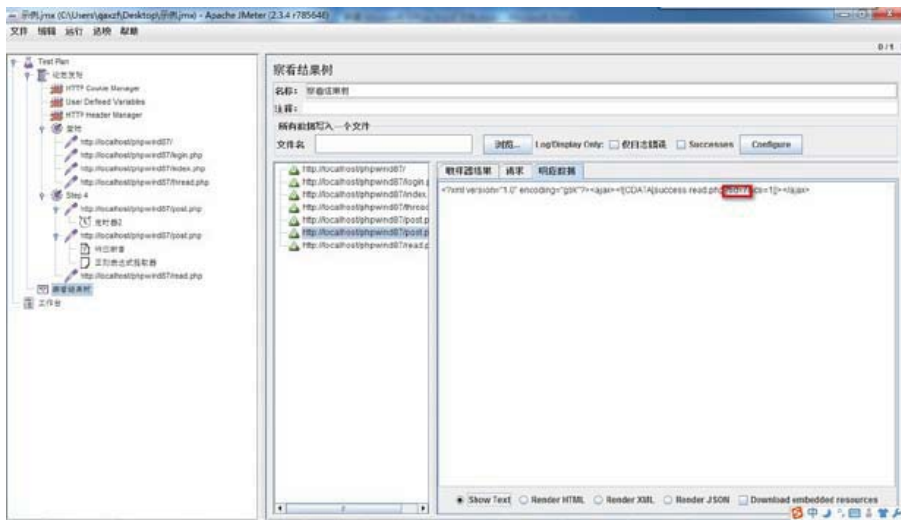


图 B.16 返回帖子的 ID

提交帖子的请求，单击鼠标右键，在弹出的菜单中选择“添加后置处理器”→“正则表达式提取器”命令如图 B.17 所示。

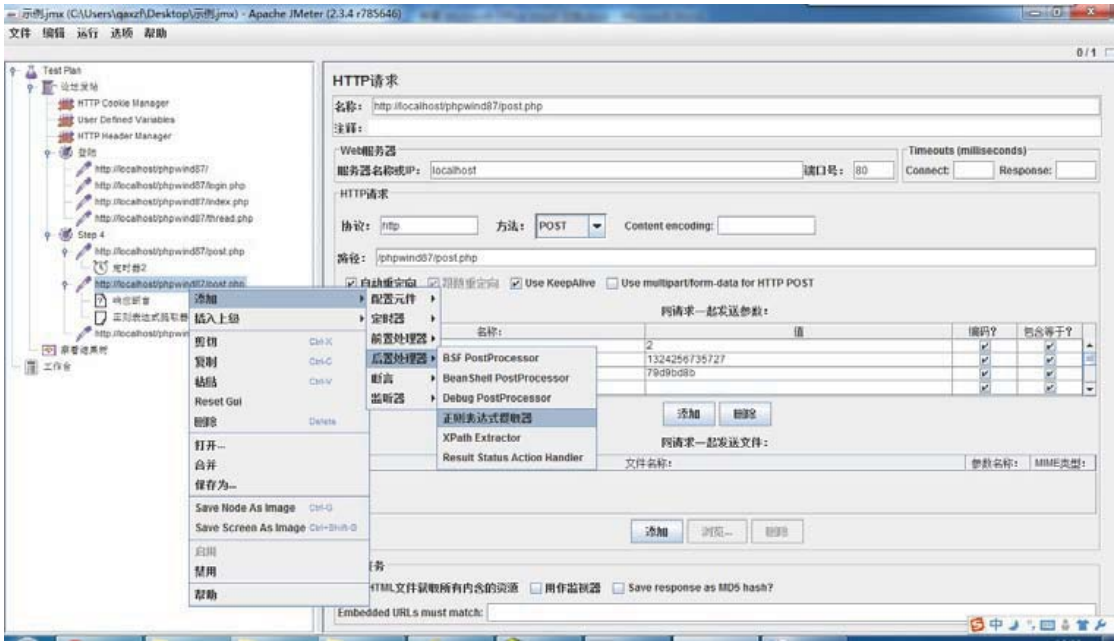


图 B.17 提交帖子的请求

我们需要关联的字段为：“tid=7&ds=1”中的这个 7。因此正则表达式为：tid=(\*)&ds=1，用“(\*)”替换我们需要关联的字段，如图 B.18 所示。

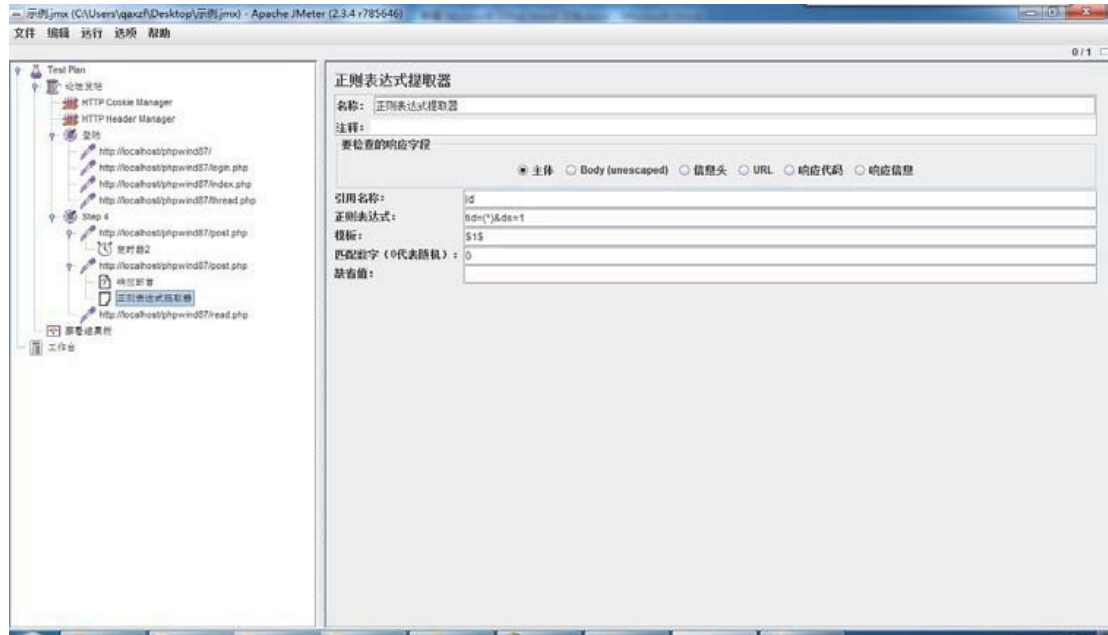


图 B.18 替换需要关联的字段

模板的编号对应正则表达式的括号对，顺序排列的。  
设置好之后，在 read 请求中，使用该关联值，调用方式为：\${id}，大括号内用之前定义的引用名称，如图 B.19 所示。

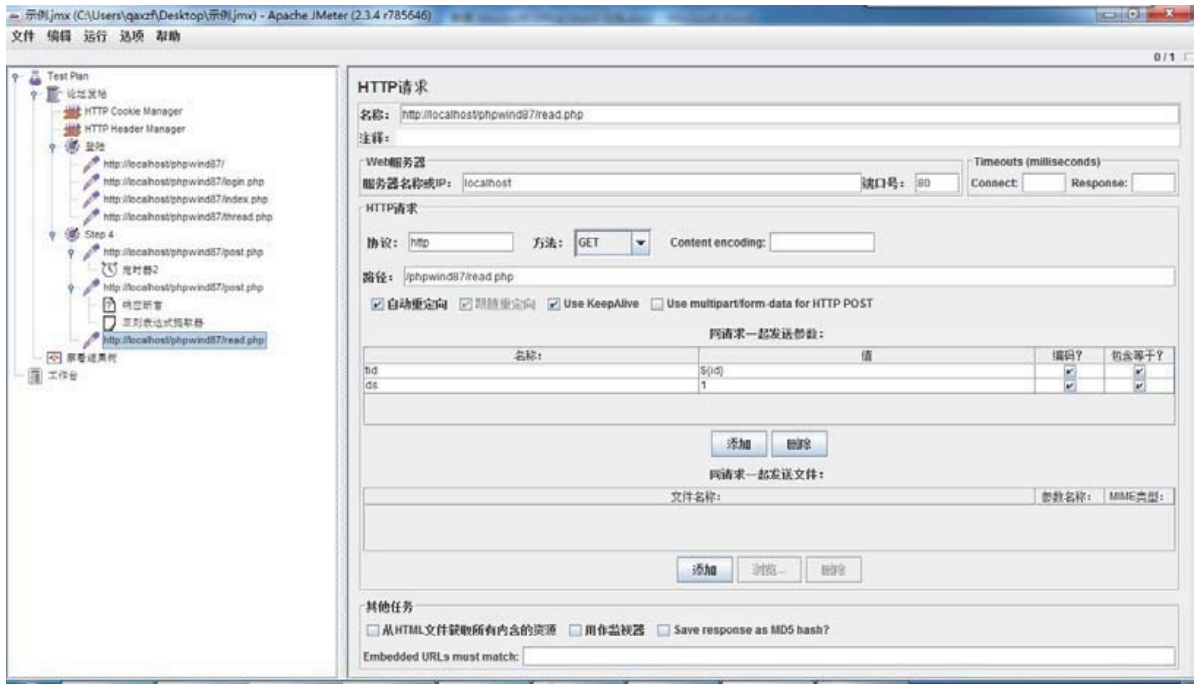


图 B.19 使用关联值

### 5) 生成帖子的标题和内容

打开“选项”下的“函数助手”对话框，使用函数助手生成一个随机函数，作为帖子的标题和内容，如图 B.20 所示。

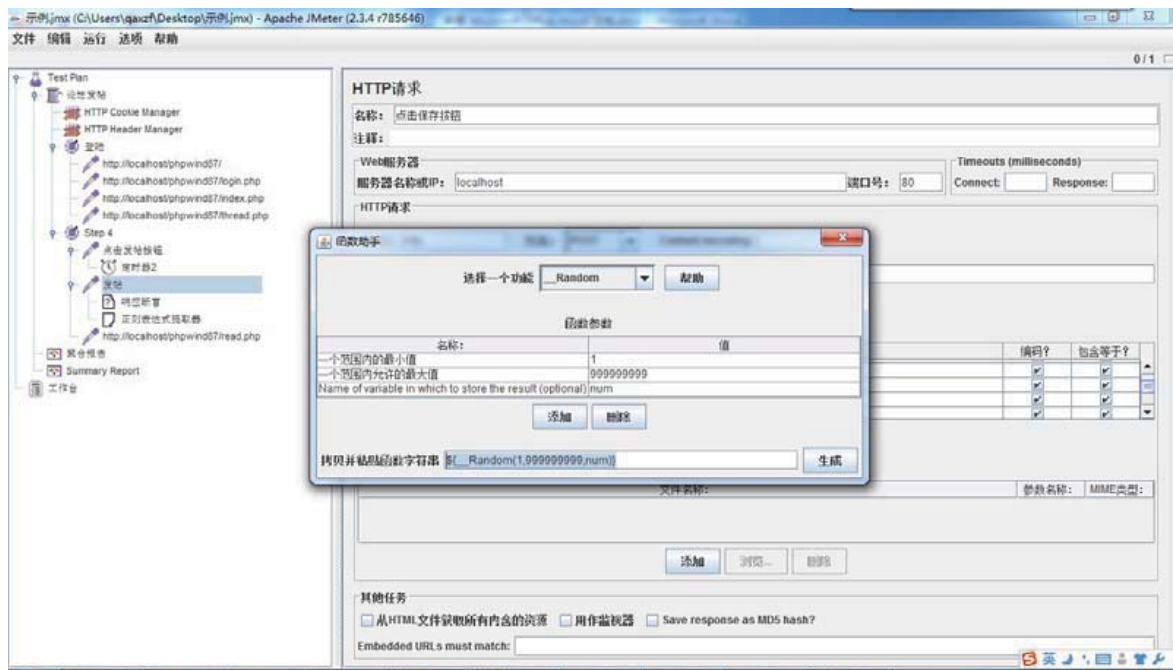


图 B.20 函数助手

设置好最大、最小值和名称后单击“生成”按钮，就会生成函数调用字符串，复制该字符串到需要使用的地方即可。

### 6) 添加集合点

JMeter 的集合点也在定时器中设置，如图 B.21 所示。

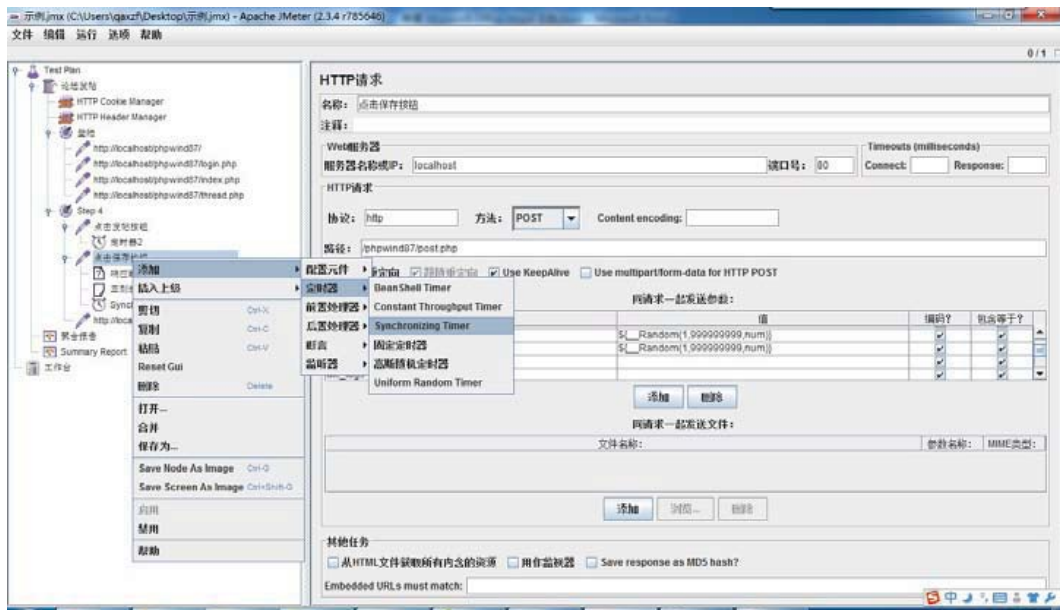


图 B.21 添加集合点

### 7) 设置用户数，添加监视器

通过单击线程组，可以设置并发的线程数、并发策略等，通过添加监视器，可以选用相应的监视器，对测试结果进行统计分析。

JMeter 对系统资源的监控集成几乎没有，大家需要通过插件或者别的辅助工具进行监控。



#### 注意

大家可以到下面的地址去下载 JMeter 的相关插件，或者去 Google 搜索，JMeter 作为开源软件，有很多很好用的开源插件，也可以自己编写相关的插件。

<http://code.google.com/p/jmeter-plugins/>



## · 业界热评 ·

本书是作者关于LoadRunner性能测试的第二本著作，但绝非前作的简单升级，而是一次从结构到内容的完整重构。本书在延续了上一版主线展开的基础上，以指导初级读者进行企业级实战为目标，以作者多年项目实践及培训教学经验为基础，彻底替换了测试案例及脚本代码，并新增介绍了许多前沿的技术和工具。本书知识系统、理论适中，由基础到高级，内容全，涵盖广，对性能定位和性能瓶颈的分析尤为深入透彻，具有较好的实用性和前瞻性，不仅可作为性能测试初入门者进阶之梯，亦可成为性能测试资深人士玩味推敲的案头书。简言之，本书脉络清晰酣畅，行文诙谐生动，剖析深入浅出，总结精辟独到，案例贴近企业实战，技术力求紧跟前沿，是性能测试领域不可多得的一本好书。

傅江如

北京西祠互动信息技术有限公司 测试部经理

整本书，作者结合自身实际的项目经验，运用实例对LoadRunner工具的使用和新特性做了深入、透彻的讲解，同时对性能测试做了深入的剖析，有效地指导了性能测试人员从方案到脚本，从脚本到场景以及后期对测试结果分析、定位、调优的工作开展，是从事性能测试工作的人员不可多得的一本好书。

姚宗余

腾讯科技（上海）有限公司专项技术（性能）测试工程师

在信息技术高速发展的时代，企业对员工的需求不仅仅是当前会什么，而是要求员工能够根据企业的实际需要快速学习、掌握、应用一门新技术、新方法，完成企业快速产品交付中的各项任务。该书从学习者快速学习角度出发，既可以按章节顺序学习，掌握性能测试理论、方法、工具、实践；也可以作为案头工具书，在项目进行中作为手册、资料随时翻看，即学即用。

王海龙

大智慧股份有限公司 测试经理

在互联网的大潮中，更多的公司对性能做了更严格的要求。当性能差时，会造成诸多问题，例如网页打开速度3秒定律延迟，支付系统数据提交延时及失败等，对公司利益和个人体验带来极为不好的影响，所以更多的技术专家为了能更好地去优化性能做出了很多努力！本书深入浅出地介绍了最新版的LoadRunner 11的各方面内容，书中包括大量的应用实例，最吸引我的是LNMP架构的测试实例，因为更多的互联网公司选择了这种架构来进行实际的开发，在我的工作中可以得到充分的应用，是一本不可多得的专业性能测试资料。

高振华

齐家网无线事业部测试经理（前盛大网络测试主管）

本书针对LoadRunner性能测试工具由浅入深地讲解，同时借助LoadRunner工具给我们讲解了对性能测试的了解思路、性能测试该如何开展，为广大抱有“性能”梦想的初级测试人员提供了一个完美切入点，同时也为奋战在“性能”之路的工程师们疏通了脉络，是使其更好地理解消化“性能”的一粒开胃药。本书内容丰富，不但从实施角度对性能测试流程进行了——详解，而且还附带了常用的测试文档、模板以及大量案例，可以做性能测试字典使用。章节思路清晰，描述有理有据，还不失幽默风趣，也是我们测试工作者学习的典范。

一本好书，值得一读。

李锋

中国互联网络信息中心（CNNIC）高级性能测试工程师

如果想进一步在性能测试横向或纵向发展的话，可以看这本书，此书内容涵盖了整个性能领域的大部分知识领域，更是在前作的基础上，加深了细节的技术知识点，可以帮助读者从“懂”到“精”的过渡。

马利伟

分众传媒测试主管（前阿里巴巴金融资深测试工程师）

上架建议：软件测试

ISBN 978-7-121-16739-3



定价：79.00元



策划编辑：李 冰  
责任编辑：葛 娜  
封面设计：李 玲