软件测试从零开始

1. 软件性能测试

当今，计算机和软件工程发展越来越快，新的概念名词和技术手段层出不穷，可谓日新月异。在软件性能测试范畴内就有很多，诸如并发测试、压力测试、基准测试、测试场景等概念和名词，这让刚接触性能测试的新手眼花缭乱，目不暇接。但我们如果能深入软件性能测试的本质，从哲学的角度看问题，找出其内在联系，比如因果关系、形式内容关系，甚至重叠关系等，理清思路之后，那么做软件性能测试就会如庖丁解牛，游刃有余。

1.1. 什么是软件的性能

1.1.1. 软件

计算机软件作为人类逻辑智慧的结晶，它可以模拟并替代人类的一些活动，替人“发号施令”。在计算机软件发展的短短几十年内，计算机软件以非常快的速度渗透到了人类社会的各个角落，比如现在我们在家上网，出门坐公交车刷卡，在工作中发电子邮件等，这些生活的背后都有大量的软件系统运行支持。

同时，有关软件的概念和名词也呈爆炸性增长，从 Google 中搜索“软件”关键词，就有 135 000 000 条记录；软件的方向和领域也在不断细化，比如软件架构和平台、软件工程、软件应用，还有软件开发测试等，因此我们可以判断软件的发展趋势是系统化、复杂化，这个趋势使软件能够提供越来越强大的功能，但同时也为我们理解和把握软件带来困难。

但我们做事的原则应该是要把复杂的事情变简单，而不是更复杂，更难理解。当我们试图理解和分析一个复杂的事物的时候，最常用的方法是分 而治之，就是要用一个或多个简单的概念去解释或描述这个复杂的事物，这符合我们人类的认知规律，人们对简单的概念能够理解，那么对简单进行综合和归纳，就形成了对复杂的认知。比如，我们想要让一个没有上过网的人明白什么是“电子邮件”，那就可以告诉他“通过网络发送的邮件”，当然他很有可能对“网络”也一头雾水，那么你可以继续向他解释“打鱼的网”。但在软件领域中，我们却经常搞不明白这个道理，一个刚入门有志于软件性能测试的菜鸟小心翼翼地向一个前辈高人请教什么是性能测试，前辈首先以威严的口气告诉他“性能测试是很复杂的”，然后徐徐道来“性能测试分为负载测试、压力测试、容量测试等”。到这里，我相信可怜的菜同学对性能测试已经更加糊涂了，他在请教问题之前，恐怕还能知道性能测试是测试软件性能的，在得到高

手回答之后，他开始勤奋地请教 Google，没想到 Google 回答他的是更多的名词概念（网上文章都是前辈仙人写的）。随着时间推移，菜同学升级成牛同学，他对性能测试名词概念烂熟于心（Google 功底深厚），并且牛同学又牢牢坚持与时俱进的思想，独立创新了 N 个性能测试概念，并开始向一群小菜粉丝们讲经布道，于是上一幕的画面和对白又开始回放，只是演员变了……这只是一个假想的故事，故事的结局就是通过“学习”，性能测试不仅没有简单，反而越来越复杂了。我们要真正掌握性能测试，那就要避免这样的事情发生，从本质上认识软件性能和软件性能测试。

辩证唯物主义哲学认为，时间和空间是运动着的世间万物的存在形式。大到社会形态，小到个人的活动，都是在一定的空间和时间内进行的。

因此，我们在试图把一件事情表述清楚时，通常要抓住事情的几个关键要素：时间、空间（地点）、人物（主体）、事件。比如“旅行者的一次 长途旅行在两个月内从北京到西藏”，这句话中包含了关键要素，其时间是两个月，空间是北京到西藏，人物是旅行者，发生的事件是旅行者在两个月时间范围内发生空间中的转移；又如“一场足球赛”，这个名词看起来简单，但仍清楚地隐含了三个要素，即：时间，通常是 90 分钟（如果没有加时赛和伤停补时）；空间，足球场内；人物，足球运动员，事件就是在足球规则下可能发生的事情，如进球等。

计算机的出现是人类历史上一次伟大的革命，在哲学“物质”这个名词的外延中又多了一个新型事物——计算机软件。如果我们认识到计算机软件也是万物之一，分析其作为“物质”的性质也逃脱不了自然法则的“紧箍咒”，那么我们同样可以把软件作如下简单的理解：

主体：程序，是人类逻辑思维的物化，表现形式为一系列指令代码。

时间：即使计算机速度再快，任何软件程序每一段代码的运行都是需要时间的，例如从用户的感受来讲，就是程序将运行结果响应给用户的速度。

空间：软件运行的环境，以资源的方式存在，通常是软件以间接或直接的方式占用并使用硬件资源和其他软件资源。

硬件资源主要指运行该软件的硬件平台，有 CPU、内存和存储系统等，如果软件是基于网络架构的，那么硬件还有网络硬件，如交换机、路由器等。

软件资源包括操作系统、开发平台、中间件和数据库等，它们以库文件和 API 的方式提供给应用软件使用。

事件：软件按照用户的要求运行，运行的同时必然要占用时间资源和空间资源。

由于软件代码是人的逻辑思想的表现，所以软件在设计思想和实现方法上也有很大差异。另外，随着软件的发展，产生了各种应用领域的软件，它们之间存在着千丝万缕的关系。从层次上看，有系统软件，应用软件和介于两者之间的中间件。因此一个软件的运行牵涉的因素很多，需要从各个方面分析。

1.1.2. 软件性能的产生

用户能够看到的是软件越来越通用，功能越来越庞大，从哲学角度上看待软件本身，其发展是一个从简单到复杂，从低级到高级，从无序到有序的过程。

在计算机发展的初期，计算机软件对硬件有很强的依赖性，而且还没有广泛的通用性，只有少数的个人或机构才使用软件这个“奢侈品”，当时用户也没有软件性能的概念，通常为了实现软件的功能而不计一切代价。比如，1946 年 2 月 15 日，世界上第一台通用电子数字计算机“埃尼阿克”（ENIAC）在美国研制成功。它当时由 1.8 万个电子管组成，是一台又大又笨重的机器，体重达 30 多吨，占地有两三间教室般大。它当时的运算速度仅为每秒 5000 次加法运算，在现在看来，它占用如此多的资源，又运行得如此慢，在当时却是相当了不起的成就，因为它已经实现了功能——能够做加法运算。可见初期的软件是简单的， 当时用户的要求用现在的眼光来看真有点可怜巴巴，对软件的要求不高，只要能工作就 OK 了。

软件诞生后，短短几十年，软件业奇迹般的高速发展，逐渐走下了高高在上的神坛，广泛应用到人类社会的各个领域，用户也不再把软件看作神秘的玩意，而是普通的商品，开始从经济学的角度来考虑软件产品，这是一个意味深长的变化。讲经济就是要运用投入产出的关系分析和指导软件工程的各种活动和环节，软件运行不能以硬件不计成本为假设，要尽可能地少占用各种硬件资源，同时，软件运行的速度也要尽可能地快，每秒 5000 次加法运算是根本不可想象的，也是不可能被用户接受的。这些其实就是用户的最原始的性能需求。

1.1.3. 功能与性能的关系

首先，软件的性能和功能的源头都是来自于用户的需求。

功能指的是在一般条件下软件系统能够为用户做什么，能够满足用户什么样的需求。拿一个电子邮件系统来讲，用户期望这个软件系统能够提供收发电子邮件、保存草稿、设置偏好等功能，只有这些功能实现了，用户才认为这是他想要的软件。但是随着软件市场竞争的激烈，软件技术的日益提高，系统能不能工作已经是一个最起码的门槛，能够“又好又快”才会得到用户的青睐，而性能则是衡量软件系统“好快”的一个重要考虑因素。“好”就是要为用户省钱，用最小的硬件成本运行软件系统；“快”就是软件响应时间要短，我们的用户都是

急性子，最好一秒钟也不要等。简单地说，性能就是在空间和时间资源有限的条件下，软件系统还能不能工作。

如果把上面邮件的功能和性能需求量化，写成用户需求说明书可能是下面这个样子：

功能：

邮件系统能够支持收发以 30 种语言为标题和正文的邮件，并支持粘贴 10MB 的邮件附件。

性能：

邮件系统能够在 2GB RAM/1GHz CPU 的服务器上，支持 10000 注册用户，日均处理 10000 邮件，响应时间不超过 5 秒/封。

我们来对比一下功能需求说明和性能需求说明，发现两者有一些不同之处：

（1）功能需求中名词和动词多，描述软件主体和动作行为，比如“标题”、“正文”、“收发”、“粘贴”等；

（2）性能需求中对涉及容量和时间词汇多，如“2GB RAM 服务器”、“1000 注册用户”、“5 秒/封”等。

相信我们的读者已经从上面的对比看出功能和性能的区别了，软件性能和功能区别的实质是，软件功能焦点在于软件“做什么”，关注软件物质“主体”发生的“事件”；而软件性能则关注于软件物质“做得如何”，这是综合“空间”和“时间”考虑的方案（资源和速度），表现为软件对“空间”和“时间”的敏感度。认识到性能的这个基本特征对于性能测试人员非常重要，因为在下面的章节中我们将要通过多个“空间”和“时间”的组合，来揭开性能指标的实质和提高的办法。另外，我们也要认清一个事实，软件的性能实现是建立在功能实现的基础之上的。

这就像一个人首先要能跑，这是一个健康的人的正常功能，然后才能参加百米比赛，这就如对人身体的性能考验。而百米比赛隐含了两个要素：一个是运动员有一个一百米的运动空间；另一个是比赛，要跑得足够快，要在短时间内跑完。因此我们说百米比赛其实就是一个空间和时间的综合结果。

“空间”和“时间”是一个哲学中抽象层次较高的概念，在不同的应用范围有不同的诠释。那么在软件理论和实践中，我们怎样理解“空间”和“时间”呢？所谓“仁者见仁，智者见智”，下面我们就分别从用户的角度和软件人员的角度来看一下软件的性能。

1.1.4. 用户眼里的软件性能

软件系统在满足用户强大的功能需求同时，架构和实现上也变得复杂，软件系统经过单机系统时代、客户机服务器系统时代，到现在跨广域网的庞大分布式系统时代，这样的例子在金融、电信系统中随处可见。

系统的业务量大了，就要使用更多的时间和空间资源，在一般情况下不能出现的软件性能问题就暴露出来了，这些问题“不鸣则已，一鸣惊人”，轻则让软件对外不能正常提供服务，重则可能会导致系统的崩溃甚至数据的丢失，这都会给用户带来无法估量的损失。

案例 1

某西部大型油田使用钻井平台数据采集系统，在上线之前已经通过功能测试，但软件系统上线之后，在使用采集的电子数据勘探油层时，总是不能准确地找到油口，导致数百万元的损失。经过研究试验，发现软件从平台采集的数据和手工采集的数据有很大出入，性能测试后，找到根本原因：由于采集过程中产生的数据量非常大，导致软件系统在采集过程中线程死掉，丢失部分数据，最终产生的是一个错误的采集结果，为工程人员提供了错误的判断依据。

案例 2

日本第三大手机运营商——软银移动 2006 年 10 月遇到了麻烦，本指望通过降低手机资费来吸引用户，谁想大量用户蜂拥而至却导致自己的电脑系统陷入瘫痪，软银移动在 10 月 29 日不得不宣布暂停接纳新的用户，直接损失逾亿日元。

用户当然不想看到以上的场景发生在自己的软件系统上，“瘫痪”意味着响应时间过长，不能为客户正常提供服务；数据丢失则是一个不可接受的严重问题，损失几乎不可弥补。因此用户对软件性能的要求日益细化严格，可以说是“与时俱进”。

简单地说，在软件发展的初级阶段，“又要马儿跑，又要马儿少吃草”，这是当时很多用户对软件系统提出的性能要求，“跑”有关时间，“草”有关空间。马儿跑，就是软件系统给用户的响应要快，处理时间要短；马儿少吃草就是软件系统能够尽可能地少占用和消耗资源，诸如内存、CPU 等。因此，测试人员在做性能测试时，往往要把响应时间、内存利用率、I/O 占用率等写在最后测试报告里，因为这是用户最关心的东西。

随着用户的软件质量意识的增强，用户对软件的性能需求也越来越多，越来越细致。这时不仅要让马儿跑，还要马儿能快能慢（软件系统的伸缩性），“路遥知马力”（软件系统在

长时间运行下的稳定性）等。细数起来，如下：

计算性能；

资源的利用和回收；

启动时间；

伸缩性；

稳定性。

计算性能——就是马儿要能跑，要有很快的速度，最好是“日行千里，夜行八百”。对软件系统来讲，计算性能是用户最关心的一个指标，即软件系统有多快。比如，用户会关注软件系统执行一个典型的业务需要花多少时间。我们要给出用户答案，我们的系统完成用户典型操作，比如业务的交易计算，数据的增、删、改、查时间是不是在用户可以接受的范围内。

资源的利用和回收——就是马儿少吃草。软件系统的“草料”就是其依存的硬件和软件资源，硬件资源包括客户端硬件、服务器硬件和网络硬件；软件资源包括操作系统、中间件和数据库等。其中要特别说的是，运行软件系统需要使用到的服务器内存数量，对于整个系统的性能表现是至关重要的。因此，软件系统能否在运行时有效地使用和释放内存是我们考察软件性能的一个重要因素。

对计算机来讲，计算机内存为程序提供运行空间（有代码区和数据区），如果内存不够大，CPU 就不能把全部的数据和程序放到内存里，只好放一部分在内存，一部分放在硬盘中，现用现取，而读取内存和读取硬盘数据的速度要差好几个数量级，这就大大影响了计算机的工作效率。如果还不能理解内存的重要性的话，可以用个形象的例子来说明：

如果 CPU 是个画家，那么内存就是他的工作台。工作台上放着画布（被操作的数据），还有各种画笔、刷子等各种工具（运行的程序）。如果工作台（内存）不能足够大，容纳不下绘画所使用的所有工具，那么画家就需要不时地去储藏室（硬盘等存储设备）里取所需的工具，这就会大大影响绘画的速度。

所以在评价一个系统性能的时候，要特别关注这个系统对内存的使用。

启动时间——这是马儿的加速度问题。用户希望系统进入正常工作状态的时间越短越好，尤其在主备系统中，软件的启动时间直接影响主备的切换效率。而不同软件系统启动时间会不同的。J2EE 系统在第一次启动的时候一般会比较慢，因为期间涉及缓存的加载、JSP 页面的编译、Java class 编译成机器指令等。所以在第一次启动应用感到非常慢是比较正常

的，这也是 J2EE 或者 Java 应用的一个特点。而 C/C++程序直接运行的是二进制机器代码，启动速度就要快一些。

伸缩性——马儿要能快能慢。伸缩性是分析系统性能经常被忽略的一个方面。比如一个系统在 50 个并发用户访问的时候表现正常，但是当并发用户达到 1000 的时候，系统表现如何？服务器的性能是逐渐下降呢，还是在某个拐点附近急剧下降呢？

如图 1-1 所示，该图是一个伸缩性不好的系统的表现，随着并发用户的增加，平均相应时间越来越长。系统最终会达到一个不可用的程度，没有一个用户会接受系统这样的性能表现。

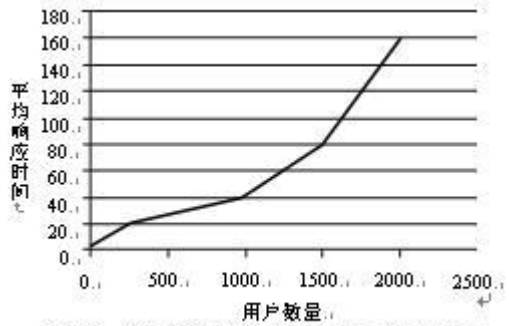


图 1-1　伸缩性不好的用户 - 响应时间图

如图 1-2 所示就是一个伸缩性较好的系统的表现，随着并发用户的增加，平均响应时间逐渐稳定下来。

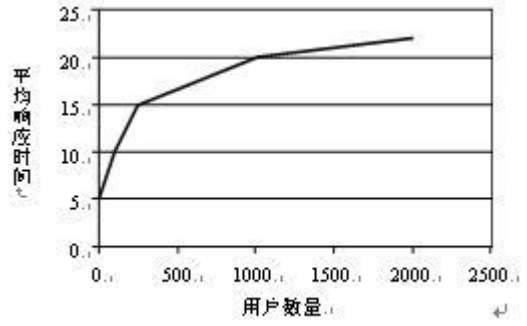


图 1-2　伸缩性良好的用户 - 响应时间图

稳定性——千里马能够“路遥知马力”，而黑马只能够一时跑得快。用户希望自己的软件系统是千里马，而不是黑马。尤其是金融和电信系统，这些系统基本上都是每天 24 小时运转，时时刻刻准备着为用户提供服务。如果它们在运行一段时间后出现了问题，不能响应用户的请求甚至破坏或丢失了数据，那么系统为用户带来的损失是巨大的。这种稳定性问题应该在软件系统上线之前就被考虑并得到解决。

“快”、“好”这只是用户的主观体验，如果能让这些感觉和要求被其他人正确地理解（尤

其是对软件人员），那么就需要用数据把上述用户的感受量化并表达出来，这就是性能指标。

通常，衡量一个软件系统性能的常见指标有：

1．响应时间（Response time）

响应时间就是用户感受软件系统为其服务所耗费的时间，对于网站系统来说，响应时间就是从点击了一个页面计时开始，到这个页面完全在浏览器里展现计时结束的这一段时间间隔，看起来很简单，但其实在这段响应时间内，软件系统在幕后经过了一系列的处理工作，贯穿了整个系统节点。根据“管辖区域”不同，响应时间可以细分为：

（1）服务器端响应时间，这个时间指的是服务器完成交易请求执行的时间，不包括客户端到服务器端的反应（请求和耗费在网络上的通信时间），这个服务器端响应时间可以度量服务器的处理能力。

（2）网络响应时间，这是网络硬件传输交易请求和交易结果所耗费的时间。

（3）客户端响应时间，这是客户端在构建请求和展现交易结果时所耗费的时间，对于普通的瘦客户端 Web 应用来说，这个时间很短，通常可以忽略不计；但是对于胖客户端 Web 应用来说，比如 Java applet、AJAX，由于客户端内嵌了大量的逻辑处理，耗费的时间有可能很长，从而成为系统的瓶颈，这是要注意的一个地方。

那么客户感受的响应时间其实是等于客户端响应时间+服务器端响应时间+网络响应时间。细分的目的是为了方便定位性能瓶颈出现在哪个节点上（何为性能瓶颈，下一节中介绍）。

2．吞吐量（Throughput）

吞吐量是我们常见的一个软件性能指标，对于软件系统来说，“吞”进去的是请求，“吐”出来的是结果，而吞吐量反映的就是软件系统的“饭量”，也就是系统的处理能力，具体说来，就是指软件系统在每单位时间内能处理多少个事务/请求/单位数据等。但它的定义比较灵活，在不同的场景下有不同的诠释，比如数据库的吞吐量指的是单位时间内，不同 SQL 语句的执行数量；而网络的吞吐量指的是单位时间内在网络上传输的数据流量。吞吐量的大小由负载（如用户的数量）或行为方式来决定。举个例子，下载文件比浏览网页需要更高的网络吞吐量。

3．资源使用率（Resource utilization）

常见的资源有：CPU 占用率、内存使用率、磁盘 I/O、网络 I/O。

我们将在 Analysis 结果分析一章中详细介绍如何理解和分析这些指标。

4．点击数（Hits per second）

点击数是衡量 Web Server 处理能力的一个很有用的指标。需要明确的是：点击数不是我们通常理解的用户鼠标点击次数，而是按照客户端向 Web Server 发起了多少次 http 请求计算的，一次鼠标可能触发多个 http 请求，这需要结合具体的 Web 系统实现来计算。

5．并发用户数（Concurrent users）

并发用户数用来度量服务器并发容量和同步协调能力。在客户端指一批用户同时执行一个操作。并发数反映了软件系统的并发处理能力，和吞吐量不同的是，它大多是占用套接字、句柄等操作系统资源。

另外，度量软件系统的性能指标还有系统恢复时间等，其实凡是用户有关资源和时间的要求都可以被视作性能指标，都可以作为软件系统的度量，而性能测试就是为了验证这些性能指标是否被满足。

1.1.5. 软件人员眼里的软件性能

用户恨不能让软件有无限的性能，但作为软件技术人员，我们需清楚地认识到，那种理想化的要求是不可能的。在软件性能方案中，没有什么万能钥匙，软件性能方案充满了辩证的各种矛盾。每种方案和方法几乎都有利有弊。只有把握设计系统的具体环境，明确设计目标，具体问题具体分析，合理平衡各种矛盾，牢牢抓住主要矛盾，才能产生出优化的软件系统性能方案。

在上面的分析中，我们得知软件性能是软件运行空间和时间综合考虑的解决方案。那么其实满足用户的性能需求，只有以下几种方案：

1．消除软件对空间和时间不必要的浪费

一个最明显的例子就是内存泄漏问题，它被开发人员看作是大忌。

严格地说，内存泄漏应该属于软件程序设计的一种缺陷，该缺陷直接导致了程序在运行过程中无法释放不再需要的内存空间，从而造成内存资源浪费，严重的会造成无可用内存，导致系统崩溃。具体来说，当用户程序在运行过程中需要动态获得内存时，操作系统总是从堆（heap）上分配相应的空间给应用，分配的结果是将该堆内存的起始地址通过指针返回给应用。正常情况下，使用完这块内存后，应通过系统调用主动通 知操作系统回收这些堆内

存以便重用。但是，如果由于设计缺陷导致在某些情况下程序没有主动地通知到操作系统，而后应用又失去了对这块内存的引用时，则该堆内存块将成为既不受程序控制，又不能被系统回收重用的“孤儿”内存，这便是我们所指的内存泄漏。

案例 1

```
void foo( )
{
   char *str;
   str = (char*)malloc(32*sizeof(char));
   strcpy(str, "hello world");
   return;
  /* str 所指向的 32 个字节的内存没有被释放，当 foo()返回时造成内存泄漏  */
}
```

解决：C 语言中 malloc 和 free 函数要配对使用。

案例 2

```
void foo()
{
//定义 string1 指针，其指向一个堆上的 100 个字节的内存空间
char *string1 = (char*)malloc(100*sizeof(char));
//定义 string2 指针，其指向一个堆上的 200 个字节的内存空间
char *string2 = (char*)malloc(200*sizeof(char));
scanf("%s", string2);
string1=string2;/*string1 原先指向的 100 个字节的内存没有被释放*/
```

/*而后又被指向 string2 所指的内存块，造成前面 100 个字节的内存泄漏*/

free(string2);

free(string1); /* 这个 free()调用会失败，因为 string1 指向的内存地址与 string2 的相同，而那块内存已经被释放了 */

return 0;

}

解决：在程序堆上分配内存后，要在使用完后及时释放，同时避免野指针的产生，比如 string1。

原理：内存是软件运行的重要的空间资源，内存泄漏实际上是浪费了软件的空间资源。因此，内存泄漏对软件的性能影响十分重要。

另外，对于程序在时间上的浪费，我们通常是采用优化算法和数据结构的解决策略。

案例 3

最近几年，很多知名软件公司在招聘软件测试人员，考察代码能力的时候，内存泄露和算法优化是经常的试题之一。这说明了用户对软件性能的要求越来越严格，已经传递到了软件公司。

2．以空间换时间

软件的高性能并不是凭空产生的，在解决了空间和时间浪费的问题之后，如果用户还有更高的性能要求，我们软件人员只好“偷梁换柱”，做一下调整，而这种调整往往是很灵活的。

空间换时间是软件人员解决性能问题最常见的方法。是在系统功能正常的前提下增大软件空间开销的方法来缩减运行的时间。一般的方法有算法调整、并行计算方法、体系结构方法和一些不是“办法”的办法。

通常的解决方案有 Cache 缓存、数据库的 index 等。

案例 4

一个动态网站服务器总发生 CPU 耗尽的问题，因此造成给用户的响应缓慢或者长时间没有响应，进而引起 Server 的宕机。经调查分析，网站首页是个 PHP 程序，每次用户访问

都要多次查询数据库，也没有 Cache 机制，数据库查询负荷过高，耗尽 CPU。

解决：改写网站首页以及部分频繁访问的程序，增加 Cache 机制，减少数据库访问。

原理：将常用数据放在服务器的内存中，虽然增加了内存的开销，但带来了时间上的优化，对用户而言，提高了处理速度。

3．以时间换空间

时间换空间的方案解决性能问题的情形比较少。有时会出现在对内存要求十分苛刻的地方，比如嵌入式操作系统中。

案例 5

程序设计的要求是不设中间变量，交换两个变量的值。

我们通常的中间变量的解决方案是：

```
Void swapOne(int *a, int *b)
{
   Int temp;
   Temp = *a;
   *a = *b;
 *b= temp;
}
```

但这里需要在程序中为 temp 变量在栈上分配一个空间。可不可以不用这个 temp 变量呢？

解决：

修改程序如下：

```
void swapTwo(int *a，int *b)
```

```
{
*a=*a+*b;
*b=*a-*b;
*a=*a-*b;
}
```

原理：修改之后，多了运算复杂度，但没有使用第三方变量，减少了空间的占用。

以上是我们从简单的程序例子来理解性能解决方案，但现实要远远复杂得多，因为随着软件系统功能的复杂强大，软件的规模也在不断扩大，我们不可能完全自己开发程序，很多时候是利用已有的平台和中间件资源。在这种场景下，我们应该怎样考虑性能问题呢？

第一，软件系统设计的架构及技术平台

软件在设计阶段一旦决定采用哪种架构和技术，其性能也就注定只能在一定的范围内变动了。这就是“先天”因素。比如在上节讲到的一个删除/增加数据的业务操作，如果用户对时间非常苛刻，密集型计算、在线的大数据量统计和分析等应用，这些场景通常 J2EE 不能够很好地解决，使用 C++或者其他平台搭建会更合理些。如果在这些场景下硬要采用 J2EE 架构，那么开发和设计人员如何绞尽脑汁，优化设计和程序，也不会满足用户的性能要求。

第二，中间件的设置和优化

这里的中间件是广义的中间件，是应用程序调用的第三方软件，包括操作系统、数据库、Web 服务器、消息服务器等。我们不能改变中间件的程序，只能通过调优手段来提高它所支持的软件系统的性能。

第三，硬件的配置

这里包括服务器硬件配置和网络环境。服务器硬件包括内存、CPU 等，网络环境有交换机、路由器等。

1.2. 软件性能测试

在上一节中，我们知道软件系统的性能问题多种多样，这给用户带来巨大的风险，那么我们如何能够在软件系统上线之前 ，找出软件中潜在的性能问题呢？目前软件性能测试是

发现软件性能问题最有效的手段，而完备有效的性能测试是最关键的，在本节中我们将从流程和技术的角度解析如何构建一个高效的性能测试模型。

1.2.1. 性能测试在软件测试的周期位置

首先，软件性能测试属于软件测试范畴，存在于软件测试的生命周期中。一个软件的生产过程通常遵循 V 型图，如图 1-3 所示。

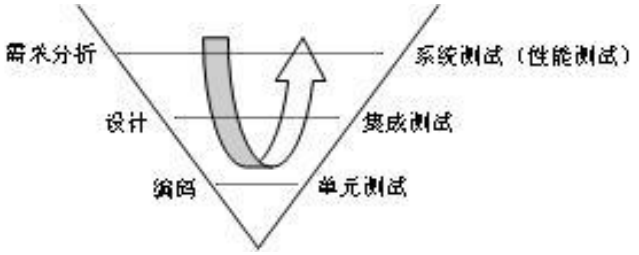


图 1-3　软件开发-测试 V 型图

在通常的软件生产周期中，先由用户提出用户需求或经系统分析核定以后提出系统需求，开发人员再经过需求分析提出软件需求规格说明，进行概要设计，提出概要设计说明，进行详细设计，提出详细设计说明，最后就是对每个模块进行编码。到测试阶段，测试按照开发过程逐阶段进行验证并分步实施，体现了从局部到整体、从低层到高层逐层验证系统的思想。对应软件开发过程，软件测试步骤分为代码审查、单元测试、集成测试、系统测试。

而性能测试就属于软件系统级测试，其最终目的是验证用户的性能需求是否达到，在这个目标下，性能测试还常常用来做：

（1）识别系统瓶颈和产生瓶颈的原因；

（2）最优化和调整平台的配置（包括硬件和软件）来达到最高的性能；

（3）判断一个新的模块是否对整个系统的性能有影响。

提示：系统瓶颈：

瓶颈本来是指玻璃瓶中直径较小并影响流水速度的一段，用它来比喻软件系统中出现性能问题的节点是很形象的，比如一个典型的分布式系统架构如图 1-4 所示。

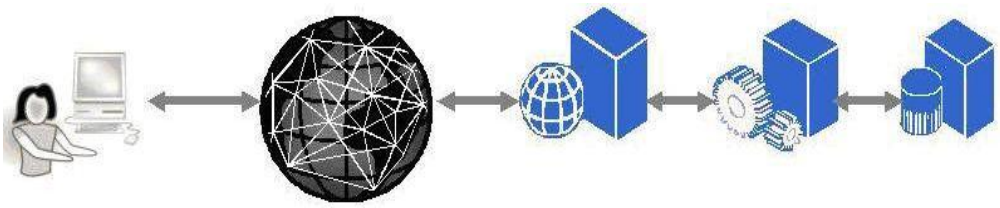


图 1-4　软件系统压力流动图

如果把软件系统看作是交通系统，那么网络就是一条条大道，客户端、防火墙、负载均衡器、Web 服务器、应用服务器（中间件）、数据库等各个系统节点就是交通要塞，客户的请求和数据就像在道路上行驶的车辆，如果在某处发生堵车，那么整个交通系统都会不畅。在 这个时候，我们就要分析是哪里出了问题，是道路不够宽，还是某处立交桥设计不合理而引起堵塞等。找到问题的关键点，那么此关键点就是本系统的瓶颈。软件系统也是如此，我们做性能测试的大部分工作都是为了寻找这个瓶颈到底在何处。

需要注意的是，软件的性能瓶颈可能不止一处。

作为软件测试的一种，软件测试的规则同样适用于性能测试中：

（1）确定预期输出是测试必不可少的一部分

如果事先无法肯定预期的测试结果，往往会把看起来似是而非的东西当作正确结果。必须提倡用事先精确对应的输入和输出结果来详细检查所有的输出。对于性能测试来说，预期输出就是用户的性能需求，一份明确的性能需求是成功性能测试的先决条件。

（2）必须彻底检查每一个测试结果

事实上，在最终发现的错误，有相当一部分在前面的测试中已经暴露出来了，然而由于人们未能细心检查先前的测试结果而遗漏了。

一段程序中存在的错误概率与在这段程序中发现的错误数呈正比。

这是 pareto 原则应用于软件测试，也包括性能测试，即性能测试发现的错误中的 80%很可能集中在 20%的程序模块中。

（3）穷举测试是不可能的

在性能测试中不可能覆盖每一个功能部分，这也意味着有性能问题的模块可能被忽略掉，这样的话，我们在设计性能测试案例时，应该采取一些策略和技巧，使用尽可能少的性能测试用例，发现尽可能多的 bug。这方面内容我们将在本书的第 10 章中介绍。

在具有软件测试共性的同时，性能测试也有自身的一些特点。

1．性能测试不是功能测试

性能测试不要求也无法做到覆盖软件所有的功能，通常我们只是对系统中某些功能或模块做性能测试。一般的，我们在选择性能测试案例时需要遵循以下的原则：

（1）基本且常用的

比如，一个 E-mail 系统，基本且常用的功能有注册、登录、收邮件、查询邮件，用户使用这些功能的频率较高，要做性能测试。而高级查询、过滤器、邮件列表等功能被使用的次数较少，就可以不做性能测试，或者进行性能测试的优先级低一些。

（2）对响应时间要求苛刻的

这样的要求经常出现在金融和电信等对实时性要求比较高的系统中。比如，从手机呼叫开始，经过基站、核心网，再到被叫手机响铃，整个系统的处理时间应该在用户能接受的范围内。另外，一个负责和手机通信的基站在发生故障或掉电后，要能很 快地恢复工作状态。这些功能都对时间有着严格的要求，一定要做性能测试，当然实际运作时，电信系统上线时所做的性能测试不仅仅限于这些功能。

将这些功能细分就是性能测试中的事务（Transaction）。关于事务这个概念我们在后面的章节中将详细阐述。

2．性能测试属于系统级测试

从 V 型图可以看到，性能测试属于系统级测试。那么性能测试是基于单元测试、集成测试、功能测试等都已经完成的基础上，站在用户的角度去测试整个系统的。这包含两个含义：

第一，性能测试是“两头在外”，软件性能需求不仅直接来自用户，最终目的也是服务于用户。通过性能测试这个过程，从上面我们讲到用户的需求和性能测试指标的对应关系，就可以看出。

第二，性能测试开始的必要条件是软件系统已经处于一个比较稳定的状态，系统架构、主要代码、中间件等都不再有大的变化，否则会给性能测试带来很大的风险。

思考

基于以上事实，我们应该在软件流程什么阶段开始性能测试？结合自己的实际工作进行分析。

1.2.2. 性能测试策略揭秘

谈到“策略”，这是如今很火、使用较多的一个词。不光在 IT 领域，其他各个行业中也都有各种各样的策略，如营销策略、风险规避策略等。策略即谋略、手段、方法，表现为权宜的行动路线、指导原则或过程。

做事情讲策略，这是一种智慧，是人们聪明起来的表现，但当越来越多的策略“概念化”的时候，我们不得不去思考我们到底要达到什么样的目标，什么样的策略才是我们需要的。

案例

引用网上一位哲人说的话：“概念只是为了方便人们理解和研究世界万物事物而制造的工具，而最终结果将使概念不再需要，就如同庄子所说的得意而忘言”。语言就是一种包装材料，它包装的是某种含义。因为人类传递信息必须使用语言，所以我们在研究的时候不得不借助于这种包装，但是当人的思维能力具备了打开包装直接取得内部的含义的时候，语言就变得多余了。这时候再关注于语言和概念本身就成了买椟还珠的现代版了。

因此，我们应该关注的不是概念本身，而是概念背后的含义。理解了含义，再冠予它什么样的名词头衔，如“攻略“，对于我们都无关紧要了。而理解一个概念，我们可以靠 WWH 方法，即对概念的三个问题：Why、What、How。

好，言归正传，回到软件性能测试策略中来。在性能测试过程中，只要有事情做，就会有策略，如设计用例有设计策略，执行时有执行策略，调优时还有调优策略。为了不产生混淆，我们要说明的是，在本节中讨论的策略是性能测试设计策略。

Why（为什么会有不同的策略）

在软件性能一节中，我们看到软件的性能来自软件对空间和时间的综合方案，这种组合是很多的，因此用户的软件性能需求可能会多种多样。对于软件人员，我们做性能测试也要因地制宜，根据不同的性能需求，选择不同的测试策略。

What（什么是性能测试设计策略）

验证性能需求是测试目的，测试策略即已经被证明是行之有效的测试方法。

How（怎样实施）

常见的性能测试方法有以下几种：

1．负载测试

在这里，负载测试指的是最常见的验证一般性能需求而进行的性能测试，在上面我们提到了用户最常见的性能需求就是“既要马儿跑，又要马儿少吃草”。因此负载测试主要是考察软件系统在既定负载下的性能表现。我们对负载测试可以有如下理解：

（1）负载测试是站在用户的角度去观察在一定条件下软件系统的性能表现。

（2）负载测试的预期结果是用户的性能需求得到满足。此指标一般体现为响应时间、交易容量、并发容量、资源使用率等。

2．压力测试

压力测试是为了考察系统在极端条件下的表现，极端条件可以是超负荷的交易量和并发用户数。注意，这个极端条件并不一定是用户的性能需求，可能要远远高于用户的性能需求。可以这样理解，压力测试和负载测试不同的是，压力测试的预期结果就是系统出现问题，而我们要考察的是系统处理问题的方式。比如说，我们期待一个系统在面临压力的情况下能够保持稳定，处理速度可以变慢，但不能系统崩溃。因此，压力测试是能让我们识别系统的弱点和在极限负载下程序将如何运行。

例子：负载测试关心的是用户规则和需求，压力测试关心的是软件系统本身。对于它们的区别，我们可以用华山论剑的例子来更加形象地描述一下。如果把郭靖看作被测试对象，那么压力测试就像是郭靖和已经走火入魔的欧阳峰过招，欧阳锋蛮打乱来，毫无套路，尽可能地去打倒对方。郭靖要能应对住，并且不能丢进小命。而常规性能测试就好比郭靖和黄药师、洪七公三人约定，只要郭靖能分别接两位高手一百招，郭靖就算胜。至于三百招后哪怕郭靖会输掉那也不用管了。他只要能做到接下一百招，就算通过。

思考

我们在做软件压力测试时，往往要增加比负载测试更多的并发用户和交易，这是为什么？

3．并发测试

验证系统的并发处理能力。一般是和服务器端建立大量的并发连接，通过客户端的响应

时间和服务器端的性能监测情况来判断系统是否达到了既定的并发能力指标。负载测试往往就会使用并发来创造负载，之所以把并发测试单独提出来，是因为并发测试往往涉及服务器的并发容量，以及多进程/多线程协调同步可能带来的问题。这是要特别注意，必须测试的。

4．基准测试

当软件系统中增加一个新的模块的时候，需要做基准测试，以判断新模块对整个软件系统的性能影响。按照基准测试的方法，需要打开/关闭新模块至少各做一次测试。关闭模块之前的系统各个性能指标记下来作为基准（Benchmark），然后与打开模块状态下的系统性能指标作比较，以判断模块对系统性能 的影响。

5．稳定性测试

“路遥知马力”，在这里我们要说的是和性能测试有关的稳定性测试，即测试系统在一定负载下运行长时间后是否会发生问题。软件系统的有些问题是不能一下子就暴露出来的，或者说是需要时间积累才能达到能够度量的程度。为什么会需要这样的测试呢？因为有些软件的问题只有在运行一天或一个星期甚至更长的时间才会暴露。这种问题一般是程序占用资源却不能及时释放而引起的。比如，内存泄漏问题就是经过一段时间积累才会慢慢变得显著，在运行初期却很难检测出来；还有客户端和服务器在负载运行一段时间后，建立了大量的连接通路，却不能有效地复用或及时释放。

6．可恢复测试

测试系统能否快速地从错误状态中恢复到正常状态。比如，在一个配有负载均衡的系统中，主机承受了压力无法正常工作后，备份机是否能够快速地接管负载。可恢复测试通常结合压力测试一起来做。

提示：每种测试有其存在的空间和目的。当我们接手一个软件项目后，在有限的资源条件下，选择去做哪一种测试，这应该根据当前软件过程阶段和项目的本身特点来做选择。比如，在集成测试的时候要做基准测试，在软件产品每个发布点要做性能测试。

1.3. 如何做性能测试

一个项目要取得成功是困难的，因为成功的项目需要多个因素和条件来支持；而一个项目失败却很容易，只要若干因素之中的一个出现问题，就有可能导致项目失败。比如中途测试人员发生变化，性能指标未和用户达成统一理解等。笔者还曾看过一个例子，因为测试报告的格式与用户要求的格式不一致，而不得不重新再执行一次所有的性能场景，来采集用户

要的数据。

实际上，当我们做过的性能测试项目越多，就会发现越多的因素可能会影响性能测试项目的成败，甚至可以是千奇百怪的。

在本节中，我们主要是寻找出不同性能测试项目的共性，而总结出一个具有普遍意义的性能测试过程。遵循过程做性能测试，在大多数情况下可以有效地规避风险，并能取得比较好的性能测试结果。这当然不是意味着我们有了这个过程，就不考虑其他因素了，只是说每个项目都会有自己的独特因素要考虑，尽管这些因素可能很重要，但它们并不在本节的讨论范围内。

在给出此过程模型之前，我们要澄清两点事实：

第一，性能测试过程从何时开始，又在何时结束？

这是一个基本而重要的问题。

在各种书籍和资料中，有关性能测试过程的描述不尽一样：

比如 LoadRunner 手册中提供的过程是：计划测试→测试设计→创建 VU 脚本→创建测试场景→运行测试场景→分析结果。

而在 Segue 中提供的性能测试过程，是一个 try-check 过程，即：评估需求→开发测试→建立基线→执行测试→分析结果→回归测试→测试结束。

上面 LoadRunner 和 Segue 描述各自的性能测试过程最大的区别不在于工具部分，而是在于两者过程的入口和出口条件不一致。这使得它们其实在描述两件事情，或者说是在描述一个事情的两个部分。

在 CMM 中，软件测试和软件设计、编码一样，隶属于软件工程过程，而需求分析过程在软件工程过程之前。这就隐含着一个默认的先决条件：在 CMM 这个体系下，产品在进入软件测试阶段的时候，软件需求是已经明确下来并文档化了的。

实际情况却经常并非如此，同样是软件需求，软件功能需求在进入测试阶段就已经产生了各种文档，包括需求文档和设计文档，确保功能需求是详细、明确、无二义性的；而软件性能需求往往进入了性能测试阶段还不明确（可参见 Controller 一章开篇的例子）。这会给性能测试项目带来很大的风险。

因此，我们应该突破已有的理论束缚，寻找更合适的性能测试过程模型。经过对多个性

能测试项目的实践经验总结，我们在本节提出 GAME（A）性能测试过程模型，其开始于软件需求分析阶段，非常符合目前国内的性能测试实践。

第二，性能测试本身有没有质量？

以前我们总是讨论软件产品的质量、开发代码的质量，但对软件测试的质量却鲜有提及。我们知道“一个好的测试用例是发现了一个原先未发现的 bug”，这其实是对用例质量的度量。软件性能测试项目也有质量，并可以度量。下面是部分度量的方法：

（1）性能测试耗费的资源，包括时间、人力、物力。

（2）性能测试中发现的 bug 数目，以及各自的级别。

（3）软件系统交付用户，在生产环境运行后发现的性能 bug 数目、级别。

而一个好的性能测试过程模型对提高性能测试质量是很有帮助的。

GAME（A）性能测试过程模型：

G：Goal，目标

A：Analysis，分析

M：Metrics，度量

E：Execution，执行

（A）：Adjust，调整。E 执行失败后才进入 A 阶段，并且涉及的大多是有关开发和系统管理工作，因此 A 设为隐式。

性能测试过程模型如图 1-5 所示。

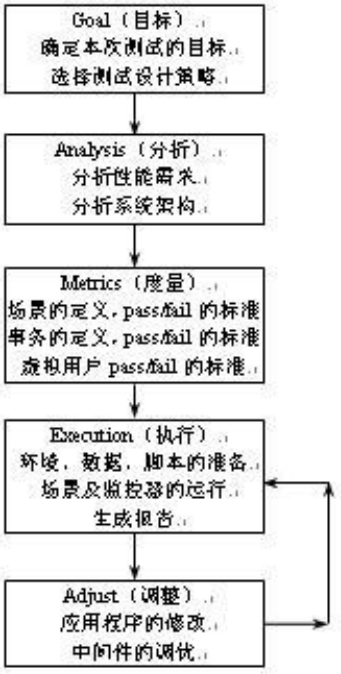


图 1-5　性能测试 GAME(A)模型

1.3.1. Goal（定义目标）

制定一个明确而详细的测试目标是性能测试开始的第一步，也是性能测试成功的关键。

本步骤的开始时间：需求获取阶段

本步骤的输入：性能需求意向

本步骤的输出：明确的性能测试目标和性能测试策略

常规的性能测试目标有以下几种：

（1）度量最终用户响应时间

查看用户执行业务流程以及从服务器得到响应所花费的时间。例如，假设我们想要检测：系统在正常的负载情况下运行时，最终用户能否在 20 秒内得到所有请求的响应。图 1-6 显示了一个银行应用程序的负载和响应时间度量之间的关系。

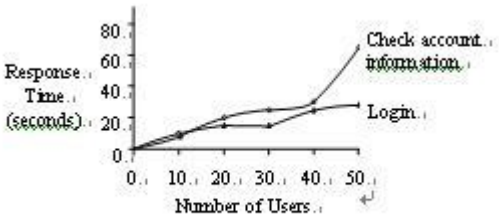


图 1-6　用户数-响应时间关系图

（2）定义最优的硬件配置

检测各项系统配置（内存、CPU 速度、缓存、适配器、调制解调器）对性能的影响。了解系统体系结构并测试了应用程序响应时间后，您可以度量不同系统配置下的应用程序响应时间，从而确定哪一种设置能够提供理想的性能级别。

例如，您可以设置三种不同的服务器配置，并针对各个配置运行相同的测试，以确定性能上的差异：

配置 1：1.2GHz、1GB RAM

配置 2：1.2GHz、2GB RAM

配置 3：2.4GHz、1GB RAM

（3）检查可靠性

确定系统在连续的高工作负载下的稳定性级别。强制系统在短时间内处理大量任务，以模拟系统在数周或数月的时间内通常会遇到的活动类型。

（4）查看硬件或软件升级

执行回归测试，以便对新旧版本的硬件或软件进行比较。您可以查看软件或硬件升级对响应时间（基准）和可靠性的影响。注意：此回归测试的目的不是验证升级版的新功能，而是查看新版本的效率和可靠性是否与旧版本相同。

（5）确定瓶颈

您可以运行测试以确定系统的瓶颈，并确定哪些因素导致性能下降，例如，文件锁定、资源争用和网络过载。将 LoadRunner 与新的网络和计算机监视工具结合使用以生成负载，并度量系统中不同点的性能，最终找出瓶颈所在的位置。

（6）度量系统容量

度量系统容量，并确定系统在不降低性能的前提下能提供多少额外容量。如图 1-7 所示，要查看容量，您可以查看现有系统中性能与负载间的关系，并确定出现响应时间显著延长的位置。该处通常称为响应时间曲线的“拐点”。

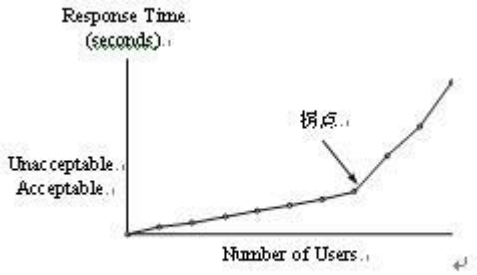


图 1-7　用户数-响应时间拐点图

我们根据不同的测试目标去选择合适的性能测试设计策略。比如，“度量最终用户响应时间”可以采用负载测试策略，“检查可靠性”就可以用压力测试策略，等等。

1.3.2. Analysis（分析）

本步骤的开始时间：需求分析阶段和性能测试启动阶段

本步骤的输入：性能需求

本步骤的输出：达成一致的性能指标列表，性能测试案例文档

1．分析性能需求

在这里，要定义性能测试的内容，细化性能需求。

客户、需求分析人员和测试工程师一起起草一个性能需求标准，对此标准获得一致认同。此标准将用户的需求细化、量化，并能在测试中作为判断依据。

比如，对于负载测试来说，可以从以下角度来细化需求，逐步找出测试关键点。

测试的对象是什么，例如“被测系统中有负载压力需求的功能点包括哪些？”；“测试中需要模拟哪些部门用户产生的负载压力？”等问题。

系统配置如何，例如“预计有多少用户并发访问？”；“用户客户端的配置如何？”；“使用什么样的数据库”；“服务器怎样和客户端通信？”。

应用系统的使用模式是什么，例如“使用在什么时间达到高峰期？”；“用户使用该系统

是采用 B/S 运行模式吗？”；“网络设备的吞吐能力如何，每个环节承受多少并发用户？”等问题。

最后得出的性能测试指标标准至少要包含测试环境、业务规则、期望响应时间等。

2．分析系统架构

对硬件和软件组件、系统配置以及典型的使用模型有一个透彻的了解。结合性能测试指标标准，生成性能测试用例。（可参看第 10 章“进阶 LoadRunner 高手”的用例设计部分。）

1.3.3. Metrics（度量）

本步骤的开始时间：性能测试设计阶段

本步骤的输入：细化的性能指标和性能测试案例

本步骤的输出：和工具相关的场景度量、交易度量、监控器度量和虚拟用户度量等

度量是非常重要的一步，它把性能测试本身量化。这个量化的过程因测试工具的不同而异。

（1）场景的定义，pass/fail 的标准

测试场景包含了性能测试的宏观信息，有测试环境、运行规则和监控数据等。具体可表现为历史数据记录数、虚拟用户数、虚拟用户加载方式、监控指标等。

（2）事务（Transaction）的定义，pass/fail 的标准

事务用来度量服务器的处理能力。事务定义应该从性能指标标准而来，是性能指标的具体体现。事务的定义是很重要的，不同的定义会导致不同的 TPS 结果。

提示：使用性能测试工具执行性能测试之后，我们能看到的是 pass/fail 的用户数、pass/fail 的事务数。而这些 pass/fail 的标准应该在执行性能测试之前就应该被定义好。比如，LoadRunner 默认的 pass/fail 标准是基于协议层的，而我们要的 pass/fail 可能是业务级的，需要在业务层上进行判断来决定是 pass 还是 fail。另外，还可能由于案例的关联性引起的 pass/fail，如果两个案例之间有关联，A 脚本负责向数据库插入数据，B 脚本负责查询数据，A 要是 fail 了，导致 B 也会 fail，虽然 B 本身不一定有错。解决这个问题，一方面可以削弱脚本之间的关联性，另一方面也可以通过增强脚本的健壮性来达到。

（3）虚拟用户 pass/fail 的标准

虚拟用户是性能测试工具中的一个普遍的概念，虚拟用户负责执行性能测试脚本，在这里应该定义虚拟用户遇到何种情况，选择 fail 或 pass，即退出或通过。

1.3.4. Execution（执行）

本步骤的开始时间：软件测试执行阶段

本步骤的输入：场景、交易、虚拟用户等设置信息

本步骤的输出：测试报告

执行测试包含两个工作：

1．准备测试环境、数据和脚本

测试环境：硬件平台和软件平台。

测试数据：包括初始测试数据和测试用例数据两部分。表现为 SQL 脚本、Excel 文件等。

提示：测试环境直接影响测试效果，所有的测试结果都是在一定软硬件环境约束下的结果，测试环境不同，测试结果可能会有所不同。需要注意：如果是完全真实的应用运行环境，要尽可能降低测试对现有业务的影响；如果是建立近似的真实环境，要首先达到服务器、 数据库以及中间件的真实，并且要具备一定的数据量，客户端可以次要考虑。实施负载压力测试时，需要运行系统相关业务，这时需要一些数据支持才可运行业务，这部分数据即为初始测试数据。有时为了模拟不同的虚拟用户的真实负载，需要将一部分业务数据参数化，这部分数据为测试用例数据。

测试脚本：用性能测试工具生成脚本。

2．运行场景和监控性能

运行性能测试场景，并监控设定好的数据指标，最终生成测试报告。按照定义好的场景 pass/fail 标准来判断性能测试是否通过。如果未能通过，进入步骤 5（Adjust）。

1.3.5. Adjust（调整）

本步骤的开始时间：第一轮性能测试结束后，而且没有通过的条件下

本步骤的输入：测试报告和测试结果数据

本步骤的输出：性能问题解决方案

调整包含两个意思：应用程序修改和中间件调优。

中间件调优可考虑如下因素操作系统调优：

数据库调优；

内存升级；

CPU 数量；

代码调优；

Cache 调优。

提示：解决一个性能瓶颈，往往又会出现另外的瓶颈或者其他问题，所以性能优化更加切实的目标是做到在一定范围内使系统的各项资源使用趋向合理和保持一定的平衡。

系统运行良好的时候恰恰也是各项资源达到了一个平衡体，任何一项资源的过度使用都会造成平衡体系破坏，从而造成系统负载极高或者响应迟缓。比如 CPU 过度使用会造成大量进程等待 CPU 资源，系统响应变慢，等待会造成进程数增加，进程增加又会造成内存使用增加，内存耗尽又会造成虚拟内存使用，使用虚拟内存又会造成磁盘 IO 增加和 CPU 开销增加（用于进程切换、缺页处理的 CPU 开销）。

从以上内容可以看出，目前 GAME（A）模型有两个优势：第一，灵活，每个过程都有自己的关注点，可以根据不同的项目特点增加或删除关注点；第二，通用，不依赖于具体的工具。目前 GAME（A）关注性能测试技术，比较简单，将来可以进行扩展，同样使用 GAME（A）模型关注性能测试的时间、人力等资源问题。

1.4. 性能测试工具的评估和选择

我们可以看到，性能测试和一般功能测试不同的是，性能测试的执行是基本功能的重复和并发，因此我们在性能开始之前需要模拟多用户，在性能测试进行时要监控指标参数，同时性能测试的结果不是那么显而易见，需要对数据进行分析。这些特点决定了性能测试更适合通过工具来完成。市场上涌现出越来越多的压力自动化测试工具，古人云“工欲善其事，必先利其器”，一个测试工具能否满足测试需求，能否达到令人满意的测试结果，是选择测试工具要考虑的最基本的问题。

我们这里讨论的主要是一些比较成熟的性能测试软件产品，都已经在市场上占有了一定的份额，得到了用户的认可。

如表 1-1 所示为主要的性能自动化测试工具。

表 1-1　主要的性能自动化测试工具

工具名	公司（组织）	License	描述
WAS (Web Application Stress Tool)	Microsoft	需要	适用于 B/S 架构，模拟浏览器请求
Qaload	Compuware	需要	支持多种系统架构
LoadRunner	Mercury Interactive	需要	支持多种系统架构
Astra quick test	Mercury Interactive	需要	支持多种系统架构
OpenSTA	OPENSTA 组织	开源	支持 HTTP 协议
Jmeter	Apache	开源	全面支持 Java

对于测试人员来说，要么自己开发性能测试工具，要么选择购买市场上已有的性能测试工具。在这里，我们要讨论的只是在选择性能测试工具时，需要考虑哪些因素。这些因素都想清楚了，然后才可以做决定。

1.4.1. 测试预算 VS 工具价格

性能测试的成本与收益比是选择性能测试工具的根本条件。这其实是在考虑“要不要用”的问题。如果购买一套价格几十万的性能测试工具只是为了去做一个几万元预算的性能测试项目，那么无论这个工具再强大，也不会被采用。

1.4.2. 协议、开发技术、平台、中间件 VS 工具的支持

要确定性能测试工具是否支持我们的被测软件系统，这其实是在考虑“能不能用”的问

题。考虑因素有被测软件系统使用的协议、采用的技术、基于的平台、调用的中间件。这些都需要性能测试工具有效的支持。

1.4.3. 工具可使用的复杂程度 VS 项目计划的影响

熟悉并使用一个性能测试工具，是需要花费人力和时间等资源的，项目计划中要有相应的资源准备。这其实是在弄清“如何用”的问题。

2. LoadRunner 入门

LoadRunner 是一个强有力的压力测试工具。它的脚本可以录制生成，自动关联；测试场景可以面向指标，多方监控；测试结果可以用图表显示，并且可以拆分组合。

作为专业的性能测试工具，通过模拟成千上万的用户对被测系统进行操作和请求，能够在实验室环境中重现生产环境中可能出现的业务压力，再通过测试过程中获取的信息和数据来确认和查找软件的性能问题，分析性能瓶颈。

2.1. LoadRunner 创建测试脚本

开发 LoadRunner 脚本需要经过图 2-1 所示的几个步骤。

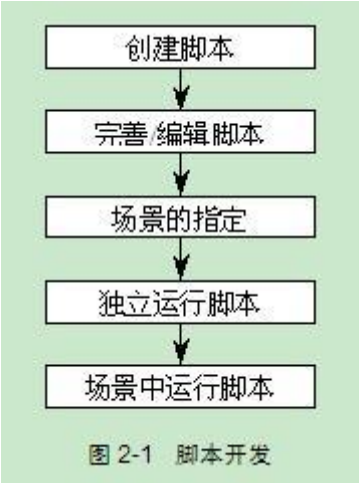


图 2-1 脚本开发

在录制脚本时要遵循以下录制原则：

1．提高脚本执行效率

所录制的脚本内容要精练，而且是用户的真实操作，不可增加多余或重复性的操作，这

样的脚本执行起来更能准确地模拟用户的真实行为，减少了执行时间，执行结果更准确。

2．录制具有代表性的功能

在一个软件中有很多不同的功能，但要录制所有的功能几乎是不可能的，所以要选择常用的、使用频率较高的业务功能来进行测试。

3．选择具有影响的事务

测试人员要对被测功能具有一定的认识和了解，选择一些对于整个测试过程中有影响的事务来测试，否则测试结果是无意义的。

当启动 Visual User Generator 后会出现选择脚本类型的对话框，在此对话框中，请选择我们常用的脚本类型，也就是 Web（HTTP/HTML）协议，这是最为常见的。以下脚本介绍以此类型为例。

2.1.1. 录制普通脚本

启动 Visual User Generator，在弹出的对话框中选择需要新建的协议脚本，通过 VuGen 可以采用单协议或多协议模式，进行脚本的录制。选择单协议还是多协议，根据测试程序的实际需要而定。

1．选择协议

采用单协议模式时，VuGen 将只录制指定的协议；采用多协议模式时，VuGen 将录制多个协议中的操作。下列协议支持多协议脚本：COM、FTP、IMAP、Oracle NCA、POP3、RealPlayer、Window Sockets（原始）、SMTP 和 Web。“双协议 Web/Web Services”的引擎使用一种不同的机制，应视为单协议，不能与其他多协议类型结合使用。

各种 Vuser 类型之间的另一个区别是多操作支持功能。大多数协议都可支持多个操作部分，如 Oracle NCA、Web、RTE、General（C Vusers）、WAP、i-Mode 和 VoiceXML 等协议。

对于大多数 Vuser 类型，在每次录制时都会新建一个 Vuser 脚本，而不能在现有脚本中进行录制。但是，在录制 Java、CORBA-Java、RMI-Java、Web、WAP、i-mode、Voice XML、Oracle NCA 或 RTE Vuser 脚本时，可以在现有脚本中进行录制。

创建脚本时，单击“New”（新建）打开“New Virtual User”（新建 Vuser）对话框，该对话框可提供选择录制脚本协议的快捷方式。

（1）单协议脚本：创建单协议 Vuser 脚本，这是“Startup”（启动）对话框打开时的默认选项。从 Vuser 生成器的“类别”中进行选择，并选择录制脚本的协议，如图 2-2 所示。

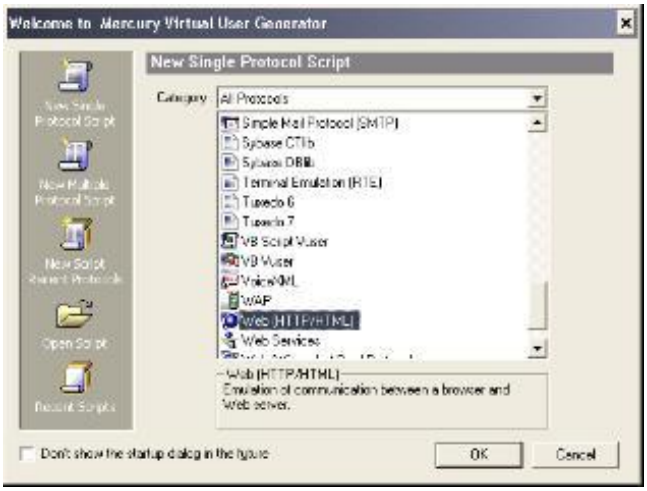


图 2-2 选择单协议脚本

（2）多协议脚本：创建多协议 Vuser 脚本，VuGen 将显示所有可用的协议。选择一个协议后，单击右箭头，将其移入“Selected Protocols”（选定的协议）部分中，如图 2-3 所示。

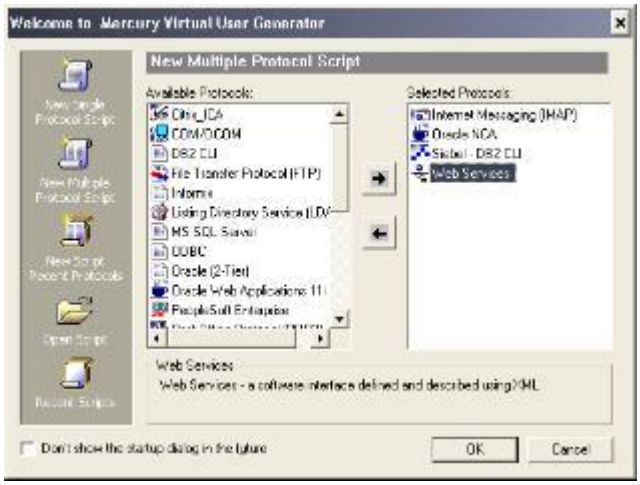


图 2-3 选择多协议脚本

（3）使用最近使用过的协议新建脚本：从最近创建脚本的协议中选择已经使用过的协议，并且这些协议已经体现了录制脚本类型，如图 2-4 所示。

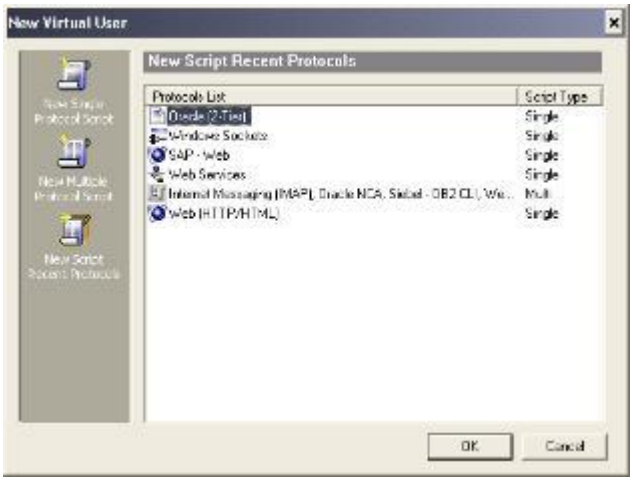


图 2-4 选择最近使用的协议

2.1.2. 录制 Web Services 脚本

在进行性能测试时，大部分对 Web 性能测试，选择“Web（HTTP/HTML）”协议即可，但录制完脚本后，回放脚本过程中有时会发生中断或停止的情况，查看错误时，如果无法找到 SOAP 文件字样时，就需要考虑更换脚本录制协议了。通常首先考虑更换 Web Services 协议，再次录制脚本，问题就相应解决了。

在录制 Web Services 脚本前，首先对 Web Services 做一个简要的介绍，这样有助于读者或者测试人员能够更好地利用 Web Services 协议录制脚本。

1．什么是 Web Services

Web Services 是一种构建应用程序的普通模型，并能在所有支持 Internet 通信的操作系统上实施运行。Web Services 令基于组件的开发和 Web 的结合达到最佳，基于组件的对象模型，如：分布式组件对象模型（Distributed Component Object Model, DCOM）、远程方法调用（Remote Method Invocation, RMI）、互联网内部对象请求代理协议（Internet Inter-Orb Protocol, IIOP）都已经发布很长时间，但是它们都依赖于特殊对象模型协议。而 Web Services 利用 SOAP 和 XML 对这些模型在通信方面作了进一步的扩展，以消除特殊对象模型的障碍。

进一步地，Web Services 还基于 HTTP 和 SOAP 协议，使得 Web 用户通过 Web 调用的方法使用 SOAP 和 HTTP 来调用远程对象，确保业务数据得以在 Web 上传输。

2．Web Services 结构

客户根据 WSDL 描述文档，会生成一个 SOAP 请求消息。Web Services 都是放在 Web

服务器（如 IIS）后面的，客户生成的 SOAP 请求会被嵌入在一个 HTTP POST 请求中，发送到 Web 服务器，Web 服务器再把这些请求转发给 Web Services 请求处理器。请求处理器的作用在于，解析收到的 SOAP 请求，调用 Web Services，然后再生成相应的 SOAP 应答。Web 服务器得到 SOAP 应答后，会再通过 HTTP 应答的方式把信息送回到客户端。

3．Web Services 体系

Web Services 体系主要包括以下几个方面：

（1）Web Services 包括 3 种组件。

服务提供者：提供服务，进行注册以使服务可用；

服务代理者：服务交换所，服务提供者和服务请求者之间的媒体；

服务请求者：向服务代理请求服务，调用这些服务创建应用程序。

（2）Web Services 提供 3 种操作。

发布/不发布（Publish/Unpublish）：服务提供者向服务代理者发布（注册）服务或不发布（移去）这些服务的注册；

发现（Find）：由服务请求者向服务代理者执行发现操作，服务请求者描述要找的服务，服务代理者分发匹配的结果；

绑定（Bind）：在服务请求者和服务提供者之间绑定，这两部分协商以使请求者可以访问和调用提供者的服务。

（3）UDDI 规范

统一描述、发现和集成（Universal Description Discovery and Integration, UDDI）是一个 Web Services 的信息注册规范，基于 UDDI 的 Web Services 注册可以被发现。UDDI 的核心部分是 UDDI 业务登记逻辑，即在 Web 上有一种分布的注册服务，这种服务以一种通用的 XML 格式进行描 述。通过 XML 中的结构化描述，可以很方便地在互联网上发现需要的数据，进而方便进行分析和操作。从概念上看，一个 UDDI 业务登记逻辑所提供的信息包括 三个部分：“白页”包括地址、协议和已有标识；“黄页”包括基于分类标准的工业类型；“绿页”是关于企业所包含的服务技术信息，包括网络服务说明参考和根据发现机制对各种文件和网址提供的标识支持。

（4）网络服务描述语言（WSDL）

网络服务描述语言（Web Services Description Language, WSDL）遵循 XML 语法，为服务提供者提供了描述构建在不同协议或编码方式之上的 Web Services 请求基本格式的方法。WSDL 用来描述一个 Web Services 能做什么，它的位置在哪里，如何调用它等。在假定以 SOAP/HTTP/MIME 作为远程对象调用机制的情况下，WSDL 会发挥最大作用。UDDI 注册描述了 Web Services 绝大多数方面，包括服务的绑定细节。WSDL 可以看作是 UDDI 服务描述的子集。

WSDL 将服务定义为一个网络端点的集合，或者说端口的集合。在 WSDL 里面，端点及消息的抽象定义与它们具体的网络实现和数据格式绑定是分离的。这样就可以重用这些抽象定义：消息（需要交换的数据的抽象描述）和端口类型（操作的抽象集合）。针对一个特定端口类型的具体协议和数据格式规范构成一个可重用的绑定。一个端口定义成网络地址和可重用的绑定的连接，端口的集合定义为服务。因此，一个 WSDL 文档在定义 Web Services 时使用如下的元素：

类型——使用某种类型系统定义数据类型的容器；

消息——通信数据抽象的有类型的定义；

操作——服务支持动作的抽象描述；

端口类型——操作的抽象集合，该操作由一个或多个端点支持；

绑定——针对一个特定端口类型的具体协议规范和数据格式规范；

端口——单一的端点，定义成一个绑定和一个网络地址的链接；

服务——相关端点的集合。

不难看出，WSDL 给客户提供了一个模板，方便客户描述和绑定服务。

上面简单介绍了 Web Services 基本的知识，下面采用 Web Services 单协议进行简要的脚本录制，读者可结合录制脚本的过程进一步了解它，具体步骤如下：

选择“开始”>“程序”>“LoadRunner”>“Virtual User Generator”（Vuser 生成器），启动 VuGen。在 VuGen 的“File”下拉菜单中选择“New”，新建一个脚本；从 “Category”（类别）列表中选择“Web Services”协议，单击“OK”按钮开始录制 Vuser 脚本。

首先配置 Web Services 录制向导，配置程序录制的方式。“Record Client Application”方式是通过客户端进行录制的，“Scan WSDL file”方式需要录入 WSDL 文件才可录制，在这里选择“Record Client Application”进行录制，如图 2-12 所示。

“Specify WSDL files for recording”向导用于配置 WDSL 文件，由于选择“Record Client Application”的录制方式，所以在此选择“Do not use WSDL file during recording”，表示不利用 WSDL 文件进行录制，如图 2-13 所示。

图 2-12　Web Services 录制界面 1

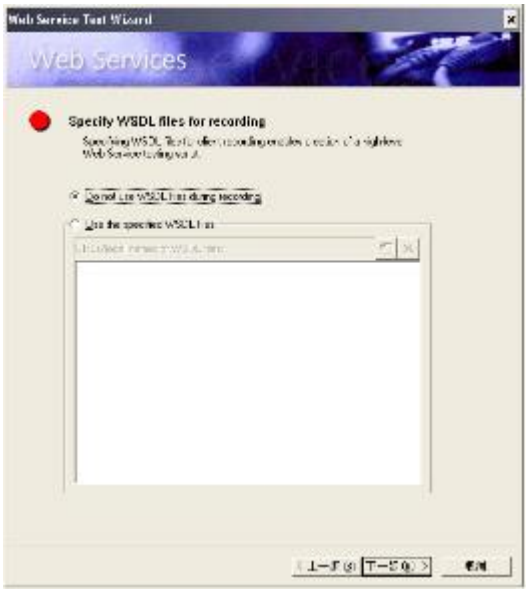


图 2-13　Web Services 录制界面 2

“Specify application to record”向导用于配置程序的访问地址、浏览器和录制脚本中的一些初始化设置。在 URL 中添加测试程序的访问地址；如果程序需要其他的浏览器，选择“Record any application”进行其他浏览器的设置，这里不需要特殊的浏览器，所以不选择此项；“Record into action”选项用于指定把录制的脚本放到哪一个部分，一般初始化放在 vuser_init 中，循环部分在 Action 中，结束退出部分放在 vuser_end 中，如图 2-14 所示。如有需要请单击后面的“Advanced Details”按钮，可以详细地配置“Options Recording”用来录制脚本，这里不介绍 Options Recording，在以后的章节中有详细的介绍。单击“完成”按钮即可完成 Web Services 的向导配置。

然后 VuGen 将根据程序的访问地址自动启动应用程序，并显示“Recording…”（录制）工具栏，开始脚本的录制，如图 2-15 所示。

图 2-14 Web Services 录制界面 3

图 2-15 “Recording…”工具栏

在整个操作过程完成后，单击“停止”按钮，脚本录制结束，LoadRunner 自动把录制的内容保存在脚本中。在录制完毕的脚本中会出现一些函数，在后面章节中会详细介绍这些函数的使用方法。

一个生成的 Web Services 的脚本节选如图 2-16 所示。

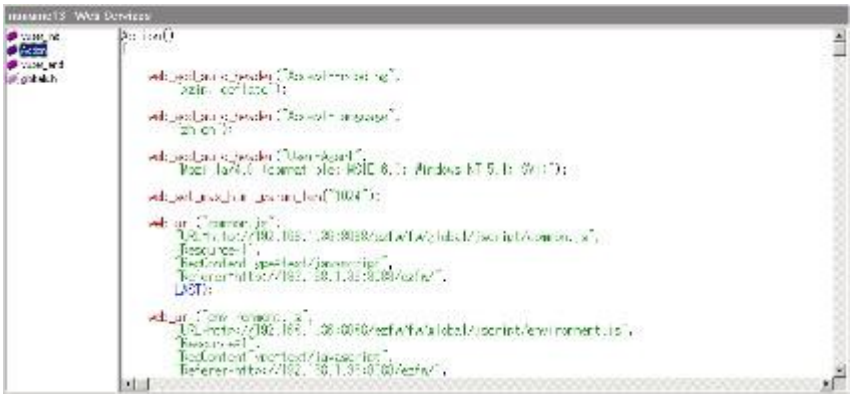


图 2-16　Web Services 脚本图例

这样就完成了 Web Services 单协议脚本的录制过程。

2.1.3. 回放脚本及调试

录制完脚本后，需要单机运行一下脚本，因为在录制脚本的过程中可能会出现错误。例如：有些连接、图片或界面无法找到，需要调试；有些地方需要参数化，只有唯一值才能执行通过；还有可能回放脚本时出现-404、-500 等错误页面，发生超时等现象。这时就需要把这些问题解决掉。

单击工具栏中的“Compile”按钮，查看脚本中是否有语法或者乱码错误，如果出现错误需要手工及时调试，如果没有错误，在执行日志中显示“No error detected”消息提示。

然后，单击工具栏中的“Run”按钮，开始执行脚本，在执行脚本期间，同样可以通过日志来查看发出的一些消息。选择“View”>“Output Window”，再选择“Execution Log”选项卡。

如果有错误，VuGen 将会提示错误。双击错误提示，VuGen 能够定位到出现错误的那一行，如图 2-17 所示。

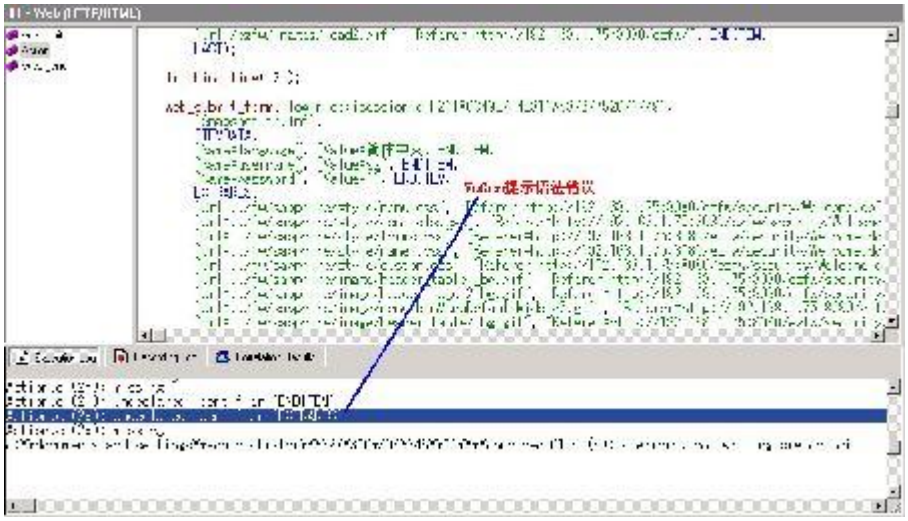


图 2-17　提示运行脚本错误

单机运行测试脚本后，如果编译通过，就会开始运行，运行结果如图 2-18 所示。

在每次单击回放脚本后，都会出现如图 2-18 所示的运行结果页。在结果页中可以清楚地看到脚本运行的情况，显示整个运行过程中出现成功、失败和警告情况各自的运行时间，并且记录下整个运行开始、结束的日期和时间。

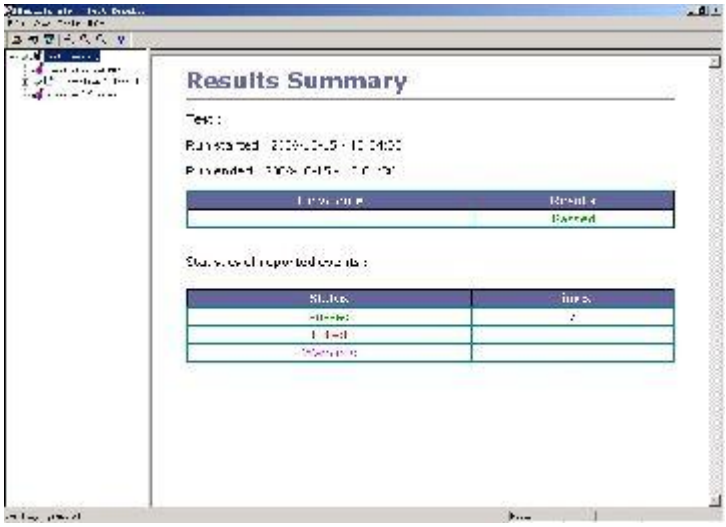


图 2-18　单机运行脚本结果

如果整个运行过程成功，在页面的左侧是整个脚本的树型结构，显示出的每个脚本的控件名称前都有绿色对号的标志，例如图片、链接、提交表单等，如图 2-19 所示。

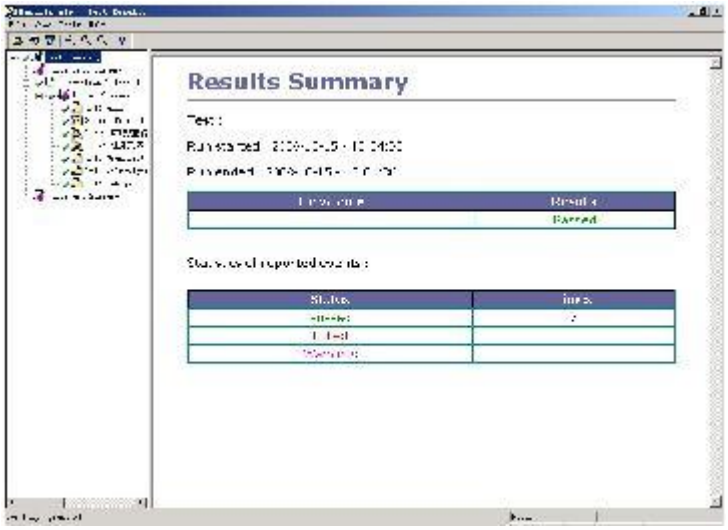


图 2-19　运行成功时的结果页

单击某个控件，在其右边便显示出其控件的页面或相应的运行步骤，如图 2-20 所示。

图 2-20　显示运行成功步骤

在此结果页中还可以检测脚本中控件或者其他错误，如果脚本回放出现错误的话，会在相应控件前出现红色叉号的错误提示，如图 2-21 所示。

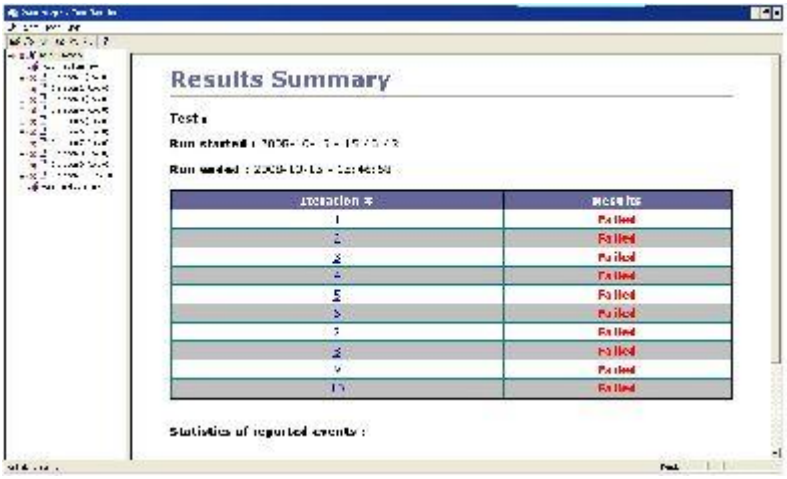


图 2-21　运行失败的结果页

单击其控件后，在右边出现脚本未通过的具体原因，以便查找出错位置进行改正，如图 2-22 所示。

图 2-22　定位运行失败

脚本录制、调试完成后，还可以通过插入事务、集合点等操作来完善、增强脚本。

3. 走近 LoadRunner

3.1 LoadRunner 的运行原理

3.1.1 LoadRunner 三大高手

3.1.2 三大高手联手的一场性能测试盛大演出

4. LoadRunner 脚本语言基础

5. VU——用户行为的模拟器

LoadRunner 之所以强大，很大的原因是 VU 的功能强大。作为虚拟用户的产生器，从横向上看，VU 几乎支持模拟当今所有主流的软件客户端，同时还在不断地推陈出新；从纵向上看，每个 Vuser 脚本的设置也是非常繁多和详细的，达到了精确模拟的效果。

因此，了解并熟悉 VU 是我们“玩转”LoadRunner 要做的第一件事情。

我们最常听到的关于 VU 的描述就是：VU 通过运行 VU 脚本模拟了用户对软件的操作行为。

如果我们不刨根问底，探究实质，就难以发现上面这句话的奥妙。

5.1. 序：图灵试验与 LoadRunner VU 模拟奥秘

5.1.1. 图灵试验场景

伟大的计算机之父阿兰•图灵曾对人工智能设计过一个著名的“图灵试验”，来判断计算机对人类的模拟能力，图灵试验场景如图 5-1 所示。

图灵试验由计算机、被测试的人和主持试验的人组成。计算机、被测试的人和测试主持人分别在三个不同的房间内。测试过程是由主持人提出问题，由计算机和被测试人分别回答。被测试人回答问题是尽可能地表明他是“真正的”人，计算机也尽可能逼真地模仿人的思维方式和思维过程。如果试验主持人听取对问题的回答后，分不清哪个是人回答的，哪个是计算机回答的，则可以认为被测试计算机是有人的智能的。

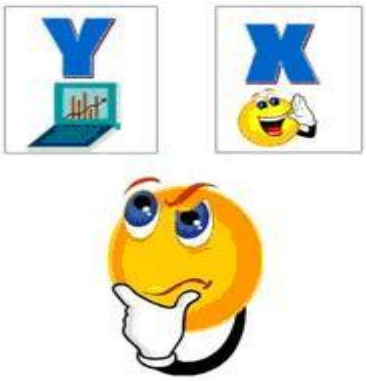


图 5-1　图灵试验场景

我们从这个试验中可以看到模拟需要有三个要素：模拟者（计算机）、被模拟者（真实的人）和观察者（主持人）。其中观察者是很重要的一个角色，模拟只有对观察者才有意义。而在图灵试验中，观察者通过一定的方式（闻其“答案”不见其人）来判断这个模拟是否成功。

5.1.2. LoadRunner 模拟揭秘

再回头看看那句话：“VU 通过运行 VU 脚本模拟了用户对软件的操作行为。”与之对应，我们能找到 VU 是模拟者，用户是被模拟者，但这句话里没有指出观察者。那么观察者是谁？我们性能测试工程师是观察者么？显然不对，用户的操作行为是一次次键盘输入，一个个鼠标点击，而 VU 在我们眼里只是一行行脚本程序和配置文件。

想到这里，可能已经有人得到了答案。没错，观察者其实就是性能测试中被测服务器。

我们发现，完全可以把图灵试验应用到我们的 VU 模拟场景中：

在软件系统运行时，一个真实的用户通过操作软件客户端来发起和服务器的会话请求，同时，一个 VU 开始运行它的脚本，也发起了同样的请求。被测服务器在处理各个“交谈”请求并与之分别会话的时候，并不知道哪些请求是来自真实的用户，哪些是来自 VU。这样，VU 成功地模拟了真实的用户，“骗过”了被测服务器。当然，VU 要实现这个目标，之前还需要通过认证，构造合法的请求等步骤，这些 VU 都要加以实现。

我们弄明白了观察者是被测服务器这件事情后，有些知识点就清楚多了。

事实：VU 是基于网络协议的。

很明显，被测服务器是通过各种各样的网络协议与客户端打交道的。VU 要“骗过”被测服务器，当然就要遵守这些协议，按规矩、按步骤来动作，否则就会吃“闭门羹”，毫不留情地被服务器拒绝。实际上，如果我们留心的话，可以发现 VU 每个协议相关的函数都是紧紧围绕其网络协议的实现的。同样，VU 录制生成的也是网络协议层次的脚本。在回放时，VU 将按照录制下来的网络事件，一一重放。

基于网络协议的脚本的一个好处是，我们可以使用相对少的硬件资源，来生成大量的虚拟用户负载。相比之下，WinRunner 和 QTP 的脚本是基于界面事件（GUI）的，它在一台主机上同时只能运行一个虚拟用户的脚本，因为一个虚拟用户会占用整个主机的资源。

推论一：VU 不关心用户在界面上发生的事情。

真实的用户对软件客户端可能有各种各样的操作，有些会触发对服务器的请求，有些则

不会。而 VU 的录制和运行都是基于网络上的事件的，因此 VU 在 录制的时候，只对那些网络事件感兴趣，如果有网络交互发生，就会记录成脚本，没有则只会生成一个空脚本，尽管可能用户在界面上做了很多操作，比如鼠标移动，填写 Web form 里的数据。

推论二：VU 中的操作关联与界面上的操作关联是不一致的

在我们对软件系统进行测试的过程中，某些操作是相关联的。例如，我们测试“查看客户交易历史明细”这个事务的系统响应时间，按界面测试思路，我们会这么做：

首先用客户查询系统，查询出此客户。

点击这个客户，打开另外一个页面，展现此客户的基本信息。

点击基本信息中的“交易历史明细”，系统经过查询后，刷新此页面，显示明细结果。

显然，在 UI 测试中，我们要测试第三步骤，之前必须要完成第一步和第二步。但在 VU 中就不是这样了：

这三个操作被 VU 记录成三个函数。如果它们是上下文无关的函数（关于上下文相关和上下文无关函数，可参看 LoadRunner 函数手册），那么其实，我们完全可以忽略掉第一和第二个函数，只执行和参数化第三个函数。

案例

疑问：使用 LoadRunner 录制一个 Java applet 技术的 Web 系统，要录制的动作是：

点击一个按钮后，此时系统调用 applet 小应用程序。

applet 小应用程序在本地加载完后，弹出一个通过 XML 文件配置对话框，在这个页面中建立节点树。

无论 LoadRunner 采用 Web 协议、Web Service 协议还是 Windows Socket 协议，在脚本录制并回放后，到系统中一检查，发现节点并没有创建，这是怎么回事？

解释：

根据推论一，我们知道 VU 只捕捉网络上的事件，在本案例中 LoadRunner 采用 Winsocket 协议也无法创建节点，这说明 XML 树节点 的创建是由 Java applet 完成的，并且很可能只保持在客户端本地，并未有和 Server 同步的迹象。因此，LoadRunner 脚本捕捉和回放的脚

本只是网络事件，并 不会对客户端本地产生影响，节点自然不会被创建。

根据推论二，如果我们的目标是测试服务器的性能，那么完全可以忽略这个问题，继续录制。

提示：基于 GUI 和基于协议的测试工具在实质上是看待同一事物的不同的角度。比如：用户点击一个 Web 页面上的“无忧测试“的链接文字，在 QTP 会被记录成 link("无忧测试").click 的函数，而在 LoadRunner 中则会被记录成 web_link 函数。前者是记录“鼠标在此链接上点击一次”这个事件，而后者是“客户端向 Web Server 发起了一个 get URL 地址为 www.51testing.com 的请求”。LoadRunner 和 QTP 如摸象的盲人一样，各自从 GUI 和网络协议的角度来看用户点击的 动作。而实际上它们所记录的只是用户操作整个过程的一部分。

提示：LoadRunner 8.1 从 SP2 开始，VU 协议族中新增了一个特殊的 Vuser 类型，叫做 Web Click and Script，为什么说它特殊呢？因为它和其他基于网络协议的 Vuser 不一样，Web Click 采用了 QTP 的技术，使得 VU 可以录制在界面上的一些操作。比如登录时填写用户名、口令这些动作都可以被记录成脚本。这在本书第 10 章中将会有 所介绍。

把 VU 的实质弄清楚了之后，我们就可以放心大胆地使用它了。Vugen 是个神奇的盒子，我们总能从盒子里拿到我们想要的东西。

5.2. 录制脚本

VU 通过录制用户在客户端软件的操作来直接生成脚本，用户的每个协议级的操作以 LoadRunner 的 API 函数方式记录在脚本里。回放脚本的时候，通过执行 API 函数来模拟最初用户的操作动作。

5.2.1. 选择协议

选择协议的两个基本原则已经在前文介绍过了。我们这里看看 LoadRunner 具体有哪些协议。

Vuser 类型可根据应用领域分为下列几种：

应用程序部署解决方案：Citrix ICA。

客户端/服务器：DB2 CLI、DNS、Informix、MS SQL Server、ODBC、Oracle（2 层）、Sybase Ctlib、Sybase Dblib 和 Windows Sockets 协议。

自定义：C 模板、Visual Basic 模板、Java 模板、JavaScript 和 VBScript 类型的脚本。

分布式组件：适用于 COM/DCOM、Corba-Java 和 Rmi-Java 协议。

电子商务：FTP、LDAP、Palm、PeopleSoft 8 mulit-lingual、SOAP、Web（HTTP/HTML）和双 Web/WinSocket 协议。

Enterprise Java Bean：EJB 测试和 Rmi-Java 协议。

ERP/CRM：Baan、Oracle NCA、PeopleSoft-Tuxedo、SAP-Web、SAPGUI、

Siebel-DB2 CLI、Siebel-MSSQL、Siebel-Web 和 Siebel-Oracle 协议。

传统：终端仿真（RTE）。

邮件服务：Internet 邮件访问协议（IMAP）、MS Exchange（MAPI）、POP3 和 SMTP。

中间件：Jacada 和 Tuxedo（6、7）协议。

流数据：Media Player（MMS）和 Real 协议。

无线：i-Mode、VoiceXML 和 WAP 协议。

当我们试图创建一个新的 Vuser 的时候，就会弹出协议选择对话框，如图 5-2 所示。

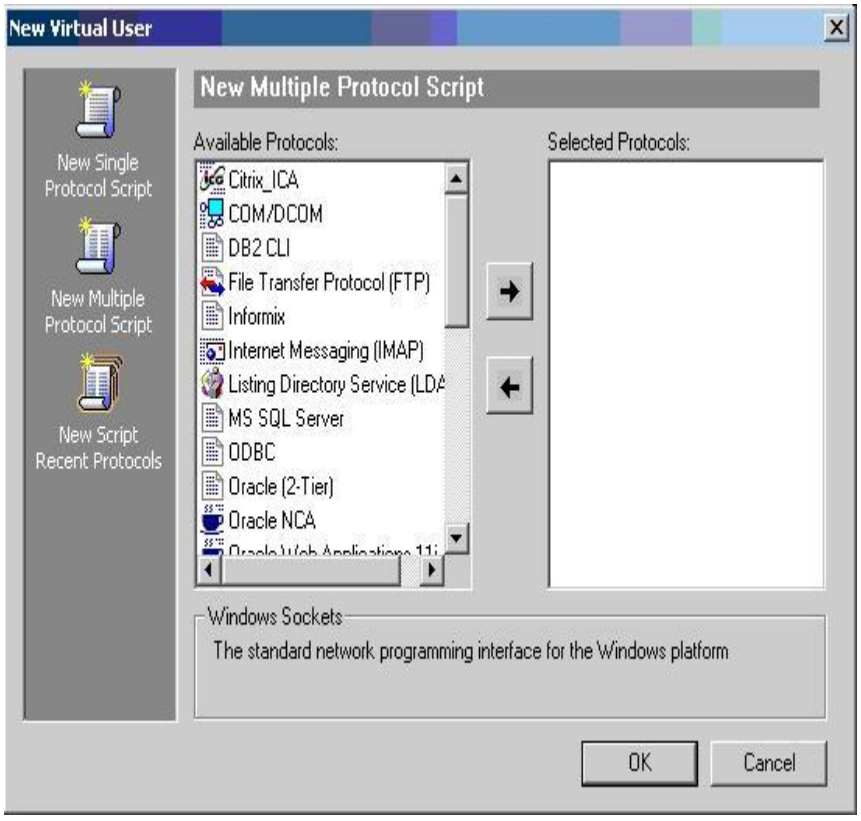


图 5-2　VU 协议选择对话框

有两种协议选择方式：单协议模式（New Single Protocol Script）和多协议模式（New Multiple Protocol Script）。

（1）单协议模式

当用户以单协议录制时，VU 只录制在既定协议上的用户操作，在 VU 中我们可以使用单协议模式选择任何一种协议。

（2）多协议模式

当用户以多协议录制时，VU 录制几个协议上的操作。并不是任意的协议都可组合成多协议模式。有以下协议支持多协议录制：COM、FTP、IMAP、Oracle NCA、POP3、Real Player、Windows Socket（raw）、SMTP 和 Web。双协议 Web/Win Socket 应该被看作是单协议，因为其机制与多协议还是不一样的。

不同类型的 Vuser 的另外一个区别是多 Action 的支持，一些协议支持多 Action，目前这些协议是：Oracle NCA、Web、RTE、general C、WAP、I-Mode 和 voice XML。关于多 Action 是何物，又如何使用，可以参看本章“多 Action”一节。

5.2.2. 规划脚本结构

在录制时，用户可以选择哪些操作生成脚本在 vuser_init、Action 和 vuser_end 中，同时，也可以在录制时随时加入 transaction 的定义、注释和同步点。VU 录制工具条如图 5-3 所示。

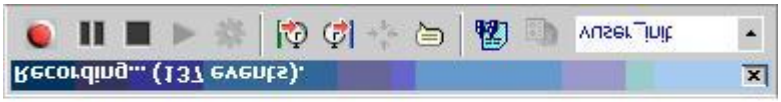


图 5-3 VU 录制工具条

5.2.3. HTTP Vuser 中的 URL mode 和 HTML mode

在录制之前，我们需要设置录制选项，如图 5-4 所示。

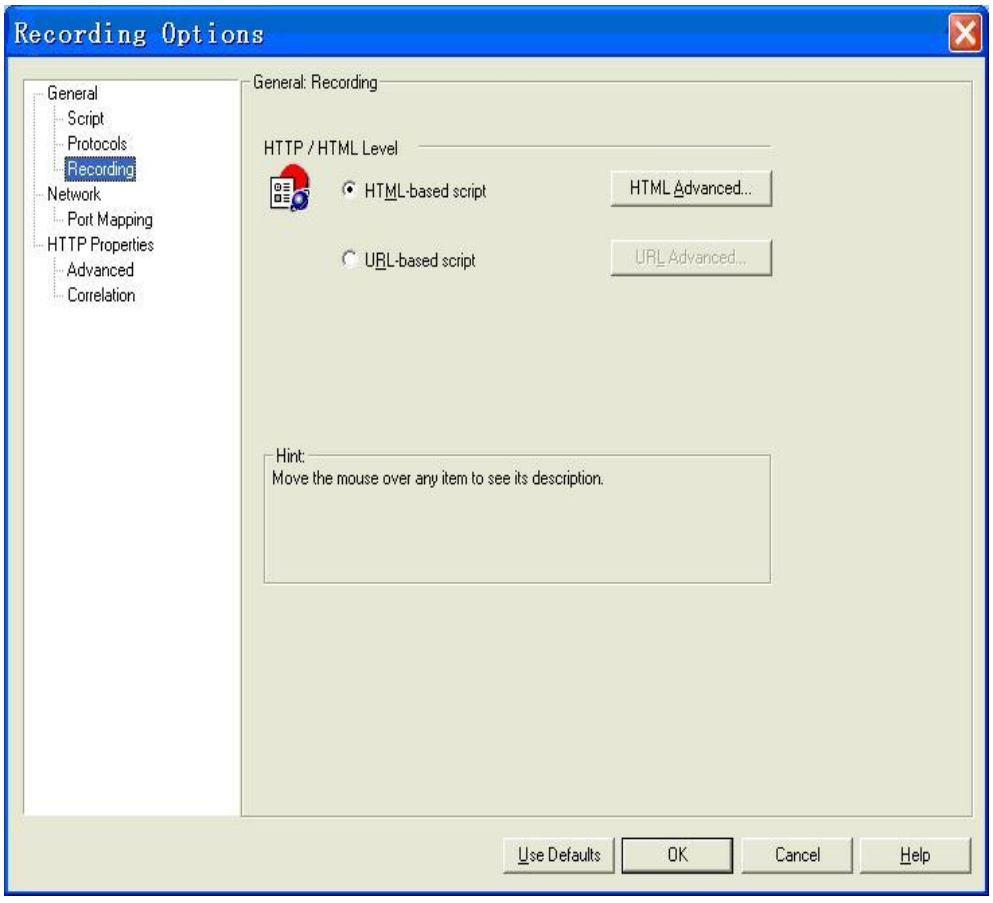


图 5-4 VU 录制设置选项

在默认情况下，选择“HTML-based script”，说明脚本中采用 HTML 页面的形式来表示，这种方式的 Script 脚本容易维护，容易理解，推荐以这种方式录制。“URL-based script”说明脚本中的表示采用基于 URL 的方式，所有的 HTTP 的请求都会被录制下来，单独生成函数，所以 URL 模式生成的脚本会显得有些杂乱。

实例

以 HTML 模式录制，访问“http://newtours.demoaut.com/”网站会生成下面的脚本：

```
Action()
{
  web_url("newtours.demoaut.com",
    "URL=http://newtours.demoaut.com/",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTML",
    LAST);
  return 0;
}
```

以 URL 模式录制同样的操作，会生成如下脚本：

```
Action()
{
  web_url("newtours.demoaut.com",
```

```
		"URL=http://newtours.demoaut.com/",
		"Resource=0",
		"RecContentType=text/html",
		"Referer=",
		"Snapshot=t1.inf",
		"Mode=HTTP",
		LAST);
	web_url("logo.gif",
		"URL=http://newtours.demoaut.com/images/nav/logo.gif",
		"Resource=1",
		"RecContentType=image/gif",
		"Referer=http://newtours.demoaut.com/",
		"Snapshot=t2.inf",
		LAST);
	web_url("html.gif",
		"URL=http://newtours.demoaut.com/images/nav/html.gif",
		"Resource=1",
		"RecContentType=image/gif",
		"Referer=http://newtours.demoaut.com/",
		"Snapshot=t3.inf",
		LAST);
```

```
web_url("boxad1.gif",
    "URL=http://newtours.demoaut.com/images/nav/boxad1.gif",
    "Resource=1",
    "RecContentType=image/gif",
    "Referer=http://newtours.demoaut.com/",
    "Snapshot=t4.inf",
    LAST);
.....
.....
//经统计，录制生成的 web_url 函数有 20 个
return 0;
}
```

是选择 HTML 还是 URL 录制，有以下参考原则：

（1）基于浏览器的应用程序推荐使用 HTML-based script。

（2）不是基于浏览器的应用程序推荐使用 URL-based script。

（3）如果基于浏览器的应用程序中包含了 JavaScript 并且该脚本向服务器产生了请求，比如 DataGrid 的分页按钮等，也要使用 URL-based script 方式录制。

（4）基于浏览器的应用程序中使用了 HTTPS 安全协议，使用 URL-based script 方式录制。

5.2.4. 查看日志

在录制和回放的时候，VU 会分别把发生的事件记录成日志文件，这些日志有利于我们跟踪 VU 和服务器的交互过程。我们可以通过 VU 输出窗口观察日志，也可以到脚本目录中

直接查看文件。其中有三个主要的日志对我们的录制很有用：

1．执行日志（Execution Log）

脚本运行时的输出都记在这个 Log 里。

“输出”窗口的“执行日志”显示的消息用于描述 Vuser 运行时执行的操作。该信息可说明在方案中执行脚本时，该脚本的运行方式。

脚本执行完成后，可以检查“执行日志”中的消息，以查看脚本在运行时是否发生错误。

“执行日志”中使用了不同颜色的文本。

黑色：标准输出消息。

红色：标准错误消息。

绿色：用引号括起来的文字字符串（例如 URL）。

蓝色：事务信息（开始、结束、状态和持续时间）。

如果双击以操作名开始的行，光标将会跳到生成的脚本中的相应步骤上。

图 5-5 显示了 Web Vuser 脚本运行时的“执行日志”消息。

执行日志是我们调试脚本时最有用的信息。有关设置执行日志级别调试脚本的技巧，在本章“高级——脚本调试技巧”一节中有详细介绍。

2．录制日志（Recording Log）

当录制脚本时，Vugen 会拦截 Client 端（浏览器）与 Server 端（服务器）之间的对话，并且通通记录下来，产生脚本。在 Vugen 的 Recording Log 中，我们可以找到浏览器与服务器之间所有的对话，包含通信内容、日期、时间、浏览器的请求、服务器的响应内容等，如图 5-6 所示。脚本和 Recording Log 最大的差别在于，脚本只记录了 Client 端要对 Server 端所说的话，而 Recording Log 则是完整记录二者的对话。因此通过录制日志，我们能够更加清楚地看到客户端与服务器的交互，这对我们开发和 debug 脚本非常有帮助。

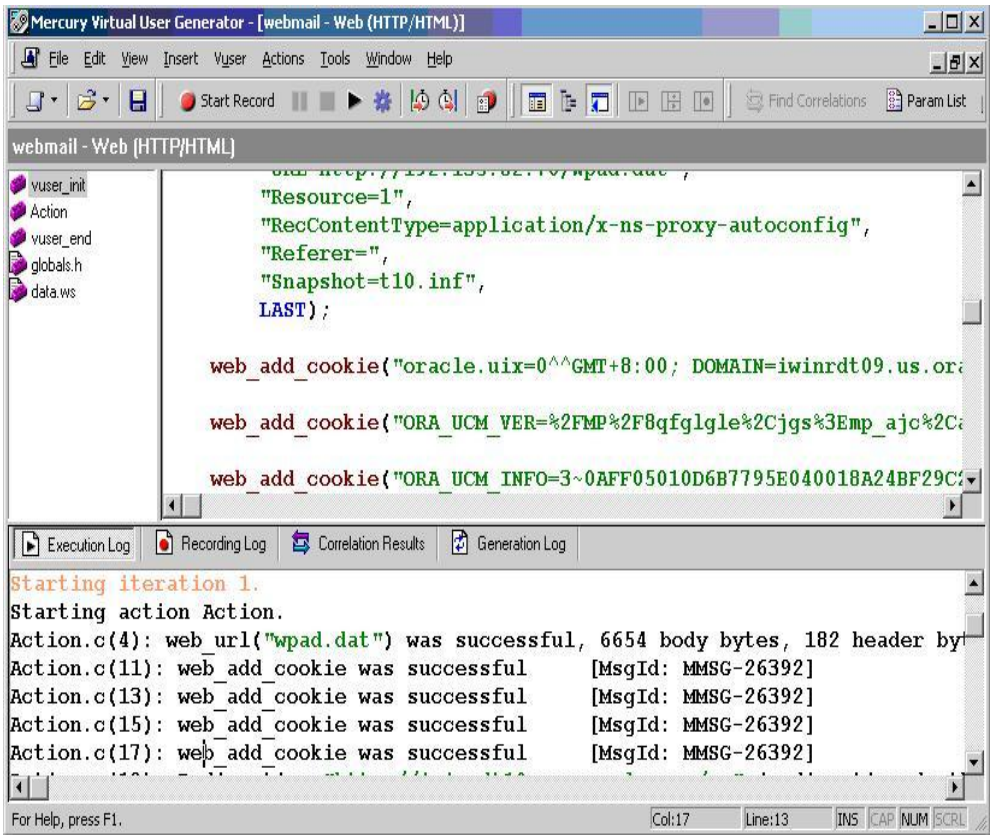


图 5-5 VU 脚本执行日志

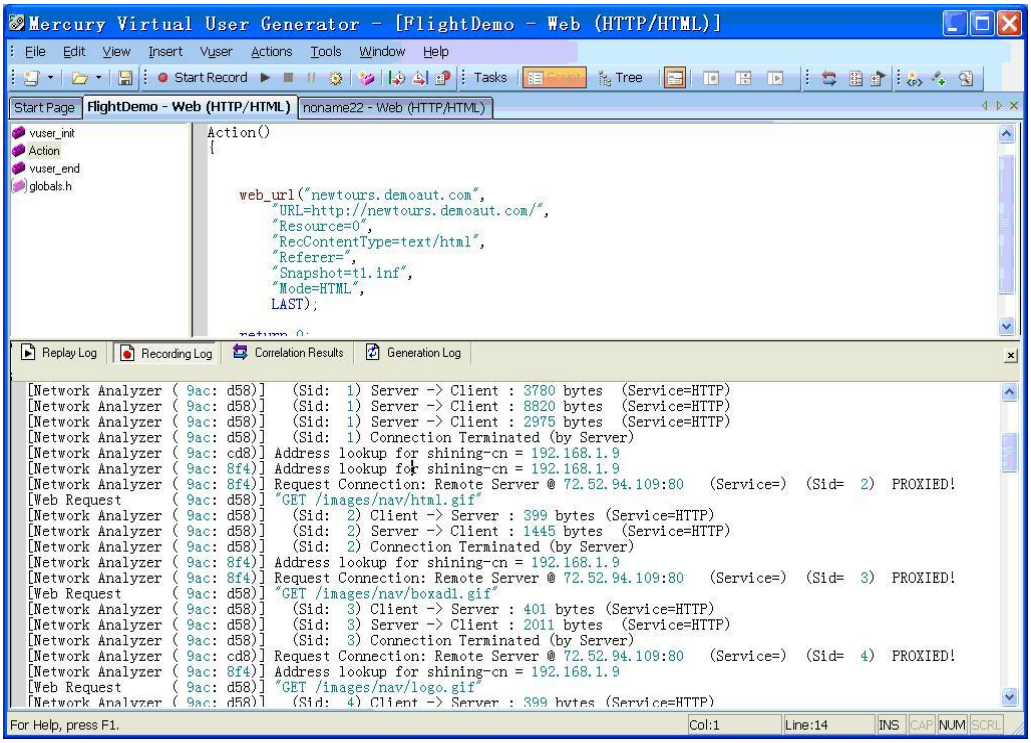


图 5-6 VU 脚本录制日志

3．产生日志（Generation Log）

产生日志记录了脚本录制的设置、网络事件到脚本函数的转化过程。

提示 ：这里同样需要注意的是：脚本能正常运行后应禁用日志。因为产生及写入日志需占用一定资源。

5.3. 回放脚本

单击 run 按钮，或按快捷键“F5”就可以运行脚本。VU 脚本运行工具条如图 5-7 所示。

在 LoadRunner 8.0 中提供了编译功能，快捷键为“Shift+F5”，这样脚本本身的语法检查等简单工作可以在编译中来做，省却了每次都要运行脚本的麻烦。

VU 提供运行时查看浏览器活动的功能。这对于我们调试脚本是非常有帮助的，可以方便我们直观地查看运行状态。这有些类似 QTP 和 WinRunner 的功能，需要修改设置来开启这个功能。

设置方法如下：在 VU 菜单“Tools”>“General Options”>“Display”选项卡中，勾选“Show browser during replay”项，如图 5-8 所示。

需要注意的是，运行时开启的 Run-Time Viewer 是 LoadRunner 自带的 HTML 解释器，并不是完全的 IE，因此有些 Java Script 和 Applet 可能会不能正常显示。VU 运行时查看器如图 5-9 所示。

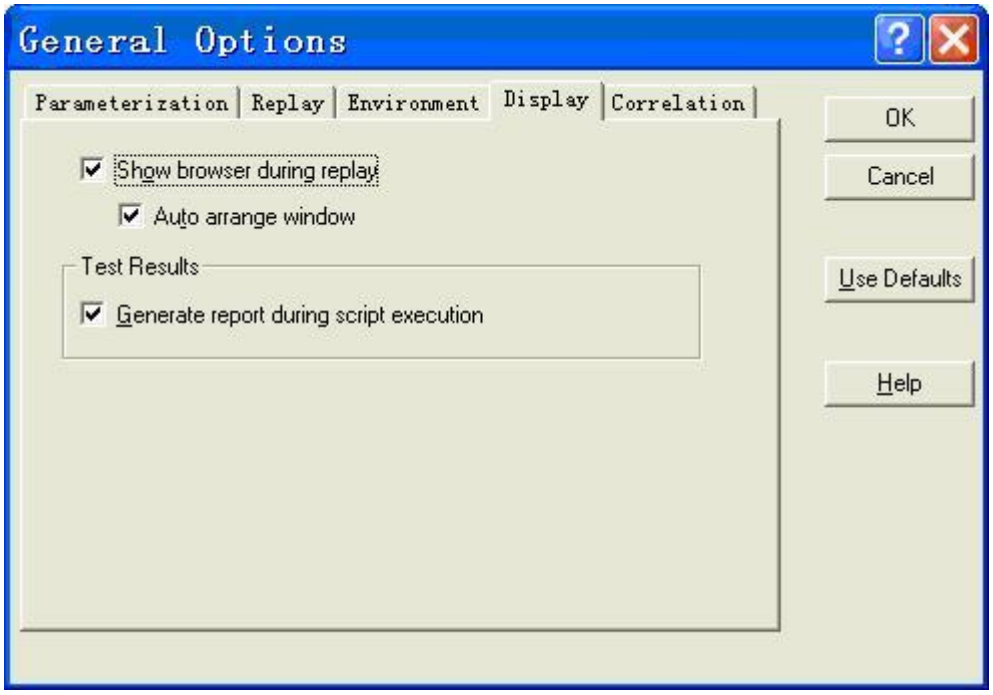


图 5-8 VU 开启运行时查看器

图 5-9 VU 运行时查看器

5.4. 关联

当刚刚录制好的脚本回放不能成功时，你首先想到第一个可能出现的问题就是关联。

关联是 LoadRunner 中一个重要的应用，也是初学者经常犯错误的地方。在前面我们已经大概介绍了关联的原理，在本节则学习 LoadRunnerr 怎么做关联。

如果脚本需要关联（Correlation），在还没做之前是不会执行通过的，也就是说，会有错误信息发生。不过，很遗憾，并没有任何特定的错误信息是和关联（Correlation）有关系的。会出现什么错误信息，与系统实际的错误处理机制有关。错误信息有可能会提醒您要重新登入，但是也有可 能直接就显示 HTTP 404 的错误信息。

这种问题在任何系统中都是非常常见的问题。在前面的章节我们已经了解了关联产生的背景和原理，其通用的解决模式是：“服务器返回给客户端一些动态变化的值，客户端使用这些值去访问服务器的时候，不能把这些值写死（hard-code）在脚本里面，而应该存放在一个变量里面。”这就是关联的概念。

关联的工作往往占据开发脚本的大部分时间，因为我们必须针对每一个具体的系统进行细致的分析，确定其需要关联的动态信息。而幸运的是，VU 为我们提供了三种关联机制。关联的方法有三种：

5.4.1. 录制前 Correlation（关联）

我们在前面章节的有关关联原理中谈到，服务器就像一个饭店，而客户端就像一个拿着小票领饭的食客。如果我们在进饭店之前，已经预先知道小票的样子，那么这时我们就可以启用录制前关联了。录制前关联的原理是，我们在录制前建立关联的规则（提前告诉 VU 小票是什么样子），录制时 VU 按照这些规则寻找并建立关联（获得小票）。因此在 VU 中，录制前关联又叫做规则关联（Rule Correlation）。可见，使用录制前关联的必要条件是我们必须在录制脚本之前就知道哪些变量是需要关联的。

Vugen 内建自动关联引擎（auto-correlation engine），可以自动找出需要关联的值，并且自动使用关联函数建立关联。

在录制过程中 Vugen 会根据制定的规则，实时自动找出要关联的值。

1．规则的建立

关联的规则主要是指定两个边界，被关联变量的左边界和右边界，当 VU 在 Server 的 Respsonse 中找到符合条件的字符串时，它就意识到这是一张“小票”，赶紧把它保存下来，作为参数。

规则来源有两种：

（1）内建关联规则（Built-in Correlation）

Vugen 已经针对常用的一些应用系统，如 AribaBuyer、BlueMartini、BroadVision、InterStage、mySAP、NetDynamics、Oracle、 PeopleSoft、Siebel、SilverJRunner 等内建关联规则，这些应用系统可能会有一种以上的关联规则。您可以在 “Recording Options”>“HTTP Properties”>“Correlation”中启用关联规则（见图 5-10），则当录制这些应用系统的脚本时，Vugen 会在脚本中自动建立关联。

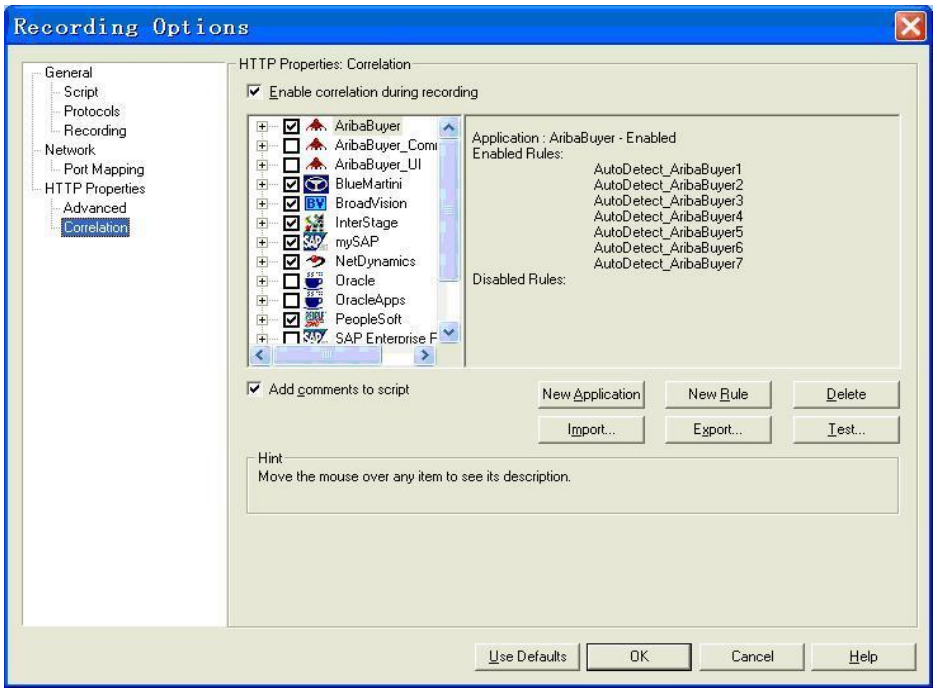


图 5-10 VU 内建关联规则

2）使用者自订关联规则（User-defined Rules Correlation）

除了内建的关联规则之外，使用者也可以自订关联规则。您可以在“Recording Options”>“HTTP Properties”>“Correlation”中建立新的关联规则（见图 5-11）。

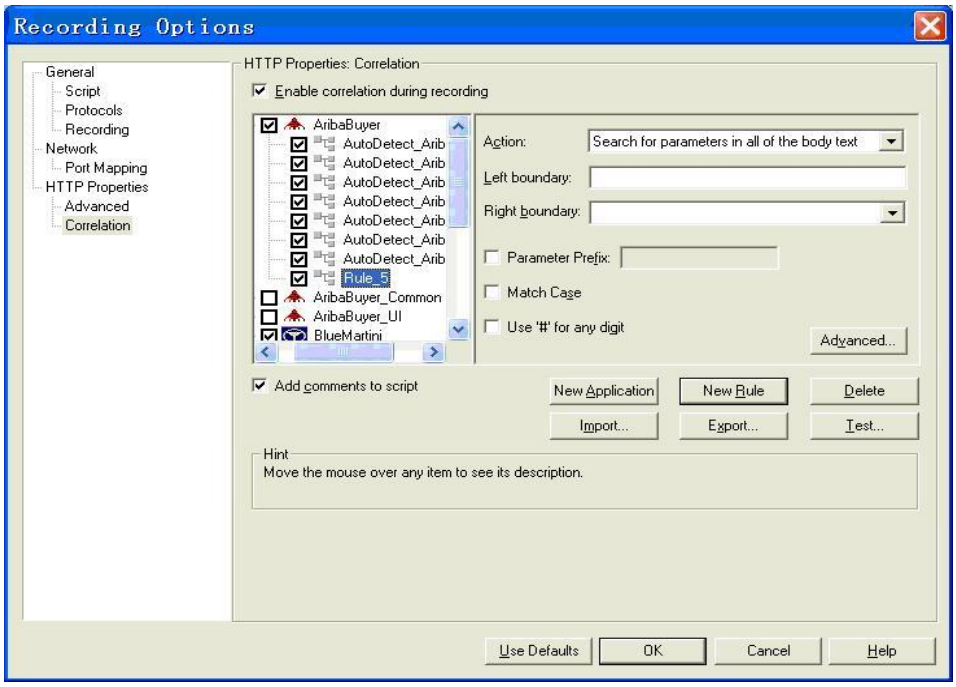


图 5-11　VU 创建一个新的关联规则

图 5-11 所示是创建新规则的页面，在创建新规则时，需要设置关联变量的左边界、右边界等信息。

2．规则的应用

依照以下步骤使用 Rule Correlation：

（1）启用自动关联（auto-correlation）

点选 Vugen 的“Tools”>“Recording Options”，打开“Recording Options”对话框，选取“HTTP Properties”>“Correlation”，勾选“Enable correlation during recording”，以启用自动关联。

假如录制的应用系统属于内建关联规则的系统，如 AribaBuyer、BlueMartini、BroadVision、InterStage、mySAP、NetDynamics、Oracle、PeopleSoft、Siebel、SilverJRunner 等，请勾选相对应的应用系统；或者也可以针对录制的应用系统加入新的关联规则，此即为使用者自订的关联规则。

（2）运行脚本验证关联机制

当关联规则都运行开始录制脚本时，在录制过程中，当 Vugen 侦测到符合关联规则的数据时，会依照设定建立关联，您会在脚本中看到类似以下的脚本，如图 5-12 所示。

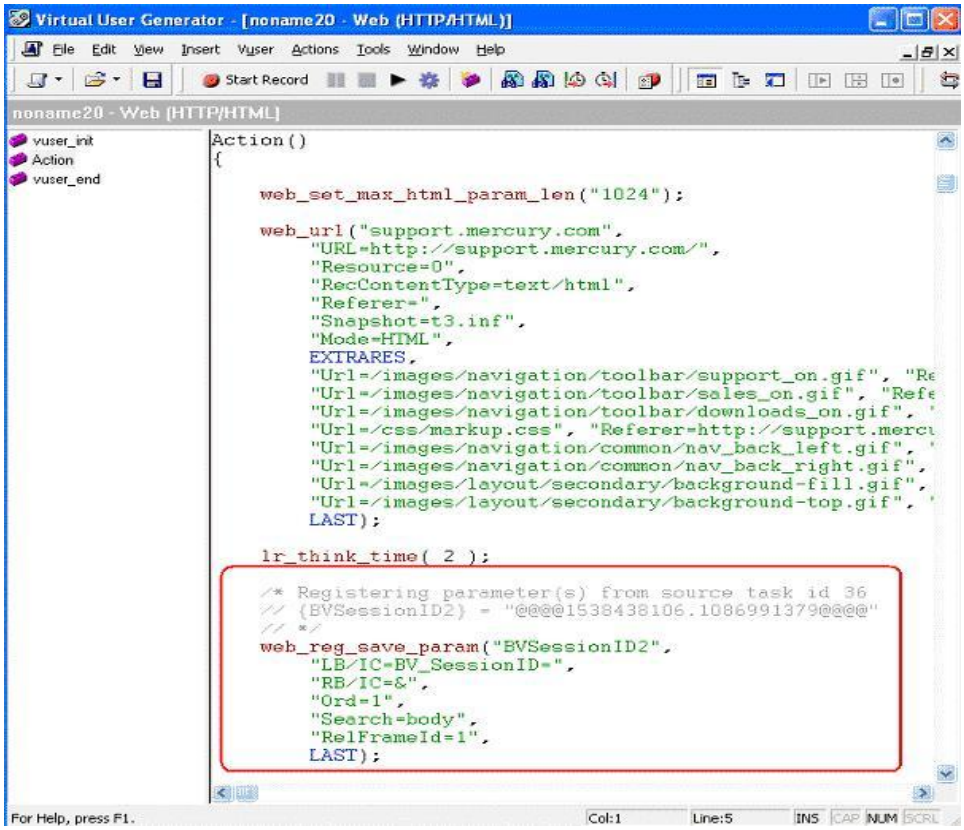


图 5-12　先建立关联规则，然后录制脚

执行链接的脚本是 OK 的。

5.4.2. 录制后关联

当录制的应用系统不属于 Vugen 预设支持的应用系统时，也就是 VU 在录制之前并不知道服务器小票的样子，这时规则关联（Rule Correlation）就无能为力了，但不要紧，VU 还提供了一招，叫做录制后关联。

录制后关联：有别于内建关联，录制后关联则是在执行脚本后才会建立关联，也就是说，当录制完脚本后，脚本至少需被执行过一次，录制后关联才会作用。录 制后关联会尝试找出录制时与执行时，服务器响应内容的差异部分，藉以找出需要关联的数据，并建立关联。

拿食客问题来解释录制后关联，就是食客第一次去某饭店吃饭时，他并不知道这个饭店的小票是什么版式，也不知道饭店给他一张写着“97”号的纸条是什么 意思，不过不要紧，一回生，二回熟，第二次他去饭店，发现饭店又给他一张“108“号的纸条，他这时就能推断出，每次他去饭店这个号码都会变的，这应该就是饭店的小票。

因此使用录制后关联必须要让脚本运行第二次，步骤如下：

录制脚本并回放，回放完毕后，Vugen 会跳出下面的“Scan Action for Correlation”窗口，询问您是否要扫描脚本并建立关联，单击“Yes”按钮，如图 5-13 所示。

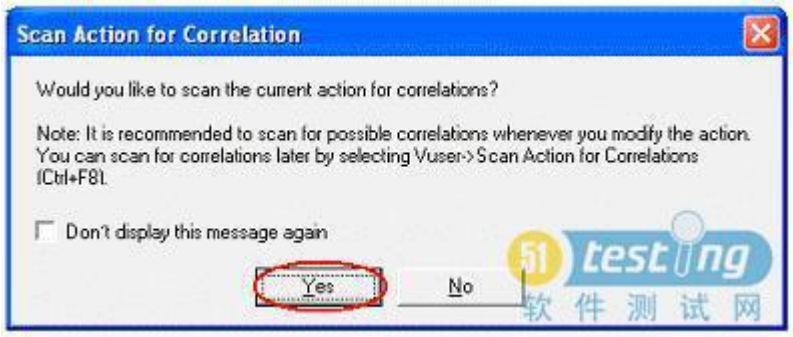


图 5-13　录制后关联提示页面

扫描完后，可以在脚本下方的“Correlation Results”中看到扫描的结果，如图 5-14 所示。

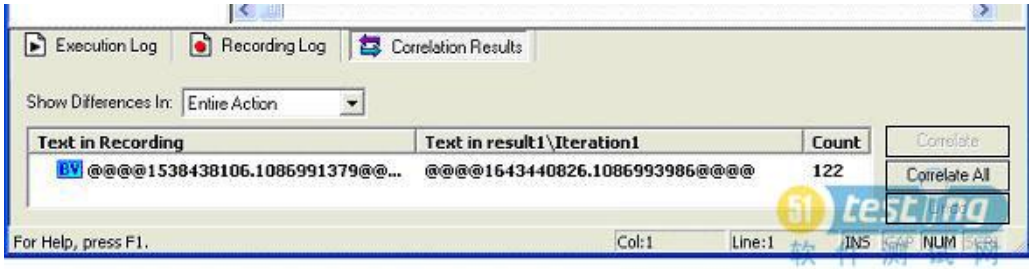


图 5-14　VU 的关联结果信息

注意：检查一下扫描的结果，选择要做关联的数据，然后单击“Correlate”按钮，一笔一笔做，或者单击“Correlate All”按钮，让 Vugen 一次就对所有的数据建立关联。

由于录制后关联会找出所有有变动的数据，但是并不是所有的数据都需要做关联，所以不建议您直接用“Correlate All”。

要手动启动“Scan Action for Correlation”功能，请先执行脚本一次后，点选“Vuser”>“Scan Action for Correlations”，如图 5-15 所示。

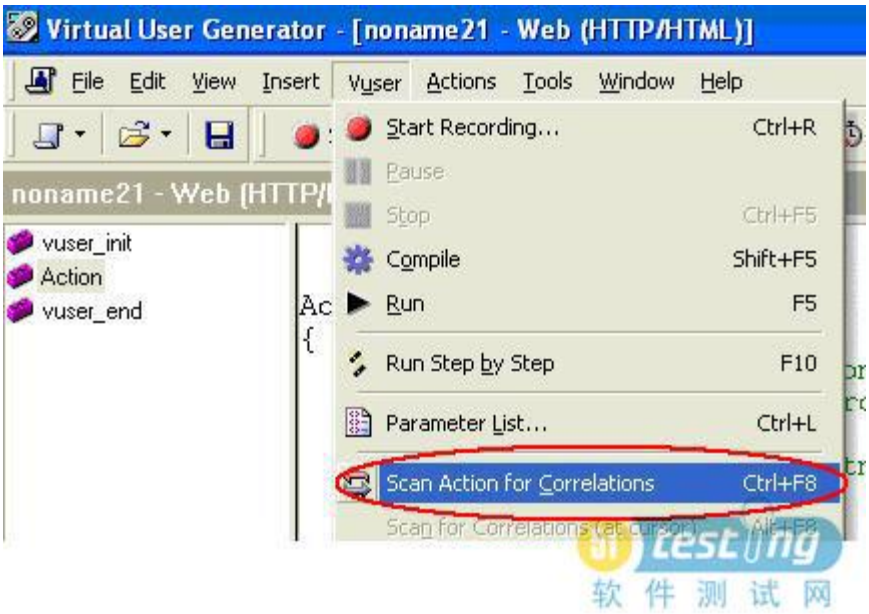
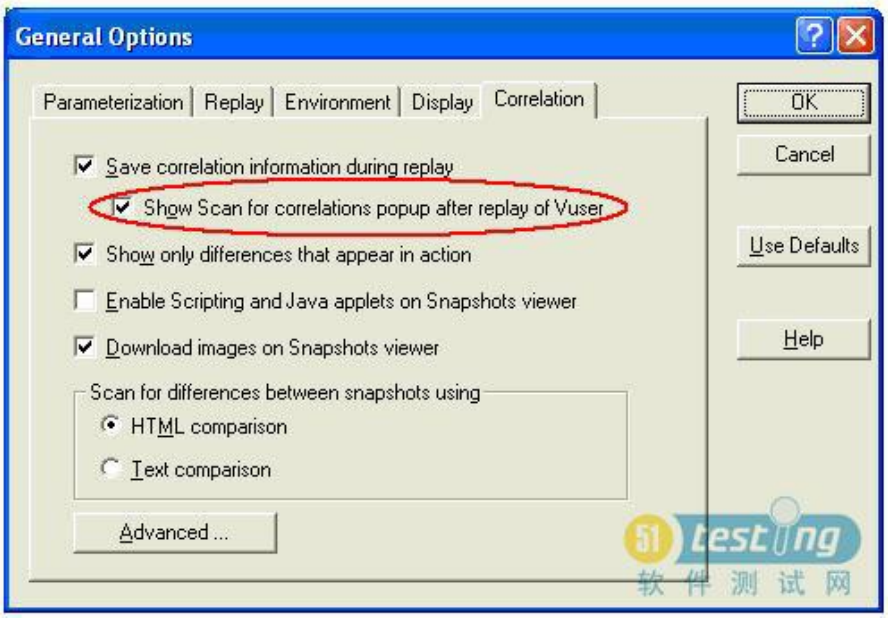


图 5-15　VU 选择运行扫描关联

执行完脚本后并未出现“Scan Action for Correlation”窗口，要启用“Scan Action for Correlation”功能，请点选“Tools”>“General Options”>“Correlation”选项卡，勾选“Show Scan for correlation popup after replay of Vuser”选项，如图 5-16 所示。



5.5. 脚本视图和树视图

VU 提供两种视图来查看脚本的内容，一个是脚本视图，另一个是基于图标的树视图（内

有快照）。

所有类型的 Vuser 都有文本脚本视图，但是只有特定的 Vuser 才会有树视图。

5.5.1. 树视图（Tree View）

Tree View 也叫做基于 icon 的 View，也就是说，脚本的每个函数在 Tree View 中都以一个带有 icon 的节点来代替。可以点击工具栏中的“Tree”按钮或者在“View”菜单下选择“Tree View”，显示 VU 树视图，如图 5-20 所示。

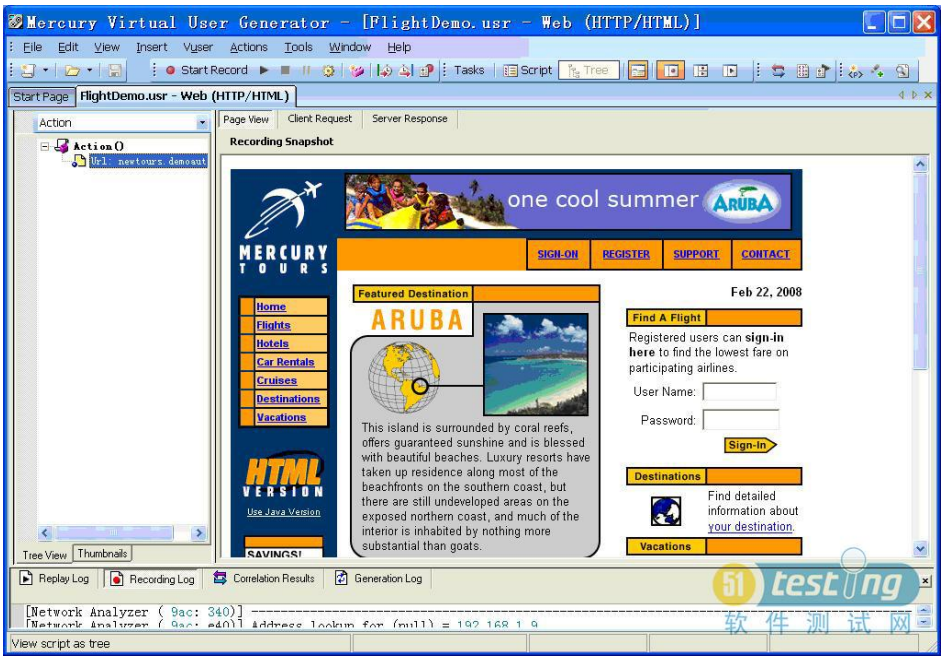


图 5-20　VU 树视图

Tree View 的好处是使用户更方便地修改脚本，Tree View 支持拖拽，用户可以把任意一个节点拖拽到他想要的地方，从而达到修改脚本的目的。用户可以右键单击节点，进行修改/删除当前函数参数属性，增加函 数等操作，通过 Tree View 能够增加 LoadRunner 提供的部分常用通用函数和协议相关函数。比如 Web Service Vuser 就不能通过 Tree View 参数化一些复杂的数据类型，在这种情况下，就需要 Script View 了。

5.5.2. 脚本视图（Script View）

在 Script View 中能够看到一行行的 API 函数，Script View 适合一些高级用户，通过 Script View 向脚本中增加一些其他 API 函数。可以单击工具栏上的“Script”按钮或者在“View”菜单下选择“Script View”，显示 VU 脚本视图，如图 5-21 所示。

注意：当用户在 Script View 中对脚本做了修改之后，Tree View 也会做相应的变化。如果脚本有语法错误，Script View 将不能转化为 Tree View 或缩略图。

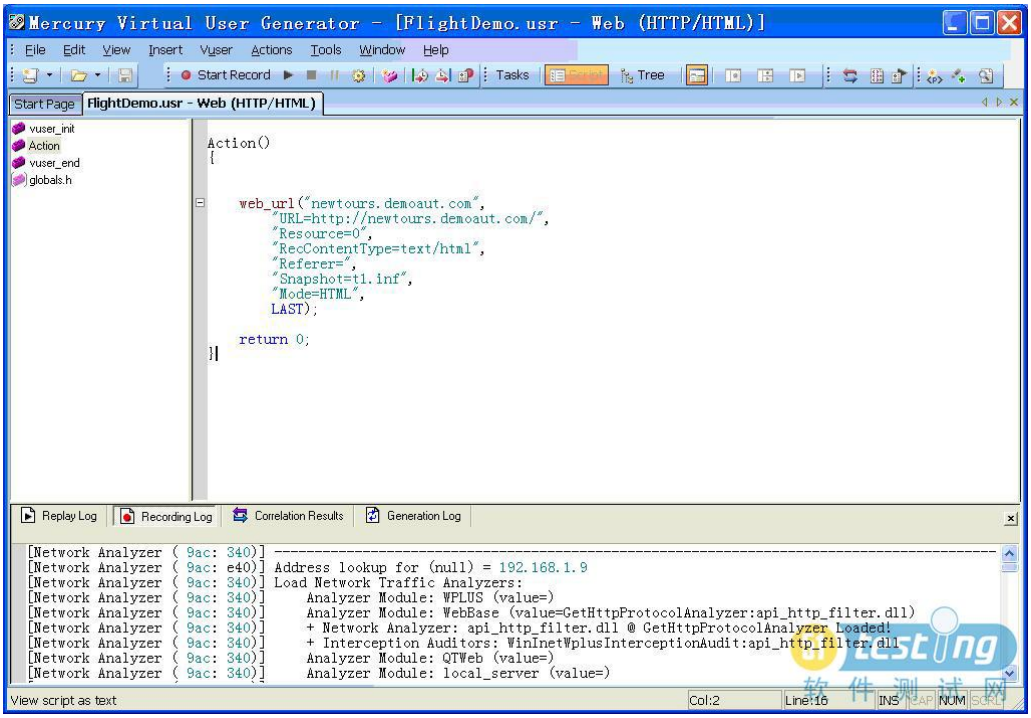


图 5-21　VU 脚本视图

5.5.3. 理解 Snapshot

Snapshot，顾名思义，就是快照，代表当前的 step，Snapshot 显示了客户端在执行完当前 step 后的样子。在 Tree Ciew 右侧的 frame 中可以查看 Snapshot，在 LoadRunner 8.0 中，Snapshot 包含 Page View、Client Request 和 Server Response。Snapshot 有两种生成方式，一种是在 record 的时候生成，另一种是在 replay 的时候生成。你可以对比两种方式生成的 Snapshot，以发现哪些是动态值，需要参数化。

5.6. 事务、同步点和思考时间

5.6.1. Transaction（事务）

事务是计算机程序设计中一个很重要的概念。一个事务应该具有原子性、一致性、隔离性和持久性。这 4 个属性的详细解释在网上都可以找到。在 LoadRunner 里，我们定义事务主要是为了度量服务器的性能。每个事务度量服务器响应指定的 Vuser 请求所用的时间，这

些请求可以是简单任务（例如等待对单个查询的响应），也可以是复杂任务（例如提交多个查询和生成报告）。

要度量事务，需要插入 Vuser 函数以标记任务的开始和结束。在脚本内，可以标记的事务不受数量限制，每个事务的名称都不同。

在场景执行期间，Controller 将度量执行每个事务所用的时间。场景运行后，可使用 LoadRunner 的图和报告来分析各个事务的服务器性能。

设置 Transaction 的方法如下：

选择新 Transaction 开始点，在被度量脚本段之前插入 lr_start_transaction。

选择新 Transaction 结束点，在被度量脚本段之后插入 lr_end_transaction。

下面的脚本例子中将登录操作设为一个名为“login”的事务：

```
Lr_start_transaction(“login”);

web_submit_form(”auth”,

“Snapshot=t2.inf”,

ITEMDATA,

“Name=ssousername”,

“Value=robin”, ENDITEM,

“Name=password”, “Value=123456″, ENDITEM,

“Name=remember”, “Value=<OFF>”, ENDITEM,

LAST);

Lr_end_transaction(”login”);
```

如果上面手工插入 Transaction 函数看作是“显式事务”的话，那么 LoadRunner 还提供了一种“隐式事务”的机制，在 VU 的 Run-time Settings 中又称为“自动事务”。

在 Run-time Settings 中，在 Miscellaneous 选项卡的 Automatic Transactions 中定义自动事务。

可以设置 LoadRunner 直接按事务处理 Vuser 中的每个 Action 或 step。这里，Action 指的是 vuser_init、Action 和 vuser_end 三大函数，而 step 指的是 LoadRunner 执行的每个函数。LoadRunner 将 Action 名或 step 名指 定为事务名。

5.7. 数据驱动——参数化（Parameters）

数据驱动就是把测试脚本和测试数据分离开来的一种思想，脚本体现测试流程，数据体现测试案例。数据不是 hard-code 在脚本里面，这样大大提高了脚本的可复用性。而 LoadRunner 的参数化功能是数据驱动测试思想的一个重要实现。

在本节中，我们要学习的是：理解参数的局限性，建立参数，定义参数的属性，理解参数的类型，为局部数据类型设置参数的属性，为数据文件设置参数的属性，从已经存在的数据库中引入数据。

5.7.1. 为什么需要参数化

在录制程序运行的过程中，Vugen（脚本生成器）自动生成了脚本以及录制过程中实际用到的数据。在这个时候，脚本和数据是混在一起的。

比如，你用 VU 的 Web Vuser 录制一个用户登录 Web 系统的过程，对于登录的操作，会生成以下脚本：

```
web_submit_form(”auth”,

“Snapshot=t2.inf”,

ITEMDATA,

“Name=ssousername”,

“Value=robin”, ENDITEM,

“Name=password”, “Value=123456″, ENDITEM,
```

“Name=remember”, “Value=<OFF>”, ENDITEM,

LAST);

web_submit_form 是登录触发的动作，而“robin”和“123456”是填入的数据。如果Controller 里以多用户方式运行这个脚本的时候，每个虚拟用户都会以同样的用户名“robin”、密码“123456”去登录 Web 系统。这样做性能测试，我们的客户可能不会答应，因为这显然不是一个真实的业务场景。尤其现在服务器大多会采用 Cache 功能提高系统性能，用同样的用户名/密码登录系统的 Cache 命中率会很高，也要快得多。

因此，我们的客户希望当用 LoadRunner 多用户多循环运行时，不会只是重复一个用户的登录，也就是说，此函数中的数据要能变化，这样的话，就把这些数据用一个参数来代替，其实就是把常量变成变量。参数化后的脚本如下：

web_submit_form(”auth”,

“Snapshot=t2.inf”,

ITEMDATA,

“Name=ssousername”,

“Value={username}”, ENDITEM,

“Name=password”,

“Value={passwd}”, ENDITEM,

“Name=remember”, “Value=<OFF>”, ENDITEM,

LAST);

参数化后，用户名“robin”被一个参数{username}替换，密码“123456”被另外一个参数{passwd}代 替。{username}和{passwd}分别和参数文件关联，在脚本运行时，用户名和密码的值从参数{username}和{passwd}中获得。 而我们会在后面介绍 LoadRunner 有一套机制来保证参数的使用和变化，这样就实现了脚本与数据的分离。

参数化是我们学习 LoadRunner 中经常用到的功能。除了实现数据驱动之外，参数化脚本还有以下两个优点：

（1）可以使脚本的长度变短。

（2）可以增强脚本的可读性和可维护性。

实际上，参数化的过程如下：

（1）在脚本中用参数取代常量值。

（2）设置参数的属性以及数据源。

这些我们会在下面内容中详细介绍。

5.7.2. 参数的创建

LoadRunner 对脚本中参数个数没有限制，我们可以在一个脚本中创建任意多个参数。

下面以 Web Vuser 为例，看看 LoadRunner 是如何创建参数的。我们已经知道，VU 可以通过 Tree View 和 Script View 两种途径来改变脚本，包括参数化功能。我们这里分别介绍。

还是上面那个登录脚本，我们可以在基于文本的脚本视图中参数化。

1．脚本视图参数化

将光标定位在要参数化的字符上，单击右键，弹出快捷菜单，如图 5-26 所示。

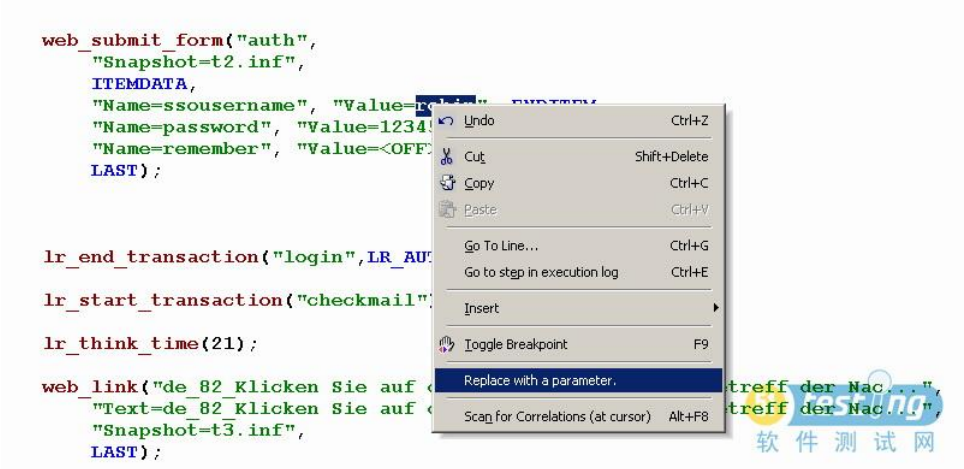


图 5-26　脚本参数化之右键选择替代参数

在弹出菜单中，选择“Replace with a Parameter”，打开选择或者创建参数对话框，如图

5-27 所示。

在“Parameter name”中输入参数的名称，或者选择一个在参数列表中已经存在的参数。

在“Parameter type”下拉列表中选择参数类型，如图 5-28 所示。

下面重点介绍一下参数的类型。

在定义参数属性的时候，要指定参数值的数据源。你可以指定下列数据源类型中的任何一种：

（1）Data Files

这是我们最常使用的一种参数类型，它的数据存在于文件中。该文件的内容可以手工添加，也可以利用 LoadRunner 的 Data Wizard 从数据库中导出。我们将在后面详细地介绍。

（2）User-Defined Functions

调用外部 DLL 函数生成的数据。

（3）Internal Data

虚拟用户内部产生的数据。

Internal Data 包括以下几种类型：

Date/Time

Date/Time 用当前的日期/时间替换参数。要指定一个 Date/Time 格式，你可以从菜单列表中选择格式，或者指定自己的格式。这个格式应该 和脚本中录制的 Date/Time 格式保持一致。

Group Name

Group Name 用虚拟用户组名称替换参数。在创建 scenario 的时候，你可以指定虚拟用户组的名称。注意：当从 VU 运行脚本的时候，虚拟用户组名称总是 None。

Load Generator Name

Load Generator Name 用脚本负载生成器的名称替换参数。负载生成器是虚拟用户在运行的计算机。

Iteration Number

Iteration Number 用当前的迭代数目替换参数。

Random Number

Random Number 用一个随机数替换参数。通过指定最大值和最小值来设置随机数的范围。

Unique Number

Unique Number 用一个唯一的数字来替换参数。你可以指定一个起始数字和一个块的大小。

注意：使用该参数类型必须注意可以接受的最大数。例如：某个文本框能接受的最大数为 99。当使用该参数类型时，设置第一个数为 1，递增的数为 1，但 100 个虚拟用户同时运行时，第 100 个虚拟用户输入的将是 100，这样脚本运行将会出错。

Vuser ID

Vuser ID 用分配给虚拟用户的 ID 替换参数，ID 是由 LoadRunner 的控制器在 scenario 运行时生成的。如果从脚本生成器运行脚本的话，虚拟用户的 ID 总是-1。

5.7.3. 定义参数的属性

创建参数完成后，就可以定义其属性了。参数的属性定义就是在脚本执行过程中，定义参数使用的数据源。在 Web 用户脚本中，既可以在基于文本的脚本视图 中定义参数属性，也可以在基于图标的树视图中定义参数属性。

1．使用参数列表

使用参数列表可以在任意时刻查看所有的参数、创建新的参数、删除参数，或者修改已经存在参数的属性。

单击参数列表按钮或者选择“Vuser”>“Parameter List”，打开参数列表对话框，如图 5-36 所示。

要创建新的参数，单击“New”按钮，新的参数则被添加在参数树中，该参数有一个临时的名字，你可以给它重新命名，然后回车。设置参数的类型和属性， 单击“OK”按钮，关闭参数列表对话框。

注意：不要将一个参数命名为“unique”，因为这个名称是用户脚本生成器本身的。用户脚本生成器创建新的参数，但是不会自动用该参数在脚本 中替换任意选中的字符串。

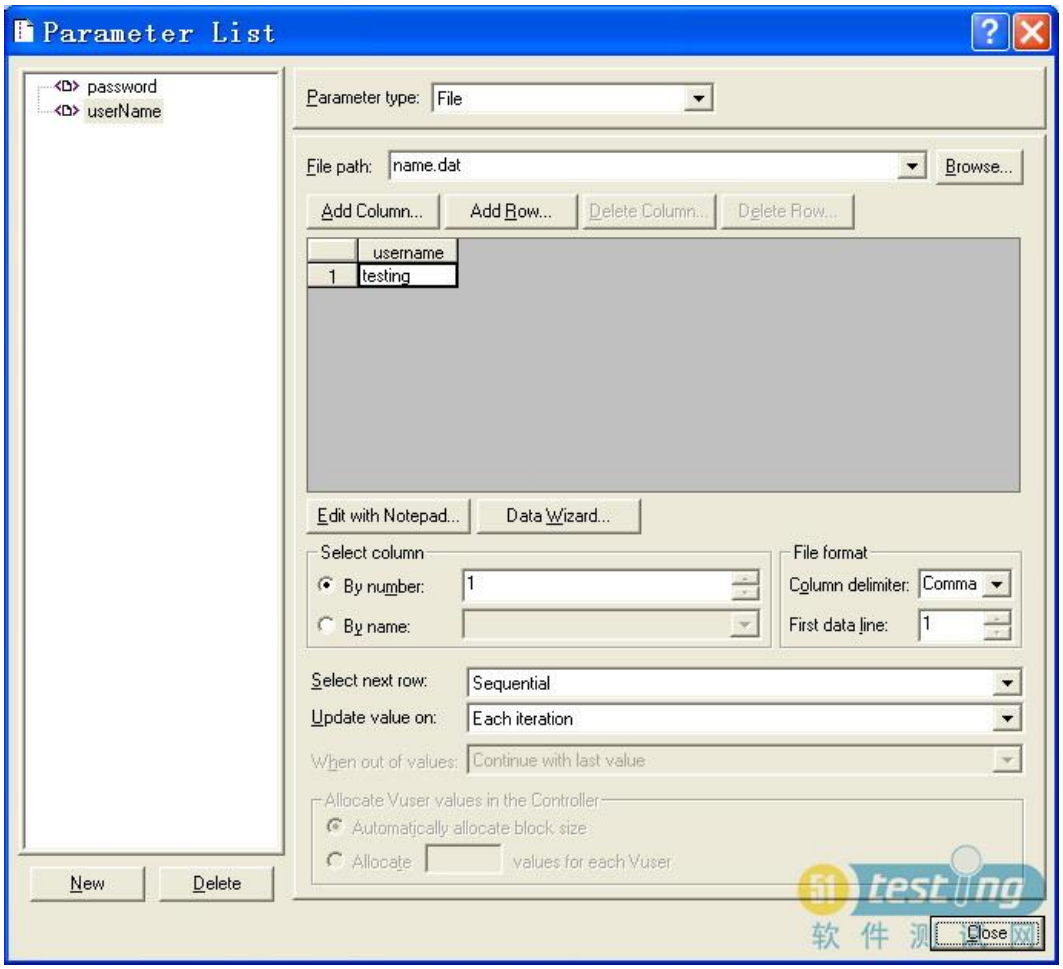


图 5-36　VU 参数列表对话框

要删除已有的参数，首先要从参数树中选择该参数，单击“Delete”按钮，然后确认你的行为即可。

要修改已有的参数，首先要从参数树中选择该参数，然后编辑参数的类型和属性。

2．数据文件

数据文件包含着脚本执行过程中虚拟用户访问的数据。局部和全局文件中都可以存储数据。可以指定现有的 ASCII 文件、用脚本生成器创建一个新的文件或 者引入一个数据库。数据文件中的数据是以表的形式存储的。一个文件中可以包含很多参数值。每一列包含一个参数的数据，列之间用分隔符隔开，比如用逗号。

如果使用文件作为参数的数据源，必须指定以下内容：文件的名称和位置、包含数据的列、文件格式、包括列的分隔符、更新方法。

如果参数的类型是“File”，打开参数属性（Parameter Properties）对话框，设置文件属性如下：

在“File path”中输入文件的位置，或者单击“Browse”按钮指定一个已有文件的位置，如图 5-37 所示。在默认情况下，所有新的数据文件名都 “parameter_name.dat”，注意，已有的数据文件的后缀必须是.dat。

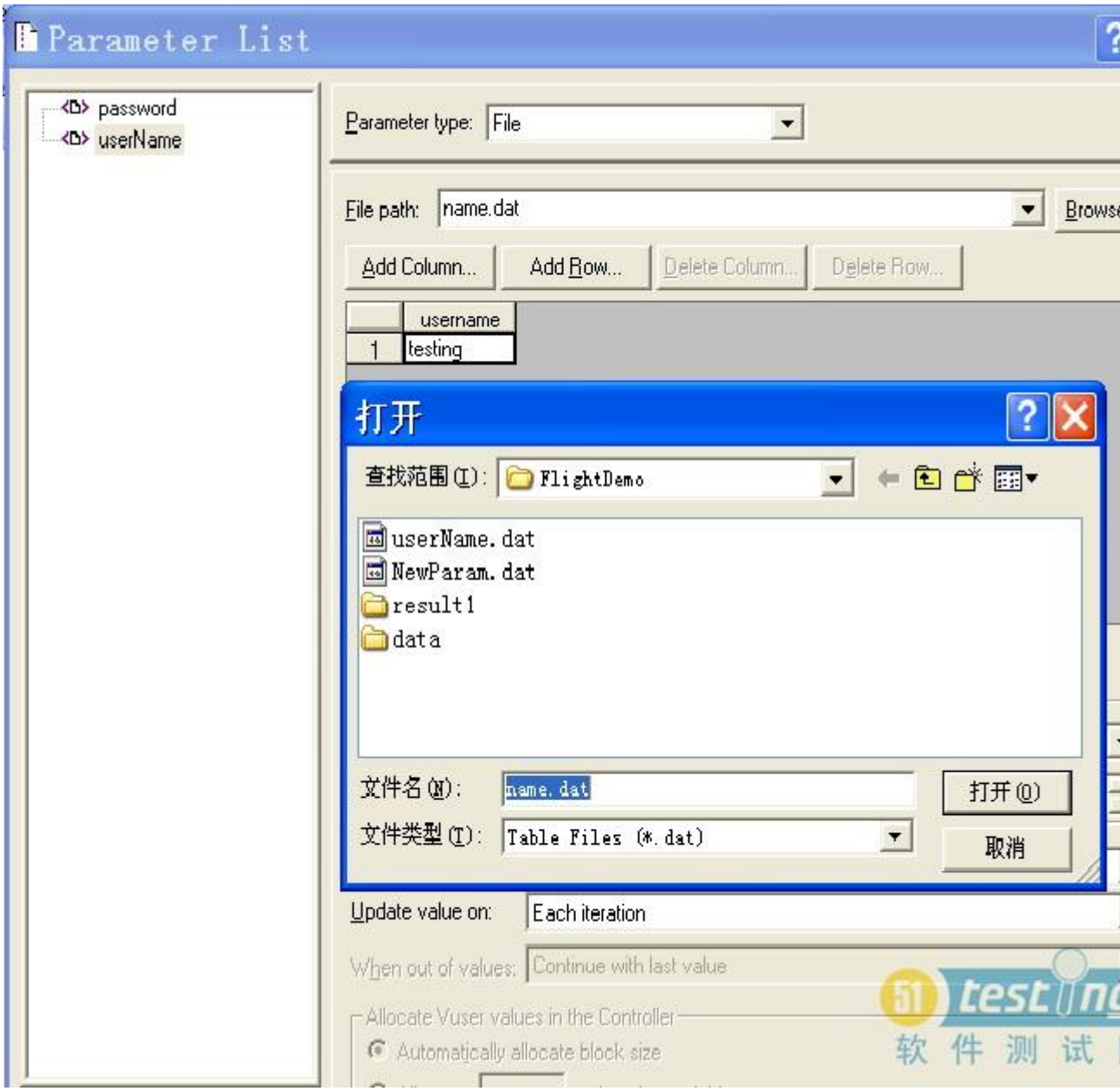


图 5-37 选择打开文件来导入参数

单击“Edit With Notepad”按钮，打开记事本，里面第一行是参数的名称，第二行是参数的初始值。使用诸如逗号之类的分隔符将列隔开。对于每一个新的表行开始一行新的 数据。

注意：在没有启动记事本的情况下如果想添加列，就在参数属性对话框中单击“Add Column”按钮，打开“Add new column”对话框。输入新列的名称，单击“OK”按钮，脚本生成器就会将该列添加到表中，并显示该列的初始值。

在“Select column”部分，指明选择参数数据的列。可以指定列名或者列号。列号是包含你所需要数据的列的索引；列名显示在每列的第一行（row 0）。

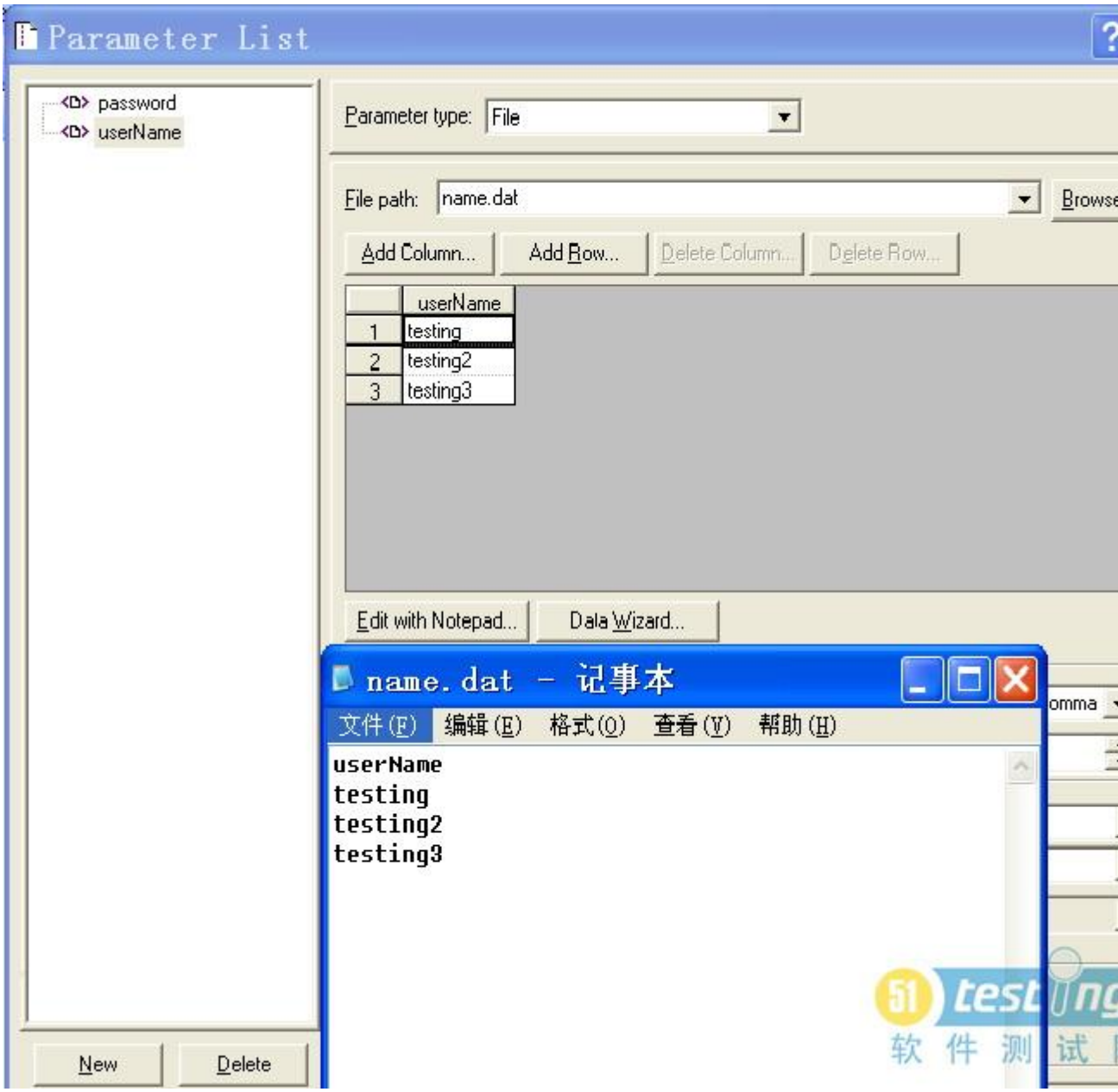


图 5-38　以记事本方式打开参数

5.7.4. 高级——从已存在的数据库中导入参数数据

LoadRunner 允许你利用参数化从已经存在的数据库中导入数据。可以使用下列两种方式之一：

（1）使用 Microsoft Query（要求在系统上先安装 MS Query）。

（2）指定数据库连接字符串和 SQL 语句。

用户脚本生成器在从数据库中导入数据的过程中提供了一个向导。在向导中，指明如何导入数据——通过 MS Query 创建查询语句或者直接书写 SQL 语句。在导入数据以后，以.dat 为后缀并作为正规的参数文件保存。要开始导入数据库中数据的过程，在参数属性对话框中单击 “Data Wizard”按钮，则打开数据库查询向导。

1．创建新的查询

(1) 选择“Create query using Microsoft Query”。如果需要 MS Query 的帮助，选择“Show me how to use Microsoft Query”，然后单击“Finish”按钮，如图 5-39 所示。

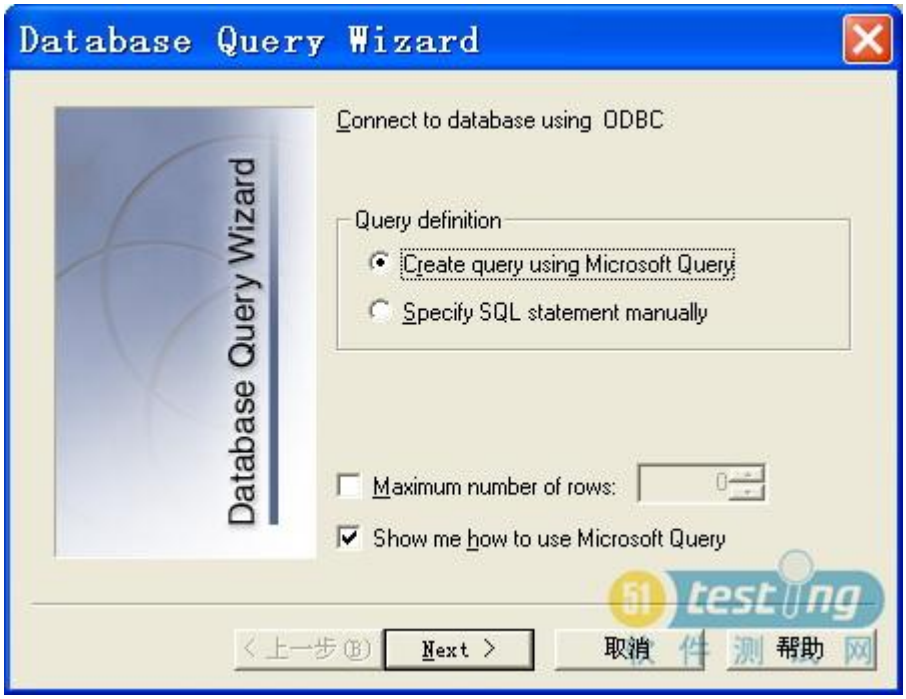


图 5-39 在 Data Wizard 中选择通过 Microsoft Query 创建查询

如果你还没有安装 Microsoft Query，LoadRunner 会提示你这个功能不可用。在进行操作之前，从 Microsoft Office 中安装 MS Query。

(2) 在 Microsoft Query 中导入期望的表和列。

- 选择表和字段（见图 5-40）

图 5-40 选择表和字段

- 设置过滤条件（见图 5-41）

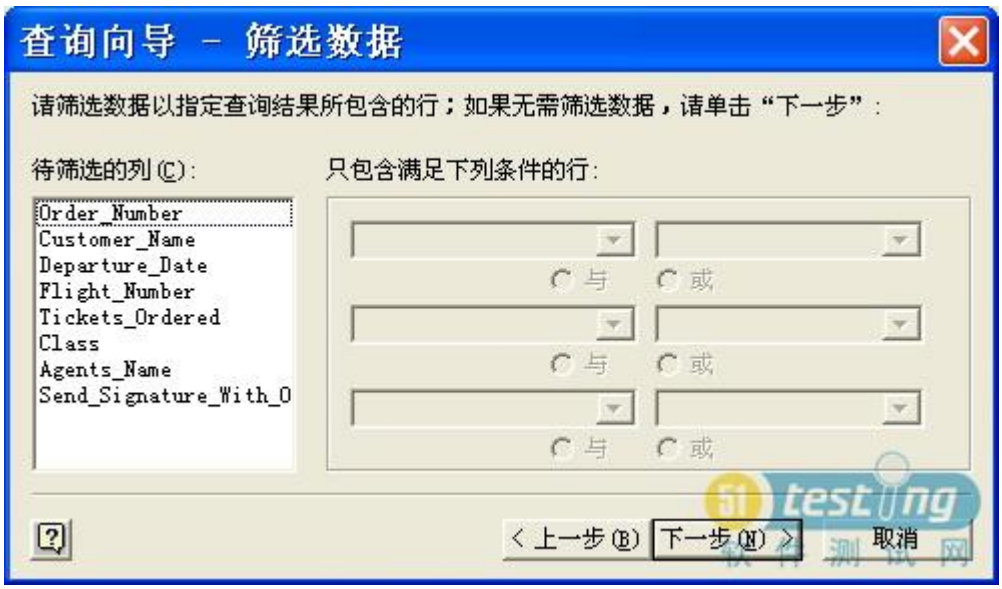


图 5-41 设置过滤条件

- 设置排序顺序（见图 5-42）

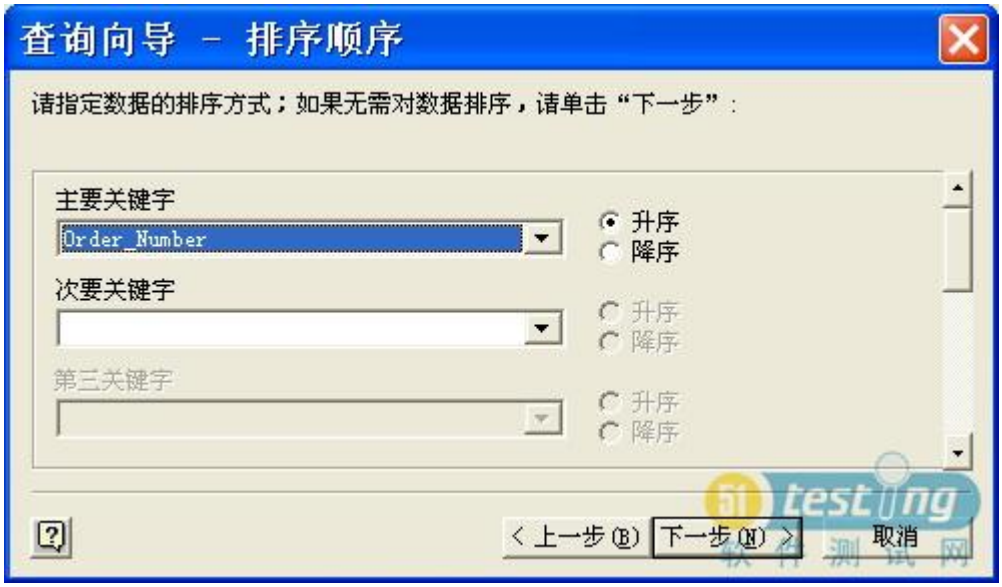


图 5-42 设置排序顺序

(3) 在完成数据的导入后，选择 “Exit and return to Mercury Virtual User Generator”，然后单击“完成”按钮，如图 5-43 所示。

图 5-43 完成查询并选择返回

在参数列表对话框中，数据库记录以 data 文件的形式显示出来，如图 5-44 所示。

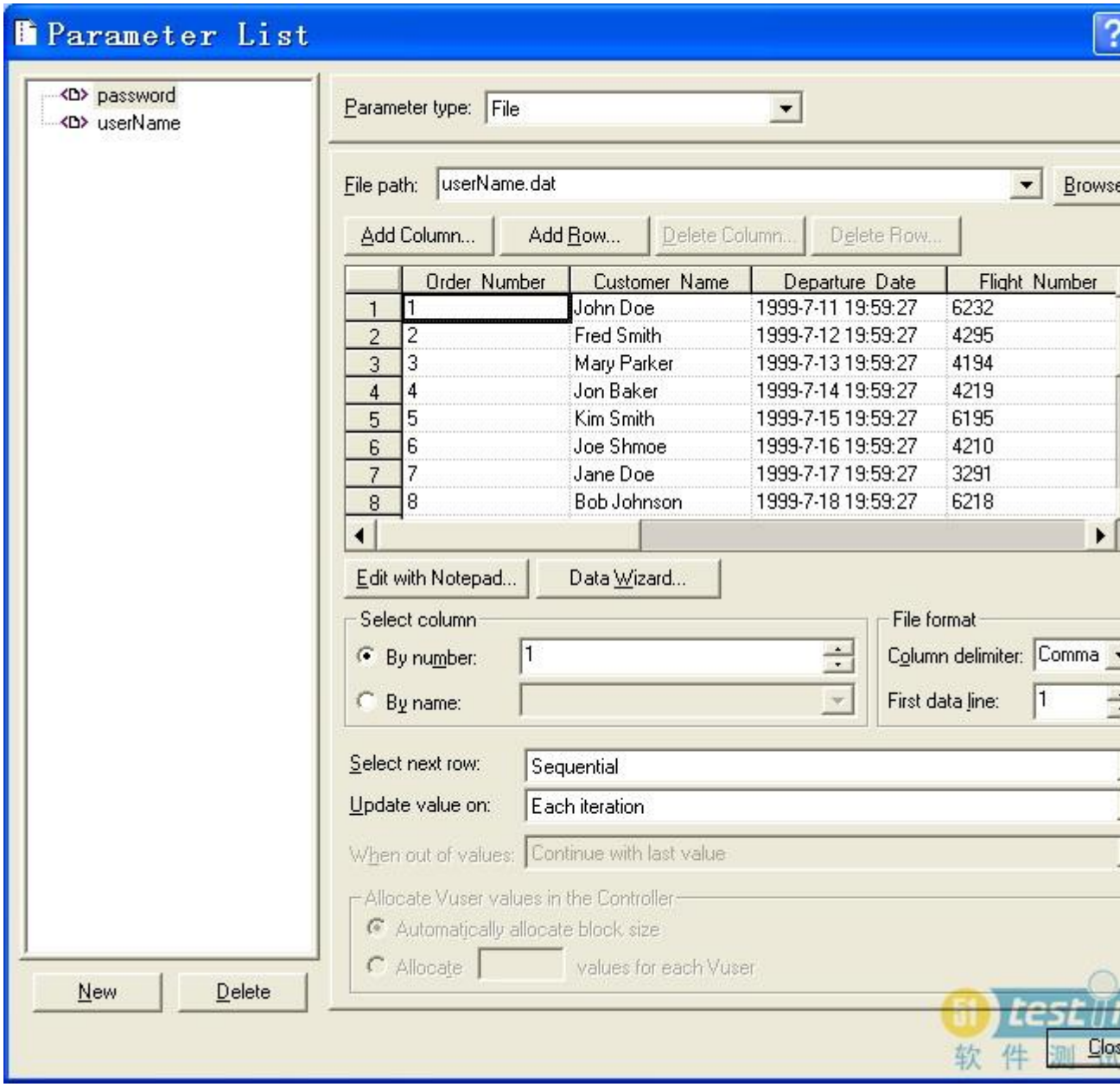


图 5-44 参数列表中返回所查询的数据

5.8. 检查点（Check point）

5.8.1. 序：为什么需要检查点

LoadRunner 的很多 API 函数的返回值会改变脚本的运行结果。比如 web_find 函数，如果它查找匹配的结果为空，它的返回值就是 LR_FAIL，整个脚本的运行结果也将置为 FAIL；反之，查找匹配成功，则 web_find 返回值是 LR_PASS，整个脚本的运行结果置为 PASS。而脚本的结果则反应在 Controller 的状态面板上和 Analysis 统计结果中。

提示：在 VU 函数手册中，点击函数的 Return Value 项，可查看此函数是否返回 LR_PASS/LR_FAIL，如图 5-50 所示。

图 5-50 在 function help 中查看函数的返回值

上图说明 web_image_check 也是一个决定脚本运行结果的函数。

但仅仅通过脚本函数执行结果来决定整个脚本的成功/失败，这未免太草率了。因为脚本往往是在执行一个业务流程，VU 脚本函数本身是协议级的，它执行的失败会引起整个业务的失败，但它运行成功却未必意味着业务会成功。比如，我们要测 100 人登录一个 Web 邮件系统，此邮件系统有个限制，即不允许使用同一个 IP 登录两个用户。显然，如果 LoadRunner 没有开启多 IP 欺骗功能的话，第一个虚拟用户登录成功后， 第二个虚拟用户试图登录，系统将返回一个页面，提示用户“您已经登录本系统，请不要重复登录！”。在这种场景下，如果没有设检查点来判断这个页面，那么 VU 认为它已经成功地发送了请求，并接到了页面结果（http 状态码为 200，虽然是个错误页面）。这样 VU 就认为这个动作是成功的。但事实如我们所见， 并非如此。因此我们要采用检查点来判断结果。

检查点（Check Point）并不是一个 LoadRunner 里专有的概念。在 WinRunner 和 QTP 中就有检查点。对于自动化测试来讲，检查点是一个很重要的功能，它的作用是验证程序的运行结果是否与预期结果相符。

对于 Web Vuser 类型，有两种设置检查点方法，下面进行介绍。

5.8.2. 检查点实施之一：ContentCheck 定义

在【Run-time settings】>【ContentCheck】中（见图 5-51），这里的设置是为了让 Vugen 检测何种页面为错误页面。如果被测的 Web 应用没有使用自定义的错误页面，那么这里不用作更改；如果被测的 Web 应用使用了自定义的错误页面，那么这里需要定义，以便让 Vugen 在运行过程中检测，服务器返回的页面是否包含预定义的字符串，进而判断该页面是否为错误页面。如果是，Vugen 就停止运行，指示运行失败。

图 5-51 通过 ContentCheck 定义检查点

5.8.3. 检查点实施之二：检查函数

另外一种设置方法是在 Web Vuser 里，通过检查函数来完成检查点功能。 Web Vuser 提供 Image Check 和 Text Check 两种方式。其原理就是在上一个请求页面的函数完成后，运行检查函数，在结果页面中搜索既定的图片/关键文字。

以下是取自 LoadRunner 函数手册的一个 Text Check 例子：

```
Web_url(”index.html”,

”URL=http://server1/people/employees.html”,

”TargetFrame=”,

LAST);

web_find(”Employee Check”,

”expect=notfound”,

”matchcase=yes”,

”onfailure=abort”,
```

```
”report=failure”,

”repeat=no”,

”what=John”,

LAST);

}
```

在这个例子中，web_find 函数在 employees.html 中搜索“john”关键字。有关 web_find 函数的各个参数的含义以及 使用方法，可参看 LoadRunner 随带的函数手册。

```
Image Check 的功能则由另外一个函数 web_image_check 实现：

web.url(”index.html”,

”URL=http://localhost/ImagesAndMaps.html”,

new String [] {

”TargetFrame=”, ”LAST”});

web_image_check(”Go2Venus”,

new String []{”Alt=Venus”, web.LAST});
```

这同样是一个 Web Vuser 脚本，是用 Java 实现的，而不是录制时默认生成的 C 语言。Web_image_check 则在 ImageAndMaps.html 中查找 alt 属性为“venus”的图片。

提示：LoadRunner 一直是使用 C 作为脚本语言的。在 Java 普及应用的当今，LoadRunner 开始加大对 Java 的支持力度，这表 现在 LoadRunner 对原先运行在 C 基础上的 Vuser 同样提供了 Java 运行环境，而一些经常使用的 LoadRunner C 函数也有了相同的 Java 函数实现。比如在 Web Vuser 中，原先的 web_url 函数摇身一变，在 Java 中成了 web.url；另外，对于 CORBA、RMI 的 Vuser，VU 可以直接录制生成 Java 脚本。需要注意的是：Java 脚本的录制选项和关联方法等与 C 脚本都有一些差别，可以参看 LoadRunner 的 VU 手册。

如果我们看过 LoadRunner 的函数手册，就会发现 VU 的 Web Vuser 还提供了和 web_find 十分貌似的另外一个检查点函 数：web_reg_find。

web_reg_find 里的 reg 意为注册（register）。因此 web_reg_find 和 web_find 的不同之处是 web_reg_find 是先注册，后查找；而 web_find 是查找前面的请求结果。因此，我们在使用 web_reg_find 函数的时候，将它放在请求语句的前面，如下：

```
web_reg_find(”Text=ABC”, ”SaveCount=abc_count”, LAST);

web_url(”MercuryWebTours”,

”URL=http://localhost/hello.html”,

”Resource=0″,

”RecContentType=text/html”,

”Referer=”,

”Snapshot=t1.inf”,

”Mode=HTML”,

LAST);

if (strcmp(lr_eval_string(”{abc_count}”), ”0″) == 0)

Action A

else

Action B
```

上面的脚本运行过程是：如果 web_reg_find 在 hello.html 页面中没有找到“welcome”字符串，则执行 Action A；如果找到了一次或一次以上，则执行 Action B。

我们可以看到，web_find 和 web_reg_find 函数两者是有一些差别的：

（1）web_reg_find 先注册的优势是脚本能够一边接收 Server 的数 据缓冲，一边进行查找，提高了查找的效率。

（2）web_reg_find 的参数与 web_find 并不完全一样，其中有个参数叫做 SaveCount，它能够记录查找匹配的次数。而 web_find 的机制是一旦查找匹配成功，就立即返回，并不

继续查找和记录匹配次数。

（3）VU run time 设置中的 “enable image and text check”对 web_find 有效，而对 web_reg_find 无效。

5.8.4. 检查点设置技巧

如何加入检查点，才能检查出正确的结果。与事实相符，这的确有一些技巧：

（1）它必须满足是验证事务通过与否的充分必要条件。检查点通过，我们就能够确信系统是一个正常的状态。

（2）检查点可以是常量，也可以是变量。

（3）检查点可以是文本、图像文件，也可以是数据库记录等。

5.9. 高级——多 Action

对于支持多 Action 的 Vuser，可以把一个脚本的 Action import 到另外一个脚本中。当然必须保证这两个 import 的脚本 Vuser 类型是相同的。注意，被 import 的 Action 中的有关参数也一同被 improt。选择被 import 的脚本，然后选择被 import 的 Action。步骤如下：

1、 选择“Action”>“Import Action into vuser”，会显示对话框，提示选择脚本。

2、 选择要 import 的 Action，单击确定按钮。

3、 生成一个新的 Action，名为 Imported_Action，函数都被 import 在此中。

在“Run-time Settings”>“Run Logic”中调整 Action 的运行策略。

打开“Run-time Settings”（运行时设置）对话框，然后选择“General”下的“Run logic”节点，如图 5-52 所示。

（1）迭代次数（Number of Iterations）：迭代的次数。LoadRunner 将 按指定的次数重复执行所有 Actions，但不会重复 Vuser 脚本的 vuser_init 和 vuser_end 部分。

需要注意的是，如果在 Controller 的计划设置中指定了方案持续时间，则持续时间设置将覆盖 Vuser 迭代设置。这意味着，如果将持续时 间设置为 5 分钟（默认设置），Vuser 将

在 5 分钟内按照需要继续运行任意多次迭代，即使运行时设置仅指定一次迭代。

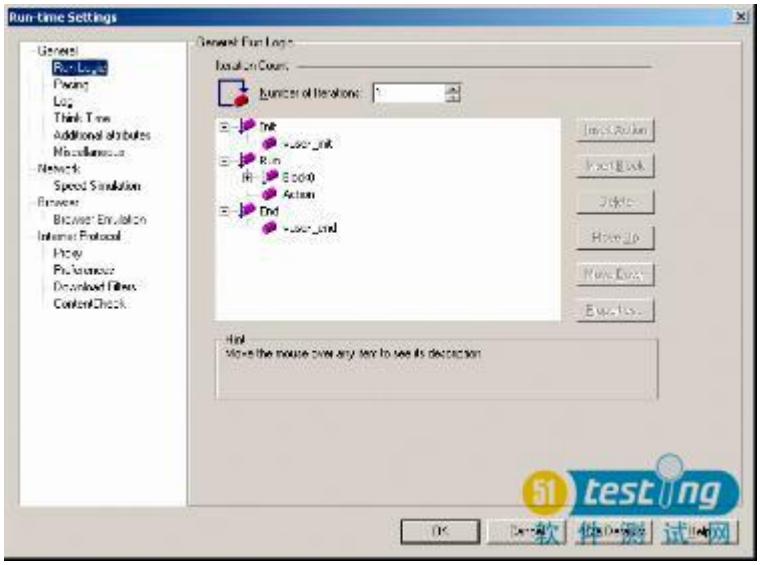


图 5-52 在 Run-time Settings 中设置运行逻辑

（2）操作块（Block）：操作块是脚本内函数的组合。每个块都有自己的单独属性设置，包括顺序、迭代和权重。可以通过单击 “Properties”设置 Block 的属性。

（3）顺序（Sequence）：可以设置块内脚本的操作顺序。有两种方式：按顺序和随机执行操作。

（4）迭代（Iteration）：除了为整个 Run 部分设置迭代次数以外，还可以设置单个操作或操作块的迭代。该设置非常有用，例如，可以用 来模拟一个商业站点，你在该站点执行了多个查询查找某个产品，但只进行了一次采购。

提示：

操作块（Block）是 Vuser 脚本的操作组。这是 VU 为用户提供的一个很贴心的功能。

比如在一个银行转账系统中，系统提供多个业务，有查询、转账、储蓄等业务，但每个业务被执行的概率可能会不一样。在这种情况下，我们可以把查询 设为 Block1，转账设为 Blcok2，储蓄设为 Block3，各个 Block 的 Iteration 属性体现了它们的概率分配，同时“登录”和“注销” 操作是这三个 Block 所共有的。这样，执行一次脚本，就执行了所有的业务。当然，如果 VU 没有 Block 功能，我们也可以使用脚本 C 语言的循环机制来实 现。有兴趣的朋友可以思考一下如何做。

4、 运行脚本。验证 Action 确实按照我们在 Run-time Settings 中的设置运行。

提示：运行时设置（Run-time Settings）

VU 中有运行时设置（Run-time Settings），而 Controller 在运行时也同样有一个运行时设置。这两个运行时设置虽然看 起来是一样的，但其实并不出自一处，而是分别存储在各自的环境下。VU 的 Run-time Settings 存储在 default.cfg 文件中，而 Controller 则把自己的 Run-time Settings 放在自己的场景文件下。

这也就是说，当通过脚本创建场景后，再修改脚本的 Run-time Settings，这不会影响到 Controller 的 Run- time Settings。而在 LoadRunner 早期版本中（6.5 以前），VU 和 Controller 是共享同一个 Run- time Settings 的。后来把两者分开，使它们“各自为政”，应该是出自方便的考虑。比如，在 VU 中我们修改 Run-time Settings 很多是为了 debug 脚本，输出尽可能多的日志，忽略 Think Time；而 Controller 中运行脚本 则是开始性能测试，这时的 Run-time Settings 策略会有变化。

5.10. 高级——脚本错误处理机制

一个好的脚本应该具有健壮性，要能够捕捉到错误，并能采取有效的错误处理方式，否则脚本一旦发生错误，唯一的出路就是退出执行。

VU 提供了一套在出错情况下的脚本处理机制。VU 的错误处理机制可设定 Vuser 在执行脚本时遇到错误怎样处理。当 Vuser 遇到错误时，可以 有两种处理策略：一是停止执行脚本，这适用于严重的问题；二是忽略这个错误，继续执行下去。

在默认情况下，当检测到错误时，Vuser 将停止执行脚本，在 Run-time Settings 中我们就可以看到它的默认设置，如图 5-53 所示。

如不想 VU 在出错后结束，而是继续执行。那么在运行时设置中，可以勾选“Continue on error”（出现错误时仍继续）。这个设置 适用于整个 Vuser 脚本，它是一个全局开关。

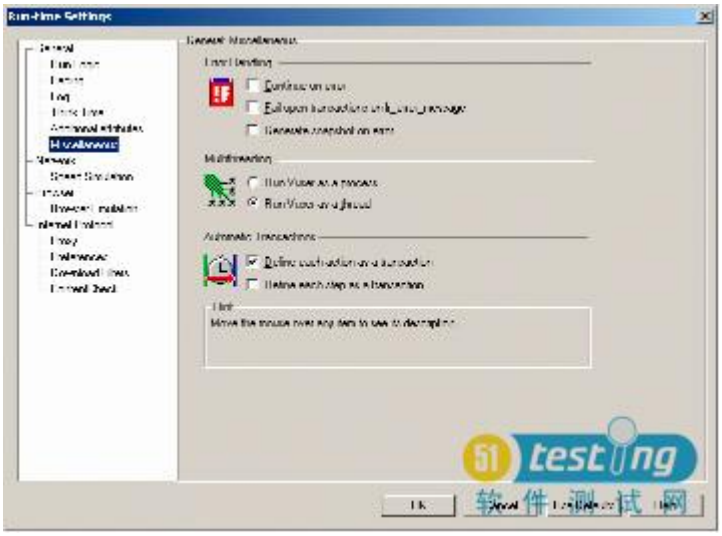


图 5-53 在 Run-time Settings 中设置出错处理方式

但这种方法有一个毛病，就是不够灵活，有“一竿子打翻一船人”之嫌，我们更希望脚本能够“具体情况具体分析”，能够对于不同的函数应用不同的错 误处理机制。既然信不过 LoadRunner，那么解决的办法就是我们自己来干，在脚本中加 入 lr_continue_error 函数。

使用 lr_continue_on_error 函数。通过 lr_continue_on_error 函数可以控制 Vuser 脚本特定段的错误处 理。要标记该段，就用 lr_continue_on_error(1)和 lr_continue_ on_error(0)语句将其括起来。

使用 lr_continue_on_error 函数的脚本段将会覆盖“出现错误时仍继续”的 Run-time Settings 运行时设置。

例如，下面的脚本，其运行时设置没有勾选“Continue on error”， VU 如果访问 www.51testing.com 遇到错误， 将会结束执行脚本。

```
Web_lin(”test“,

”text = www.51testing.com”,

LAST);

Lr_output_message(”finished linking”);
```

运行时，如 web_link 出错，“finished linking”这段文字永远也不会输出。但是如果我们认为这个 web_link 函数无 关紧要，它即使失败，我们也要继续执行脚本。那么解决的办法是用相应的 lr_continue_on_error 语句将该段括起来：

```
lr_continue_on_error(1);

Web_lin(”test”,

”text = www.51testing.com”,

LAST);

Lr_continue_on_error(0);

Lr_output_message(” finished linking”);
```

在上面的脚本中，即使 web_link 函数执行失败，lr_output_message(”finished linking”)也会被继续 执行。第一个 lr_continue_error(1)是将“Continue on error”设为 true，其后面的语句的错误处理机制都被应用为 “Continue on error”，直到 VU 遇见 lr_continue_error(0)，再将“Continue on error”恢复成 false。

5.11. 高级——脚本调试技巧

解释性语言的调试一直是一个棘手的问题。如果没有专门的 debug 工具，只能通过加入变量输出语句来查看每个变量值的变化。比如，在过去 JavaScript 调试的常用方法就是删除所有代码，然后一行行地增加直到错误出现，这是相当麻烦的。VU 的 C 脚本虽然也像 VC 一样提供了断点、单步运行等功能，但并不能像 VC 一样在调试状态下查看变量、表达式和内存值。因此，个人认为 VU 的调试功能还有待增强。

下面是两种比较常用的调试脚本的方式，它们并不是真正意义上的调试，只能算作“土办法”，但比较有效。

5.11.1. 动态运行（Animated run）与非动态运行（Non-Animated run）

脚本可以设置成为 Animated 运行模式和非 Animated 运行模式。在 Animated 运行模式下，VU 会显亮当前正在执行的脚本语句。 用户可以通过 VU 设置语句执行的延迟时间，以便观察每行代码执行的效果；相比之下，在非 Animated 运行模式下，VU 在执行脚本时不会显亮语句，而一 气执行下去。显然，Animated 运行模式是我们调试脚本的一个很好的方法。

Animated 的设置在 VU“Tools”菜单下的“Options”中，选中“General Options”中的“Replay”选项卡，在此对话框中设置 Animated 的 Replay 时间，生效范围，如图 5-54 所示。

图 5-54 在 General Options 中设定动态运行和非动态运行

然后在“View”菜单中勾选“Animated Run”方式，如图 5-55 所示，则脚本就能以 Animated 的模式运行了。

图 5-55 在“View”菜单中勾选动态运行

提示：当执行脚本期间在输出窗口中显示错误消息时，我们可以双击该错误消息，Vugen 将使光标跳到导致问 题的测试行； 如要得到更详细的出错信息，也可以将光标置于错误代码上并按 F1 键，查看该错误代码的联机帮助解释。

5.11.2. 日志设置

对于软件开发人员来讲，日志无疑是跟踪和调试最好的线索。一般在日志中存放的是程序产生的各 种信息，包括业务执行、警告和错误。一个优秀的软件应该有一套完备的日志系统，包括日志的内容、格式和级别等内容。在 Java 中，就有专门的 log 类来完 成这些工作。 同样地，VU 也提供了一套日志系统，我们可以通过 Run-time Settings 的 Log 选项来设置日志的级别，如图 5-56 所示。

（1）仅在出错时发送消息（Send messages only when an error occurs）：指示 Vugen 仅当出错时记录 log。

（2）标准日志（Standard log）：创建在脚本执行期间所发送的函数和消息的标准日志，供调试时使用。大型负载测试方案应禁用该选 项。

（3）扩展日志（Extended log）：创建扩展日志，包括警告和其他消息。大型负 载测试方案禁用该选项。

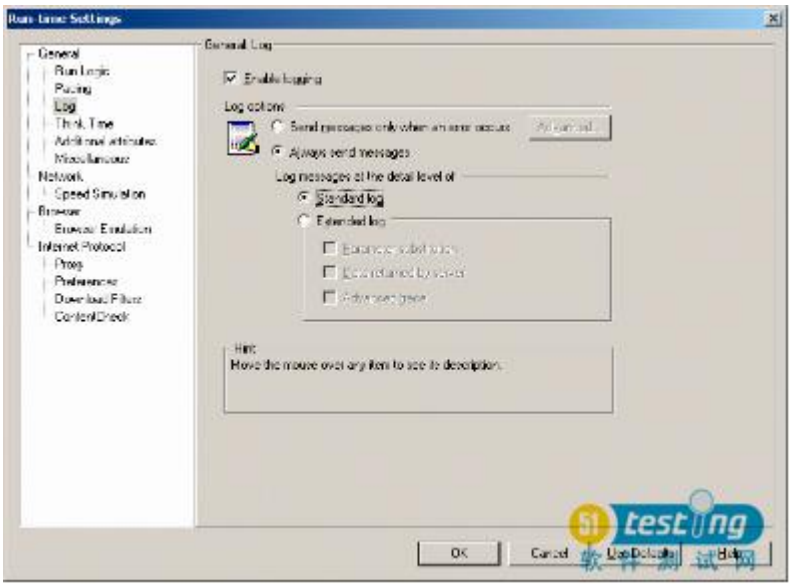


图 5-56 在 Run-time Settings 中设置日志级别

5.12. 高级——编写脚本的最佳实践

有过开发经验的朋友都知道，程序的开发大多不是一蹴而就的，通常要经过代码评审、编码和测试等流程后，才会趋于稳定。而 VU 脚本本身虽然不算复杂，但 VU 脚本运行往往涉及多个因素，也会经常出问题。提早发现和解决脚本中的问题，不仅从技术上下工夫，在流程上也要做文章。通常验证一个脚本的比较好的过程是这样的：

① Generate：录制或开发脚本。

② SUSI（Single User Single Iteration，单用户单循环）：运行录制生成的脚本，解决可能存在的关联问题。

③ SUMI（Single User Multi Iterations，单用户多循环）：参数化脚本，在 Run-time 中设置 Iteration，再次 运行，验证参数化问题。

④ MUSI（Multi User Single Iterations，多用户单循环）：Controller 里多用户运行脚本，验证脚本中可能的多线程 同步问题。

⑤ MUMI（Multi User Multi Iteration，多用户多循环）：即性能测试开始。

其中①~③是在 VU 中进行的，而④和⑤是在 Controller 中进行的。

VU 编写脚本最佳实践流程图如图 5-57 所示。

图 5-57 VU 编写脚本最佳实践流程图

6. Controller——性能测试的指挥中心

6.1 得到细化后的性能需求

6.2 设计性能测试场景

6.2.1 创建手工场景

6.2.2 百分比模式创建手工场景

6.2.3 创建面向目标场景

6.2.4 多 IP 的实现原理以及模拟

6.3 运行场景

6.3.1 场景控制

7. Analysis——寻找系统瓶颈的得力助手

8. 欲善其事先利其器——VU 脚本开发实战

VU 开发脚本是我们使用 LoadRunner 做性能测试中一个重要的步骤，这里涉及软件系统架构、计算机编程技巧和 VU 本身的功能特性。在本章，我们将结合邮件系统，介绍如何使用 VU 生成基于 SMTP/IMAP 协议的脚本。

我们将按照如下的思路完成脚本。

熟悉 SMTP/POP3/IMAP 协议规范和原语。

使用 VU 录制邮件收发操作，分析 VU 脚本对标准 SMTP/IMAP 协议的封装与实现机理。

使用 VU Java 模板用户嵌入 Java message 接口，实现邮件收发，并以此介绍 VU Java 模板用户的使用方法，以及 Java classpath 等相关设置。

8.1. 邮件服务 SMTP/IMAP 协议介绍

电子邮件是我们日常工作中经常使用到的一种交流方式，它是 Internet 应用最广的一种服务。

8.1.1. 电子邮件的工作原理

电子邮件是 Internet 上最为流行的应用之一。如同邮递员分发投递传统邮件一样，电子邮件也是异步的，也就是说，人们是在方便的时候发送和阅读邮件的，无须预先与别人协同。与传统邮件不同的是，电子邮件既迅速，又易于分发，而且成本低廉。另外，现代的电子邮件消息可以包含超链接、HTML 格式文本、图像、声音甚至视频数据。

电子邮件的工作过程遵循客户-服务器模式。每份电子邮件的发送都要涉及发送方与接收方，发送方构成客户端，而接收方构成服务器，服务器含有众多用户的电子信箱。发送方

通过邮件客户程序，将编辑 好的电子邮件向邮件服务器（SMTP 服务器）发送。邮件服务器识别接收者的地址，并向管理该地址的邮件服务器（IMAP 或 POP3 服务器）发送消息。邮件 服务器将消息存放在接收者的电子信箱内，并告知接收者有新邮件到来。接收者通过邮件客户程序连接到服务器后，就会看到服务器的通知，进而打开自己的电子信箱来查收邮件。

8.1.2. SMTP 协议介绍

简单邮件传送协议（SMTP）是 Internet 电子邮件系统首要的应用层协议。它使用由 TCP 提供的可靠的数据传输服务把邮件消息从发信人的邮件服务器传送到收信人的邮件服务器。SMTP 协议服务的默认端口是 25。

SMTP 协议与人们用于面对面交互的礼仪之间有许多相似之处。首先，运行在发送端邮件服务器主机上的 SMTP 客户，发起建立一个到运行在接收端 邮件服务器主机上的 SMTP 服务器端口号 25 之间的 TCP 连接。如果接收邮件服务器当前不在工作，SMTP 客户就等待一段时间后再尝试建立该连接。这个连接建立之后，SMTP 客户和服务器先执行一些应用层握手操作。就像人们在转手东西之前往往先自我介绍那样，SMTP 客户和服务器也在传送信息之前先自我介绍一下。在这个 SMTP 握手阶段，SMTP 客户向服务器分别指出发信人和收信人的电子邮件地址。彼此自我介绍完毕之后，客户发出邮件消息。SMTP 可以指望由 TCP 提供的可靠数据传输服务把该消息无错地传送到服务器。如果客户还有其他邮件消息需发送到同一个服务器，它就在同一个 TCP 连接上重复上述过程； 否则，它就指示 TCP 关闭该连接。

让我们看一个客户（C）和服务器（S）交互的例子。前面标以“C:”的文本行是名为 Mike 的客户端发送的请求，Mike 想给 Rose 发送一封 “I love you”的电子邮件情书，前面标以“S:”的是 cesoo.com 服务器的回应。以下传输内容在 TCP 连接建立之后马上发生。

S:220 cesoo.com

C:HELO

S:250 Hello Mike, pleased to meet you

C:MAIL FROM: Mike@cesoo.com

S:250 Mike@cesoo.com ... Sender OK

C:RCPT TO: rose@cesoo.com

S:250 rose@cesoo.com...Recipient OK

C:DATA

S:354 Enter mail, end with "." on a line by its self

C: I love you, Rose

C: .

S:250 Message accepted for delivery

C:QUIT

S:221 cesoo.com closing connection

觉得难以置信么？这不是在写言情小说，而确确实实是客户端和邮件服务器通过 SMTP 协议在网络上交互的内容，网络协议就这么简单。客户总共发出了 5 个命令，分别为：HELO、MAIL FROM、RCPT TO、DATA 和 QUIT。这些命令又叫做原语，可理解为应用协议层上最原始最小的命令颗粒。服务器给每个客户端的命令发回应答，其中每个响应都由返回码 和一些英语解释构成。这里需要指出的是，SMTP 使用持久连接，也就是说，如果客户端有多个邮件消息需发送到同一个邮件服务器，那么所有这些消息可以在同一个 TCP 连接中发送。对于其中的每一个消息，客户端以一个新的“HELO”命令开始整个消息发送过程，但是 QUIT 命令要等到所有消息都发送完之后才发出。

8.1.3. POP3 协议介绍

大家一听这个 POP，读起来有点像中文中的泡泡，其实这是一个英文术语的缩写。POP 的全称是 Post Office Protocol，即邮局协议，用于电子邮件的接收，它使用 TCP 的 110 端口。现在常用的是第三版，所以简称为 POP3。POP3 仍采用 Client/Server 工作模式，Client 被称为客户端，一般我们日常使用电脑都是作为客户端，而 Server（服务器）则是 POP3 的邮件服务器。举个形象的例子：Server（服务器）是许多小信箱的集合，就像我们所居住楼房的信箱结构，而客户端就好比是一个人拿着钥匙去信箱开锁取信，一样的道理。

POP3 和上面的 SMTP 协议一样，其实现也是一个客户端与服务器的对话过程。

当我们单击了电子邮件软件中的收取按钮后，电子邮件软件首先会调用 DNS 协议对 POP 服务器进行解析 IP 地址，当 IP 地址被解析出来后，邮件程序便开始使用 TCP 协议连

接邮件服务器的 110 端口，因为 POP 服务器是比较忙的，所以在这个过程中我们相对要等比较长的时间。当邮件程序成功地连上 POP 服务器后，其先会使用 USER 命令将邮箱的账号传给 POP 服务器，然后再使用 PASS 命令将邮箱的账号传给服务器。当完成这一认证过程后，邮件程序使用 STAT 命令请求服务器返回邮箱的统计资料，比如邮件总数和邮件大小等，然后 LIST 命令便会列出服务器里邮件数量。接下来邮件程序就 会使用 RETR 命令接收邮件，接收一封后便使用 DELE 命令将邮件服务器中的邮件置为删除状态。当使用 QUIT 命令时，邮件服务器便会将置为删除标志的邮件给删了。通俗地讲，邮件程序从服务器接收邮件，其实就是一个对话过程，POP 协议就是用在电子邮件中的一门语言。

8.1.4. IMAP 协议介绍

用户使用 POP3 把邮件消息下载到本地机之后，就可以把它们移动到本地创建的文件夹中。用户然后可以删除邮件，移动邮件，按发信人名字或消息主题搜索邮件等。然而，所有这些邮件操作都是在本机上完成的。这对于游动的用户却构成了问题，游动用户更愿意在远程邮件服务器主机上维护邮件夹，这样从任何主机都可以访问它，使用 POP3 是不可能做到这一点的。

这时 IMAP 协议就应运而生了，同样也是邮件接收协议，但是 IMAP 却比 POP3 复杂得多，因为 IMAP 提供的特性比 POP3 多出不少。 IMAP 被设计成允许用户像对待本地邮箱那样操纵远程邮箱。具体地说，IMAP 使得收信人能够在自己的邮件服务器主机中创建并维护多个存放邮件的文件夹。他们可以把邮件存入文件夹，也可以将邮件从一个文件夹转移到另一个文件夹，还可以在这些远程邮件夹中搜索匹配特定准则的邮件消息。IMAP 的实现比 POP3 的实现复杂得多，原因之一就是 IMAP 服务器必须为每个用户维护一个文件夹层次结构。某个用户使用不同的客户端相继访问自己的 IMAP 服务器时，这个 IMAP 服务器为该用户维护并同步相应的状态。POP3 服务器则相反，一旦用户退出当前的 POP3 会话，它们就不再为用户维护状态信息了。

8.2. VU 的 SMTP Vuser 对 SMTP 协议的封装及实现

上面我们分析了 SMTP/POP3/IMAP 各自的协议原理及相应实现原语，下面我们通过对比 SMTP 协议原语与 VU 录制发送邮件操作而产生的 脚本函数，来看一下 VU 是如何对 SMTP 协议进行封装的。在录制之前，需保证本机已经安装邮件客户端，本例中采用 Outlook 2003，并且已经配置好指向相应的邮件服务器。

配置步骤如下：

在控制面板中，双击“邮件”图标，弹出如图 8-1 所示的对话框。

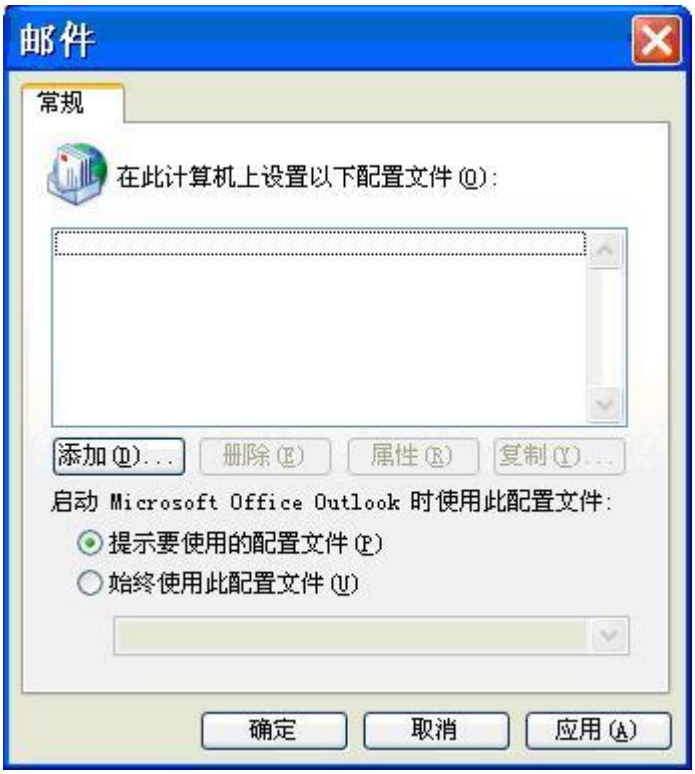


图 8-1　邮件配置文件列表

单击“添加”按钮，输入一个配置文件名，比如“cesooMail”，然后单击“确定”按钮，弹出如图 8-2 所示的对话框。

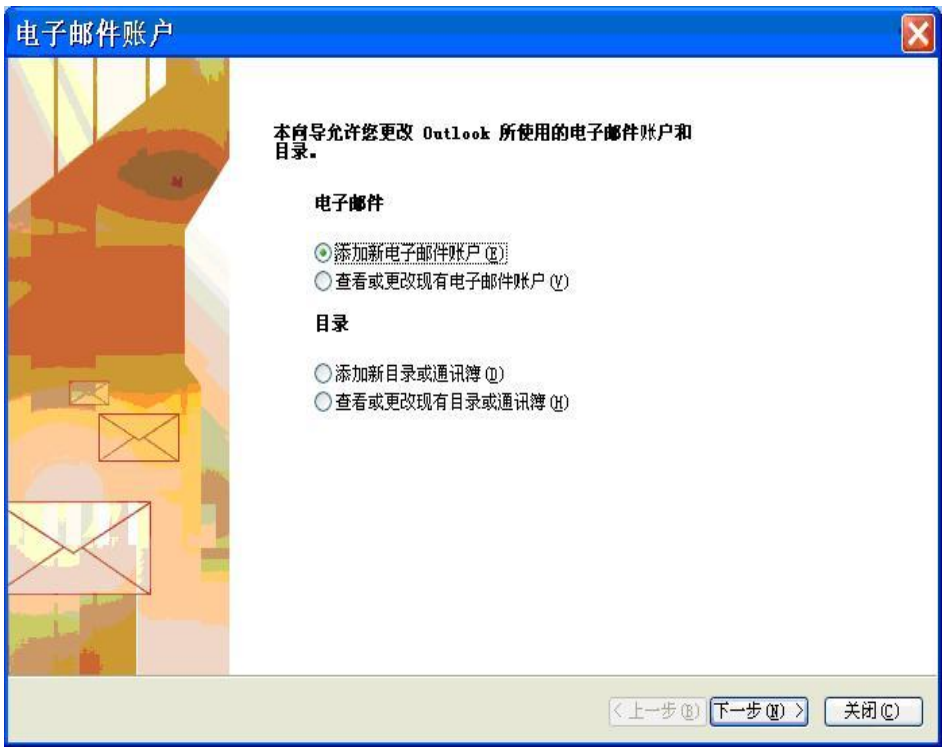


图 8-2　邮件配置向导第一步

保持图中的默认选项不变，单击“下一步”按钮，进入邮件配置向导第二步，如图 8-3 所示。

勾选“IMAP”类型，单击“下一步”按钮，进入邮件账户配置信息对话框，在对话框里输入相应的账号和 SMTP/IMAP 服务器信息，注意这里的账号和邮件服务器信息应该与你实际环境信息保持一致。在本例中，我们的 IMAP 服务器和 SMTP 服务器同为 cesoo.com，IMAP 端口号是默认的 143，SMTP 端口号是默认的 25，账号用户名为 test1，如图 8-4 所示。

图 8-3　邮件配置向导第二步

图 8-4　邮件配置详细信息

确认输入信息完整后，单击“下一步”按钮，弹出提示成功信息对话框，Outlook 2003 的 IMAP 和 SMTP 配置完成，如图 8-5 所示。

配置文件成功创建后，我们就可以收发邮件了。而 VU 则能把这个过程全部捕捉下来，生成脚本。

图 8-5　邮件配置成功页面

8.2.1. 使用 SMTP Vuser 录制 Outlook 2003 发送邮件

打开 VU，在“File”菜单下，选择“New”，在 Vuser 类型列表中，选中“Simple Mail Protocol（SMTP）”，如图 8-6 所示。

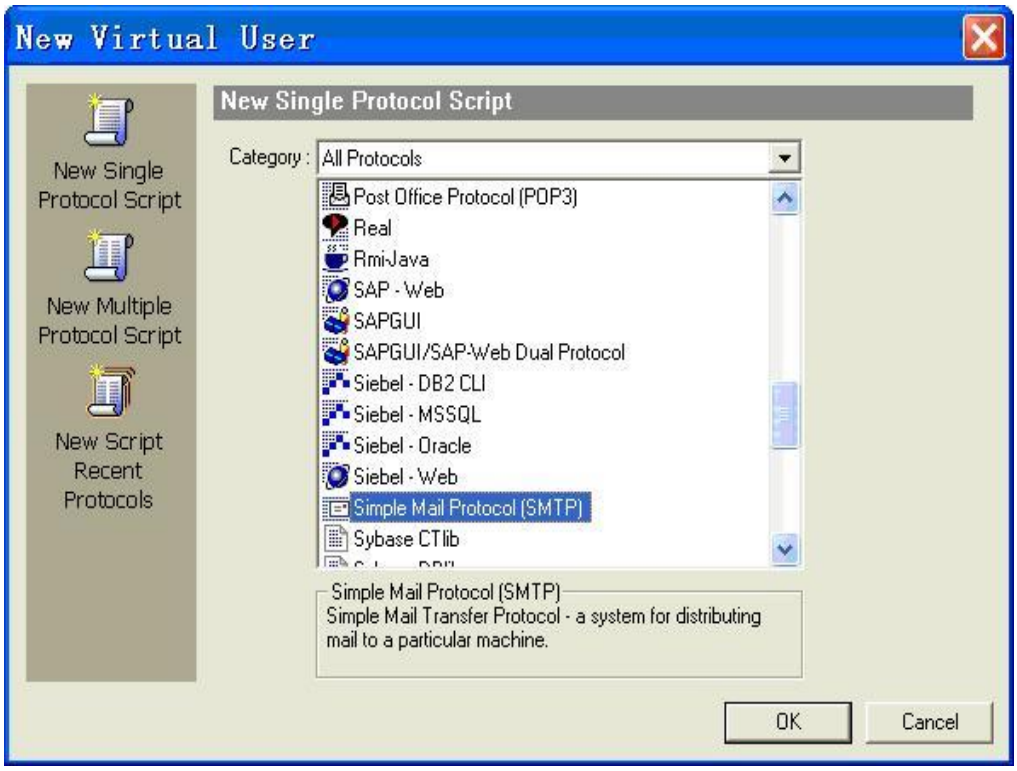


图 8-6　在虚拟用户列表中选择 SMTP Vuser

单击“Record”按钮，会弹出如图 8-7 所示的录制选项设置对话框。

图 8-7　在录制选项中设置 Outlook 应用程序路径

“Application type”（应用程序类型）选择“Win32 Applications”，在“Program to record”（录制程序）选项中输入 Outlook 2003 的应用程序路径，在本例中为“C:\Program Files\microsoft office\OFFICE11\OUTLOOK.EXE”，单击“OK”按钮，开始录制。

录制开始后，VU 会找到应用程序路径，并启动 Outlook 2003。其主界面如图 8-8 所示。

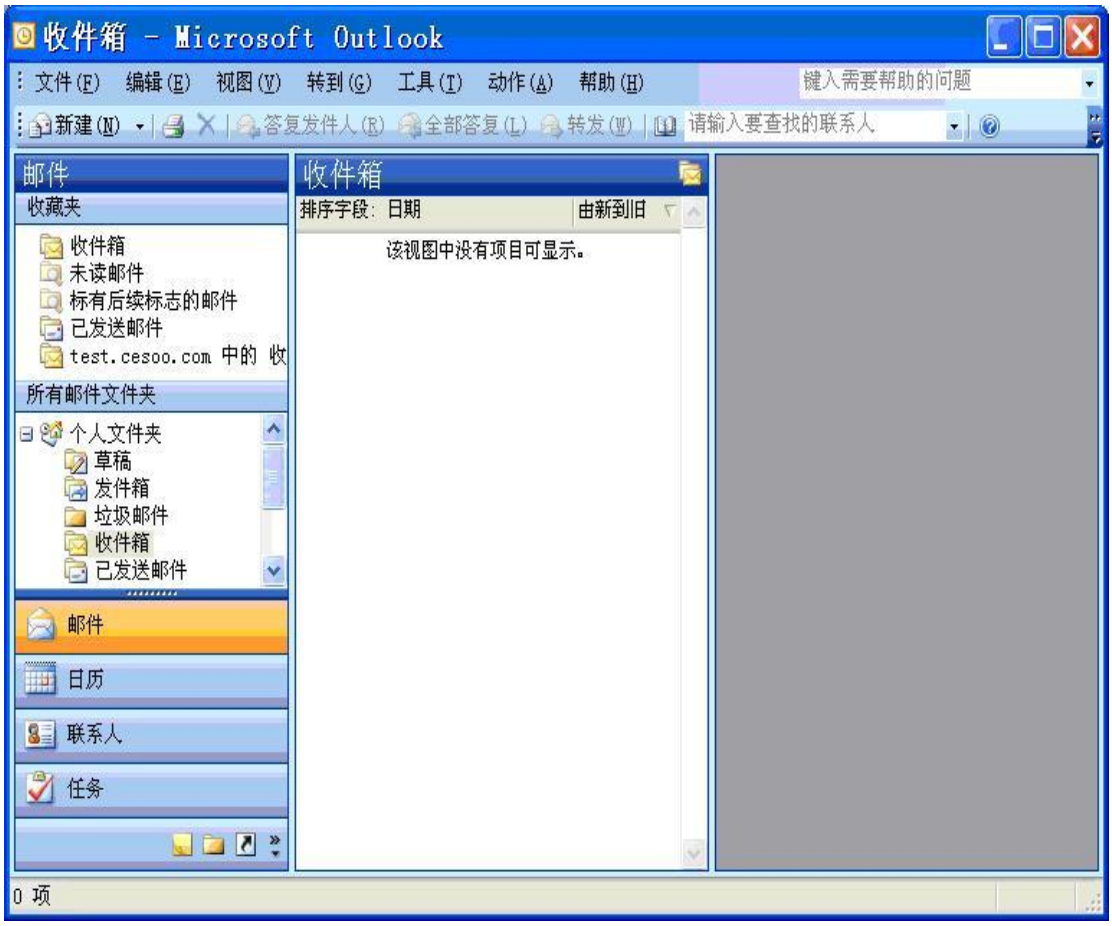


图 8-8　Outlook 启动成功后的主页面

这时 VU 录制工具条有如下提示，如图 8-9 所示。

图 8-9　Outlook 登录引起网络 67 次交互事件

这说明 Outlook 2003 完成登录这个操作，在网络上已经有了 67 次交互。

下面我们要重点观察发送邮件的操作，因此要在这里定义一个 Transaction（事务），用来度量发邮件的操作。

单击工具条上的 Transaction 开始点，定义一个名为“sendMail”的 Transaction，如图 8-10

所示。

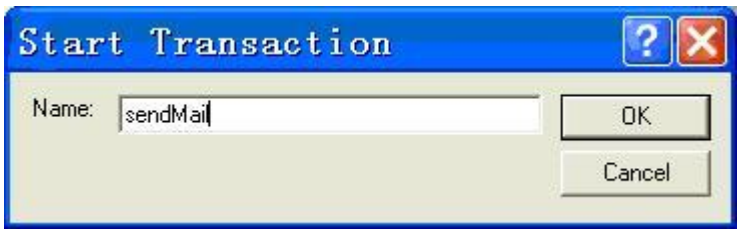


图 8-10　设置发送邮件的事务开始点

回到 Outlook 2003，继续操作。单击“新建”按钮，会弹出新邮件的窗口，在收件人、邮件标题、邮件正文中填写信息，如图 8-11 所示。

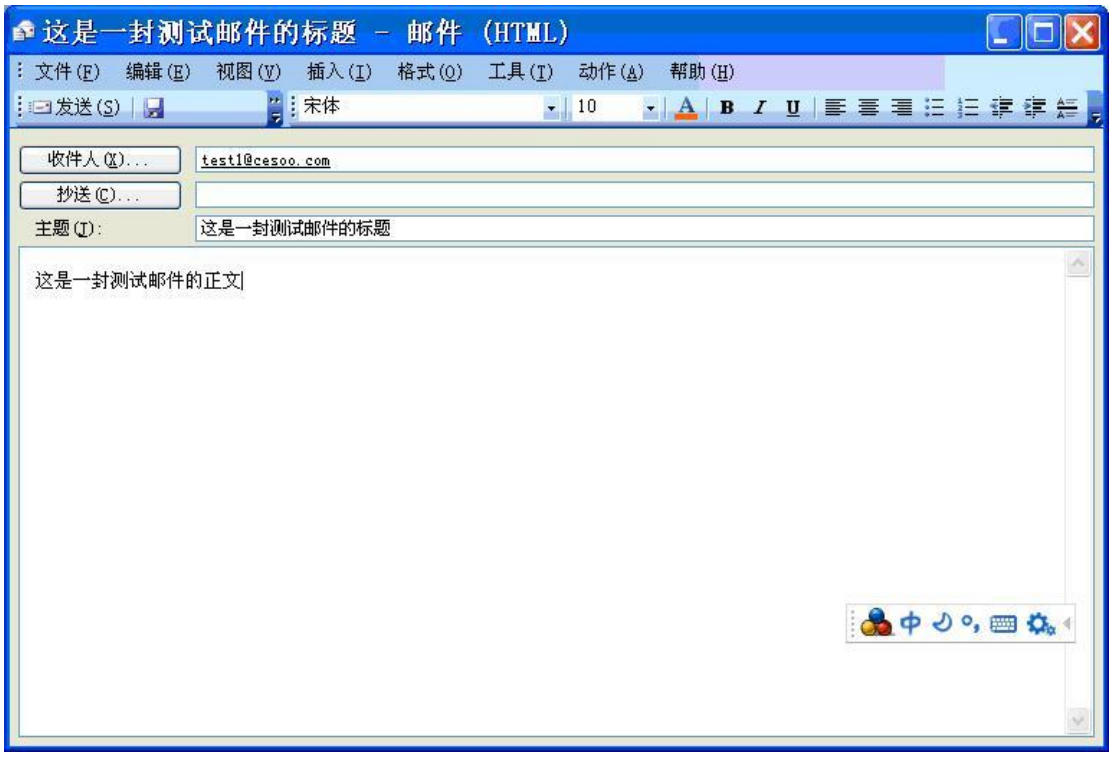


图 8-11　在 Outlook 中构建一封新邮件

单击“发送”按钮，邮件被发送。这时我们再观察 VU 录制工具条，发现如图 8-12 所示的提示。

图 8-12　Outlook 发送邮件引发的网络交互事件

67=24 次交互。–交互事件已经由之前的 67 次增长到 91 次，这说明发送邮件的操作在网络上产生了 91

单击工具条上结束 Transaction 的按钮，自动提示“sendMail”，直接单击“OK”按钮，如图 8-13 所示。

图 8-13　设置发送邮件事务的结束点

停止 VU 录制，VU 生成脚本如下：

```
Action()
{
    lr_start_transaction("sendMail");
    smtp1 = 0;
    smtp_logon_ex(&smtp1, "SmtpLogon",
        "URL=smtp://www.cesoo.com",
         "CommonName=LoadRunner User",
    LAST);
smtp_send_mail_ex(&smtp1, "SendMail",
         "To=test1@cesoo.com",
         "From=test1@cesoo.com",
```

```
                    "Subject==?utf-8?B?6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6
aKYdGVzdA==?=",
            "ContentType=multipart/alternative;",
             MAILOPTIONS,
             "X-Mailer: Microsoft Office Outlook, Build 11.0.5510",
             "Thread-Index: AciOgCIrErOOfaSTRUulBNIY7vEV0Q==",
            "X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.3028",
             MAILDATA,
            "AttachRawFile=mailnote1_01.dat",
            "AttachRawFile=mailnote1_02.dat",
          LAST);
    smtp_logout_ex(&smtp1);
        smtp_free_ex(&smtp1);
        lr_end_transaction("sendMail",LR_AUTO);
    return 0;
    }
```

8.2.2. 对 SMTP Vuser 录制生成的脚本进行分析

我们在 Outlook 上的一系列操作，包括登录和发送邮件，被 VU 转换成一系列函数，比如 smtp_logon_ex 和 smtp_send_mail_ex 等。下面我们对这些函数进行分析。

（1）在 lr_start_transaction(“send”) 语句之前未有其他语句。而在 send Mail 之前，实际上在 Outlook 之前已经完成了登录，并且在网络上有了 67 个事件交互。显然，SMTP Vuser 对这些网络交互事件并不感兴趣，因为它们并没有转化成脚本。这只能说明一个事实：Outlook 登录操作的底层走的网络协议并不是 SMTP 协议 （实际上是 IMAP 协议）。

（2）smtp_logon_ex 是 SMTP Vuser 提供的一个函数，实现的是 SMTP 的登录认证。

（3）smtp_send_mail_ex 是 SMTP Vuser 的一个最主要的功能，其作用是将一封邮件发送到指定的 E-mail 地址。

“From=test1@cesoo.com”是设置发件人地址。

“To=test1@cesoo.com”是设置收件人地址。

“Subject==?utf-8?B?6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6aKYdGVzdA==?=”是设置邮件的标题（subject）。

从上一节的 SMTP 协议规范我们知道，SMTP 网关一直保持 7 位 ASCII 码的“古老传统”，因此任何多字节的数据在经过 SMTP 协议之前必须 要经过编码。在这里，“?utf-8?”说明 Oultlook 2003 用的是 utf-8 编码方式，而“6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6aKYdGVzdA”则是 “这是一封测试邮件的标题”的编码后字节表现形式。

（4）“AttachRawFile=mailnote1_01.dat”和“AttachRawFile=mailnote1_ 02.dat”，则是邮件正文的传送方式。检查录制后的脚本，发现左侧导航栏中多了“mailnote1_01.dat”和“mailnote1_02.dat”两个节点，如图 8-14 所示。

点击打开 mailnote1_01.dat，我们看到如下数据：

Content-Type: text/plain;

charset="utf-8"

Content-Transfer-Encoding: base64

6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu255qE5q2j5paHdGVzdA0K

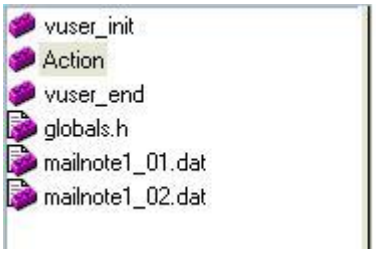


图 8-14 SMTP 虚拟用户脚本中的数据区

这是对邮件正文的编码，同样的 utf-8 编码，“6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu255qE5q2j5paHdGVzdA0K”则是“这是一封测试邮件的正文”的 utf 编码后字节表现形式。

点击打开 mailnote1_02.dat，我们看到如下数据：

Content-Type: text/html;

charset="utf-8"

Content-Transfer-Encoding: quoted-printable

=EF=BB=BF<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN">

<HTML><HEAD>

<META http-equiv=3DContent-Type content=3D"text/html; charset= 3Dutf-8">

<META content=3D"MSHTML 6.00.2900.3157" name=3DGENERATOR></HEAD>

<BODY>

<DIV><FONT face=3D=E5=AE=8B=E4=BD=93 =

size=3D2>=E8=BF=99=E6=98=AF=E4=B8=80=E5=B0=81=E6=B5=8B=E8=AF=95=E9=82=AE=E4=

=BB=B6=E7=9A=84=E6=AD=A3=E6=96=87test</FONT></DIV></BODY></HTML>

这些看起来都不是很陌生，和网页的 DOM 源码非常相像。没错，因为 Outlook 发送邮件的格式默认是 html 的，所以其实邮件正文里暗含了 html 的格式数据。

8.2.3. 回放 SMTP Vuser 脚本并分析网络日志

我们通过上一节的分析已经得知，SMTP Vuser 将发送邮件的动作转化成了 VU 函数，但这是不是对 SMTP 协议一个完整的模拟呢？我们知道 SMTP 协议是由一系列操作原语组成的，如 HELO、MAIL FROM、RCPT TO 等。如果回放 SMTP Vuser 脚本，同时我们在网络上能捕获到这些原语，就说明 SMTP Vuser 确实是对 SMTP 协议进行了封装和实现。

回放脚本，同时使用 Ethereal 在网络上捕捉从客户端到 cesoo 服务器 25 端口的数据。我们捕捉到网络交互如下：

发送者 接收者 数据内容

01. Intel_a3:e8:ef BroadCast Who has 192.168.1.100?

02. 1d:7d:4a:66 Intel_a3:e8:ef 192.168.1.100 is at 1d:7d:4a:66

03. 192.168.1.1 192.168.1.100 smtp syn

04. 192.168.1.100 192.168.1.1 220 server ready.

05. 192.168.1.1 192.168.1.100 HELO

06. 192.168.1.100 192.168.1.1 cesoo.com hello, glad to meet u

07. 192.168.1.1 192.168.1.100 MAIL FROM:test1@cesoo.com

08. 192.168.1.100 192.168.1.1 250 sender is OK

09. 192.168.1.1 192.168.1.100 RCPT TO:test1@cesoo.com

10. 192.168.1.100 192.168.1.1 250 receipt is OK

11. 192.168.1.1 192.168.1.100 DATA

12. 192.168.1.100 192.168.1.1 Enter mail, end with “.”

13. 192.168.1.1 192.168.1.100 Message body

14. 192.168.1.1 192.168.1.100 Message body

15. 192.168.1.1 192.168.1.100 Message body

16. 192.168.1.100 192.168.1.1 221 cesoo.com closing connect

注：以上数据经过过滤和整理。192.168.1.1 为客户端，192.168.1.100 为 Server 端，即 cesoo.com。

我们可以看到 1 和 2 是网络 ARP 协议，用来确定服务器地址。而下面的交互完全遵循

SMTP 协议规范，老老实实地按照 HELO、MAIL FROM、RCPT TO、DATA 的顺序进行对话。所以我们从这里可以看到 SMTP Vuser 确实对 SMTP 协议进行了一次完全的封装。最后在网络中传递的 message body 如下：

Message-ID: <000047e9148d$00000001$00bb6f3c@cesoo.com>

From: "LoadRunner User" <test1@cesoo.com>

Date: 星期二, 25 三月 2008 23:04:45 +08:00

Subject: =?utf-8?B?6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu25qCH6aKYdGVzdA==?=

MIME-Version: 1.0

Content-Type:multipart/alternative;

boundary="------=_NextPart_00000001_000009f8.00bb6f3c"

X-Mailer: Microsoft Office Outlook, Build 11.0.5510

Thread-Index: AciOghaui9AQdpK8TO+CcxyY5WXVPQ==

X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.3028

This is a multi-part message in MIME format.

--------=_NextPart_00000001_000009f8.00bb6f3c

Content-Type: text/plain;

charset="utf-8"

Content-Transfer-Encoding: base64

6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu255qE5q2j5paHdGVzdA0K

--------=_NextPart_00000001_000009f8.00bb6f3c

Content-Type: text/html;

charset="utf-8"

Content-Transfer-Encoding: quoted-printable

=EF=BB=BF<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional //EN">

<HTML><HEAD>

<META http-equiv=3DContent-Type content=3D"text/html; charset= 3Dutf-8">

<META content=3D"MSHTML 6.00.2900.3157" name=3DGENERATOR></HEAD>

<BODY>

<DIV><FONT face=3D=E5=AE=8B=E4=BD=93=

size=3D2>=E8=BF=99=E6=98=AF=E4=B8=80=E5=B0=81=E6=B5=8B=E8=AF=95=E9=82=AE=E4=

=BB=B6=E7=9A=84=E6=AD=A3=</FONT></DIV></BODY></HTML>

以上就是网络中 SMTP 协议实际传输的数据。最后我们在 Outlook 端看到的这封邮件，其实是由 Outlook 客户端进行解释并显示的，如图 8-15 所示。

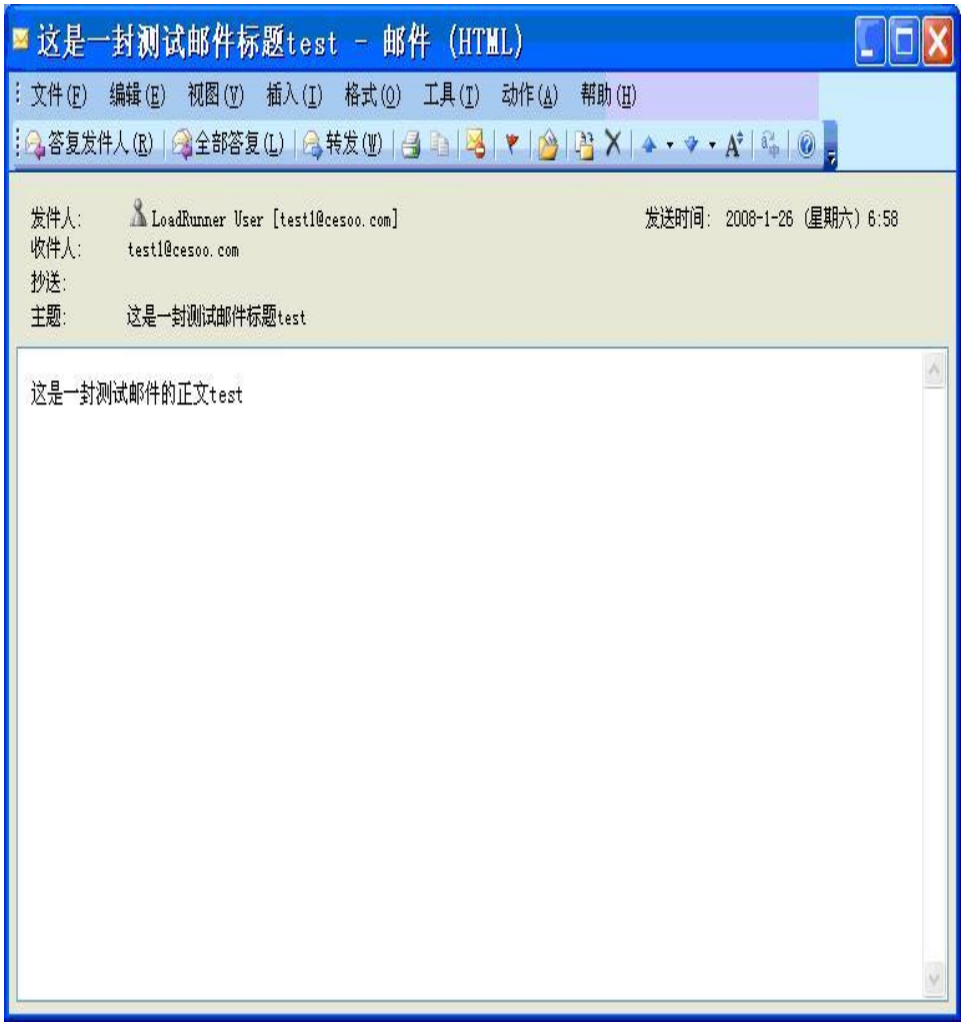


图 8-15　SMTP Vuser 成功回放脚本并发送邮件

8.3. VU Java Vuser 自开发 SMTP 程序

从上节的学习可知，VU 可以使用 SMTP Vuser 来录制生成基于 SMTP 协议的脚本，这对于刚入门的测试工程师来说非常快捷和高效，但同时也为调试脚本带来一定困难，尤其是邮件中包含一些非英文字符，在 dat 文件中都是编码后的字节，难以维护和修改。在这种情况下，如果性能测试工程师具有一定的开发经验和技术，就可以考虑使用 Java Vuser 自主开发 Java 程序来达到相同的目的。

8.3.1. VU 创建 Java 模板虚拟用户

我们首先需要创建 Java Vuser，然后在 Java Vuser 环境下加载或开发 Java 程序。

启动 VU，在“File”菜单下选择“New”，在弹出的 Vuser 列表中选择“Java Vuser”，如图

8-16 所示。

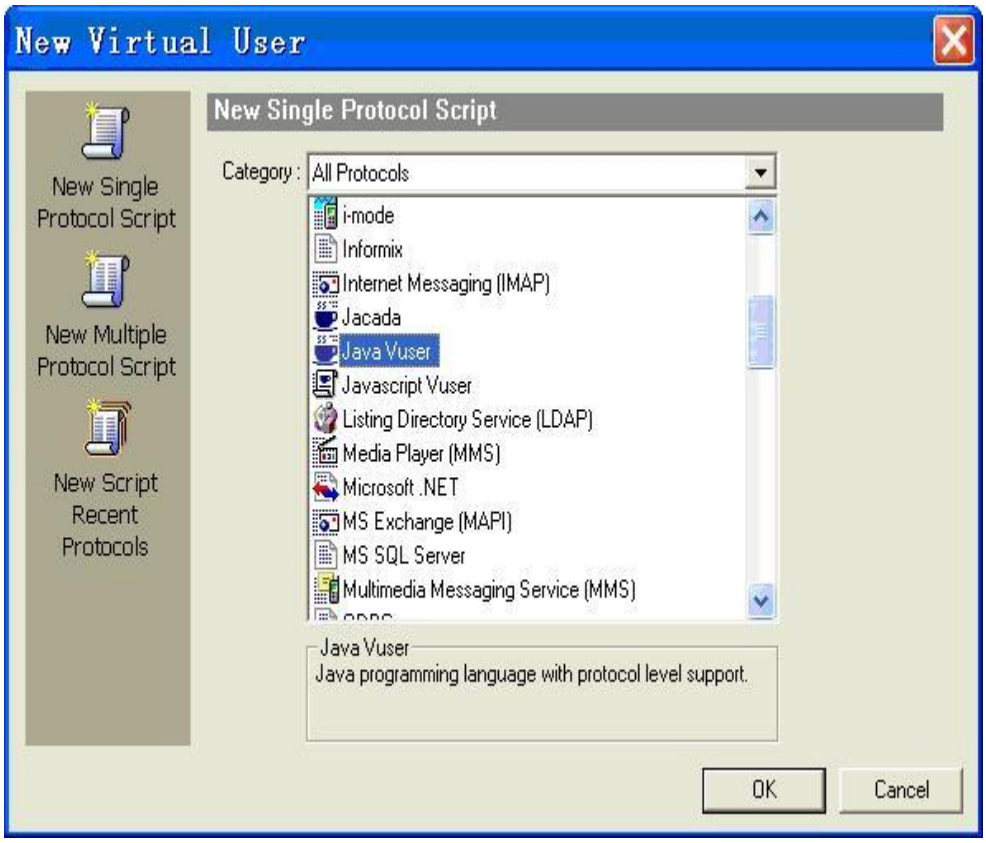


图 8-16　在虚拟用户列表中选择“Java Vuser”

单击“OK”按钮，在 VU 脚本中我们看到生成如下模板脚本：

```
import lrapi.lr;

public class Actions

{

public int init() {

    return 0;

    }//end of init

    public int action() {

    return 0;
```

```
    }//end of action

    public int end() {

    return 0;

    }//end of end

}
```

由上可见，Java Vuser 中的 Int()、action()、end()函数和 C Vuser 中的 vuser_init、action、vuser_end 一样，迭代的设置只对 action 生效，而对 init 和 end 无效。我们的开发主要在 action 函数中完成。

8.3.2. 设置 Java 环境

我们知道 Java 环境最主要的参数有两个：一个是 Java 虚拟机 path，另一个是 classpath。

打开 Vuser 的运行时设置（Run-time Settings）窗口，选择 Java VM，如图 8-17 所示。

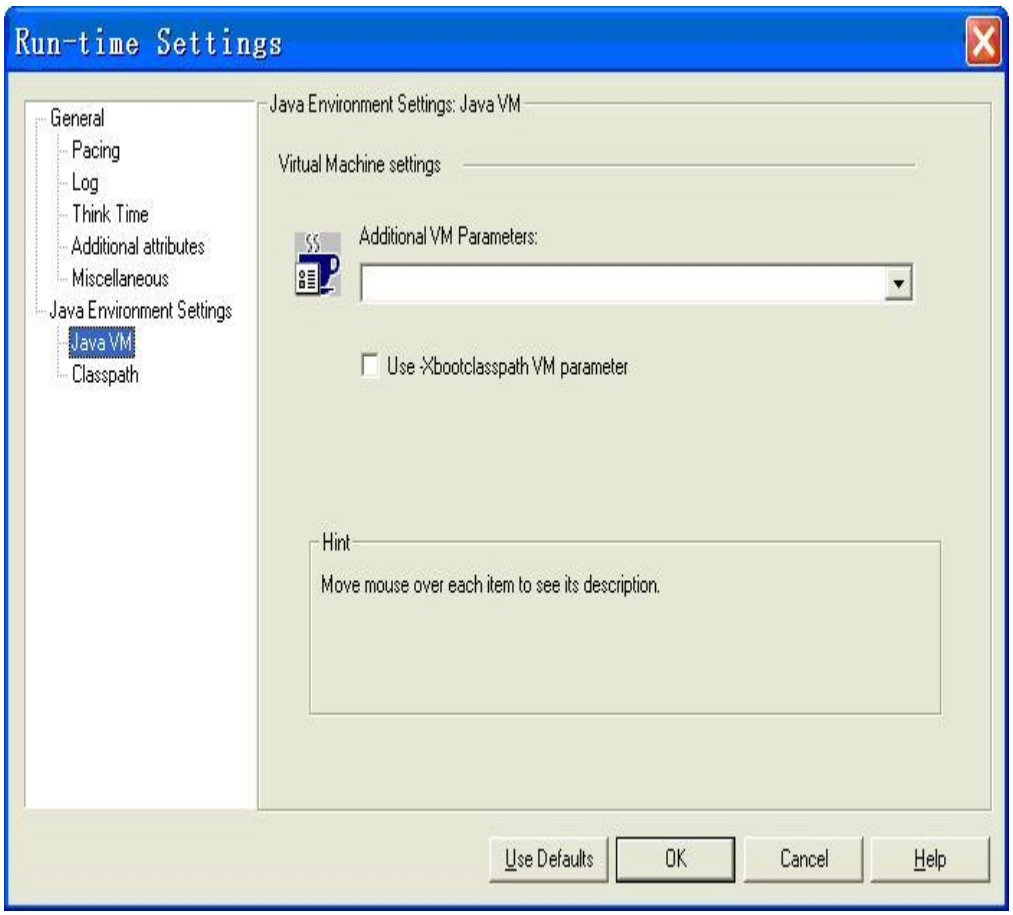


图 8-17　运行时设置中的 Java VM 配置页面

这里如果 Xbootclasspath 没有特殊的参数设置，对此页面不要做任何修改。因为 VU 将自动从 path 中获得 Java 虚拟机的路径。

单击“Classpath”，显示如图 8-18 所示的配置页面。

对于 SMTP 协议，Java 开源组织已经提供了很多相关 package 封装，我们可以在 Internet 上下载，比如在 jakarta 网站。

我们在这里添加 3 个 mail 的 jar 包，即 mail. jar、poi. jar 和 jakarta-oro-2.0.8.jar。单击“OK”按钮，关闭“Run-time Settings”窗口。

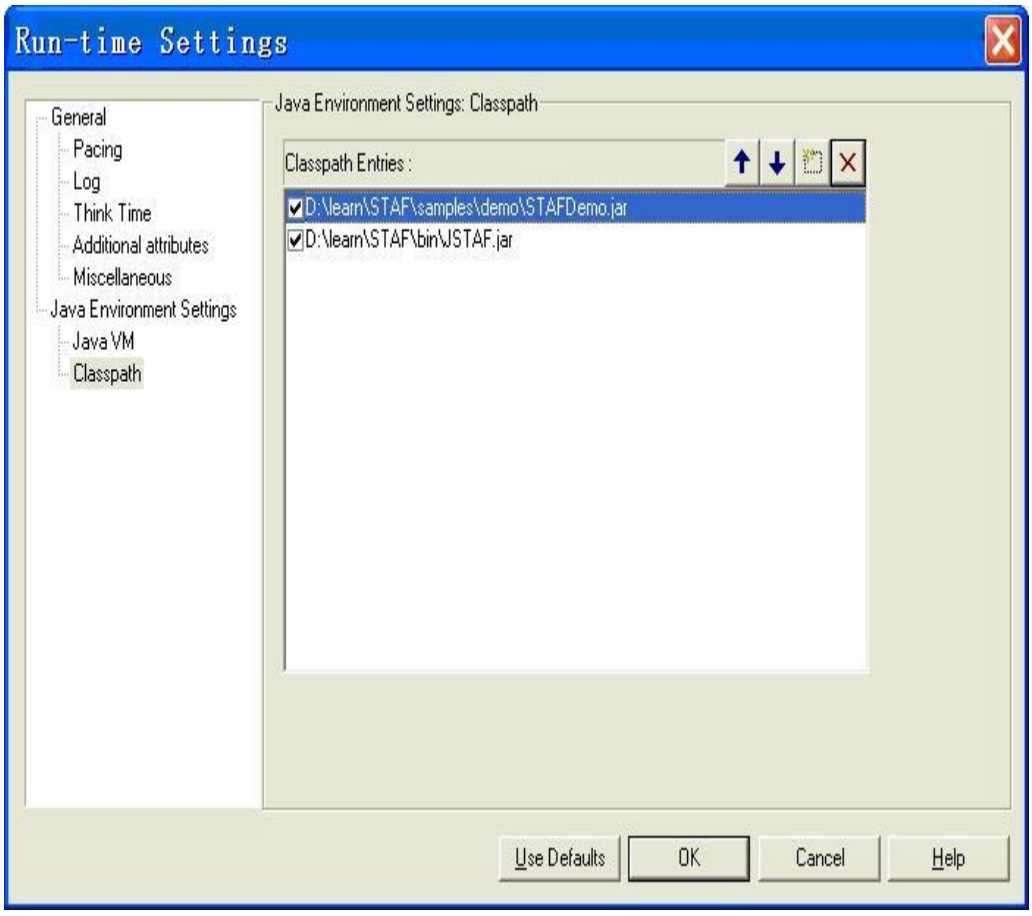


图 8-18　运行时设置中的 Java Classpath 配置页面

单击 VU 运行按钮，编译运行通过，验证我们的设置是正确的。

8.3.3. 在 Java Vuser 中开发 SMTP 发送 mail 脚本

首先我们需要在 Java 文件头进行 import 声明，包含相关 package。

```
import lrapi.lr;

import java.io.File;

import java.io.FileInputStream;

import java.util.ArrayList;

import java.util.Date;

import java.util.Properties;

import javax.mail.*;

import javax.mail.internet.*;

import org.apache.poi.hssf.usermodel.*;

import com.sun.mail.smtp.*;
```

再次运行脚本，以验证 import 是否成功，否则 Java 会报错，显示 package 找不到。

在 action 函数中编写发送邮件的 Java 代码，如下：

```
public int action() {


    boolean ssl = false;

    //获得 SMTP 环境

    Properties props = System.getProperties();

    //设置 SMTP 主机地址

    props.put("mail.smtp.host", "192.168.1.100");
```

```
//设置 SMTP 端口号

props.put("mail.smtp.port","25");

//设置 SMTP 用户名

props.put("mail.smtp.user","test1");

Session session=Session.getInstance(System.getProperties(),null);

//设置邮件 header 字段

String mailer = "send from VU java SMTP";

//设置收件人

String sendTo = "test1@cesoo.com";

//设置抄送人

String sendCC = "test1@cesoo.com";

//设置发件人

String sendFrom = "test1@cesoo.com";

try{

  //获得发送实体

  SMTPTransport t = (SMTPTransport)session.getTransport(ssl ? "smtps" : "smtp");

  //与 SMTP 主机相连

  t.connect("192.168.1.100","test1","123456");

  MimeMessage msg = new MimeMessage(session);

  //设置邮件各个字段

  String subject = "testing mail subject";
```

```
        String personalName = "LR JAVA Vuser";
        String body = "testing mail body";
        String charset = "utf-8";
        InternetAddress[] to = new InternetAddress[1];
        InternetAddress from = new
    InternetAddress(sendFrom,personalName,charset);
        to[0] =new InternetAddress(sendTo,personalName,charset);
        msg.setFrom(from);
        msg.setRecipients(Message.RecipientType.TO,to);
        msg.setSubject(subject,charset);
        msg.setText(body,charset);
        msg.setSentDate(new Date());
        //发送邮件
        t.sendMessage(msg, msg.getAllRecipients());
    }
    catch (Exception e){
        e.printStackTrace();
    }
    return 0;
}//end of action
```

以上代码运行成功后，Outlook 即可收到一封邮件，如图 8-19 所示。

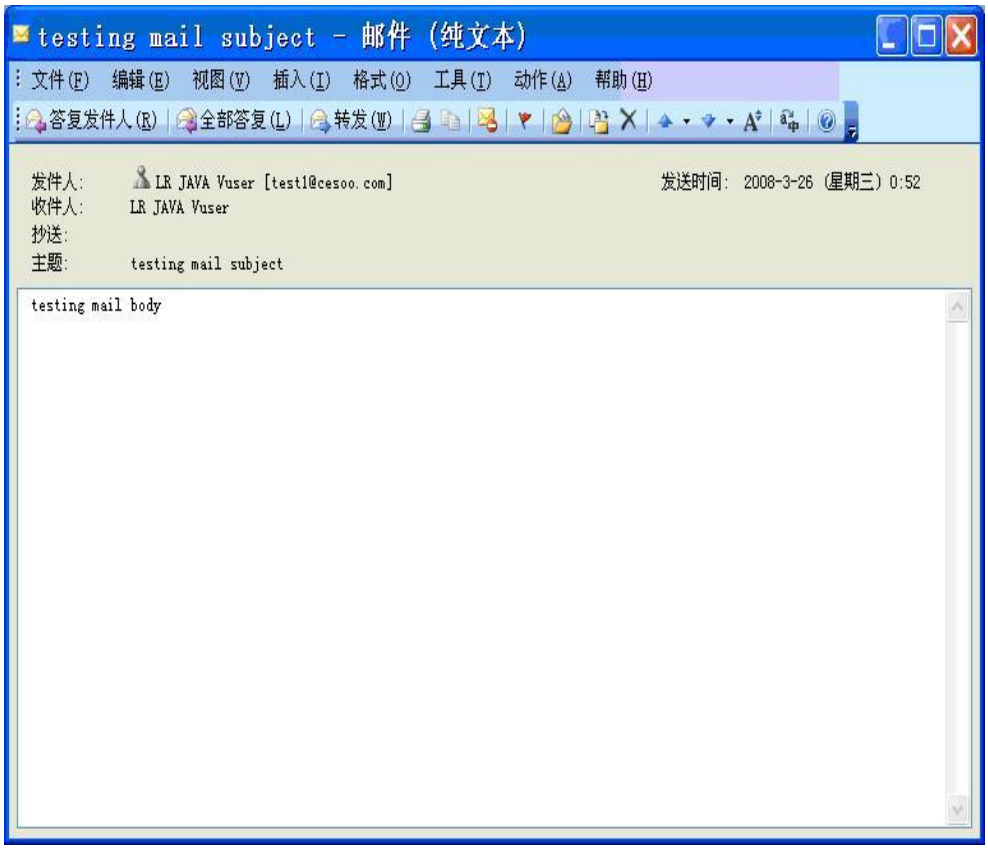


图 8-19　Java Vuser 成功回放脚本并发送邮件

8.3.4. 参数化增强脚本

以上脚本中的配置信息和邮件字段信息可使用参数化方法，转移到参数表中存储。参数化后，代码有如下变化：

```
public int action() {

boolean ssl = false;

    //获得 SMTP 环境

    Properties props = System.getProperties();

    //设置 SMTP 主机地址

    props.put("mail.smtp.host", "<SMTPHost>");

    //设置 SMTP 端口号
```

```
props.put("mail.smtp.port","<SMTPPort>");

//设置 SMTP 用户名

props.put("mail.smtp.user","<UserName>");

Session session=Session.getInstance(System.getProperties(),null);

//设置邮件 header 字段

String mailer = "<MailHeadere>";

//设置收件人

String sendTo = "<SendTo>";

//设置抄送人

String sendCC = "<sendCC>";

//设置发件人

String sendFrom = "<sendFrom>";

try{

  //获得发送实体

  SMTPTransport t = (SMTPTransport)session.getTransport(ssl ? "smtps" : "smtp");

  //与 SMTP 主机相连

  t.connect("<SMTPHost>","<UserName>","<Passwd>");

  MimeMessage msg = new MimeMessage(session);

  //设置邮件各个字段

  String subject = "<mailSubject>";

  String personalName = "<PersonalName>";
```

```
        String body = "<mailBody>";
        String charset = "utf-8";
        InternetAddress[] to = new InternetAddress[1];
        InternetAddress from = new
    InternetAddress(sendFrom,personalName,charset);
        to[0] =new InternetAddress(sendTo,personalName,charset);
        msg.setFrom(from);
        msg.setRecipients(Message.RecipientType.TO,to);
        msg.setSubject(subject,charset);
        msg.setText(body,charset);
        msg.setSentDate(new Date());
        //发送邮件
        t.sendMessage(msg, msg.getAllRecipients());
     }
    catch (Exception e){
        e.printStackTrace();
    }
return 0;
}//end of action
```

参数化后，验证脚本，能够通过。

我们就可以在参数表中更改测试数据，发送不同字符的邮件了！

中文邮件：如图 8-20 所示。

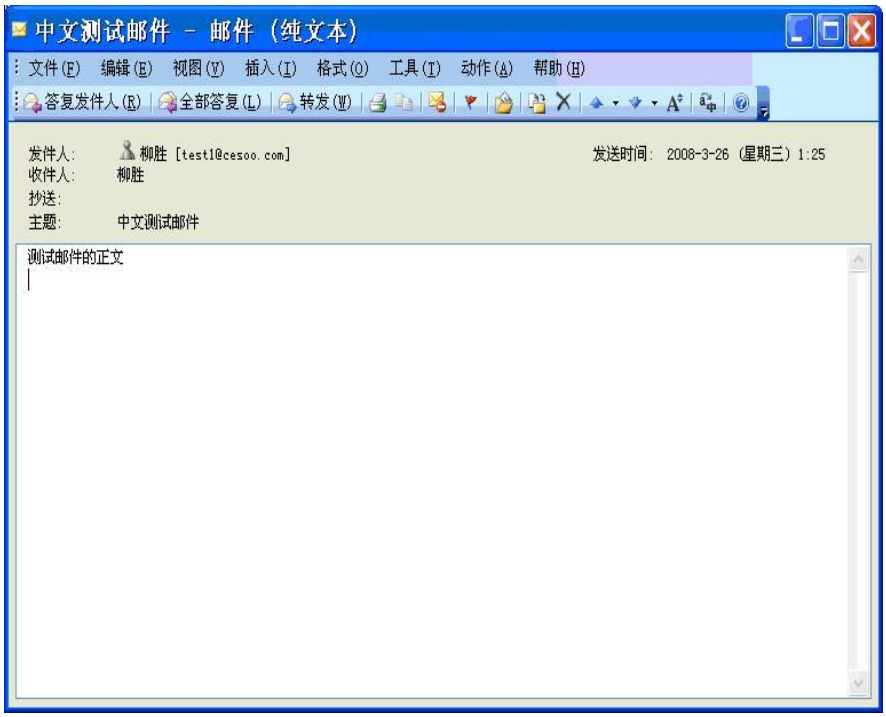


图 8-20　Java Vuser 参数化后成功发送中文邮件

日文邮件：如图 8-21 所示。

由此可见，通过 Java Vuser 的自开发程序代码，脚本的灵活性大大增强。这是录制脚本无法比拟的优势。

录制和开发是 VU 生成脚本的两种方式，我们应该根据项目需求、时间、资源等综合因素考虑选择最合适的方式。

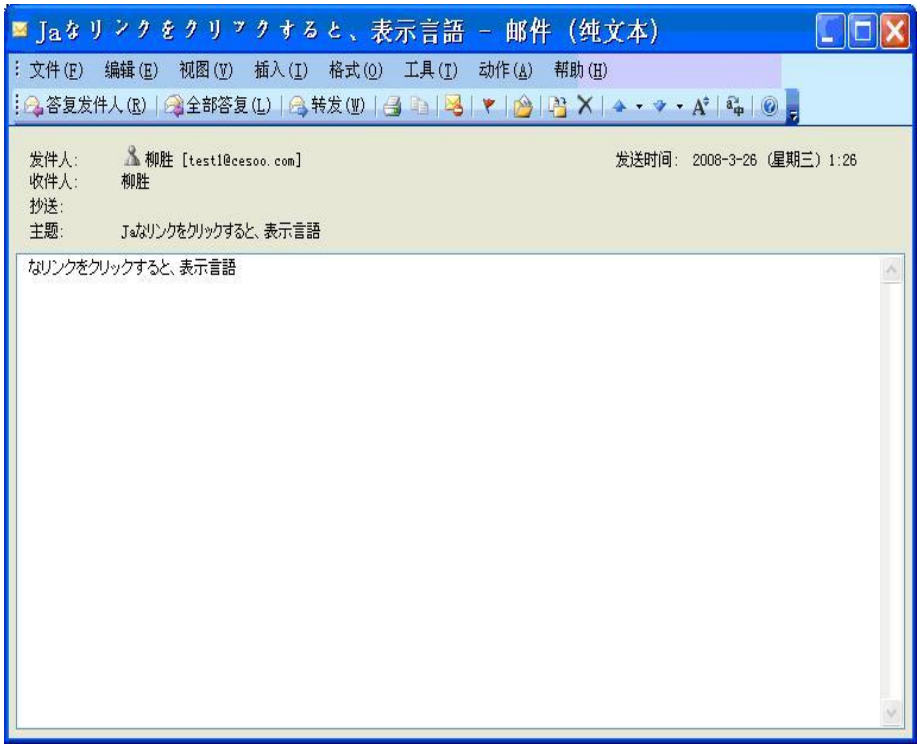


图 8-21　Java Vuser 参数化后成功发送日文邮件

9. 学以致用——一步一步做 Web 系统性能测试

9.1 软件系统背景及架构设计介绍

9.2 性能要求和性能指标分析(Goal 阶段)

9.2.1 性能指标

9.2.2 业务模型分析和需求细化

9.3 性能测试方案和用例设计(Analysis 阶段)

9.4 性能测试各种度量的建立(Metrics 阶段)

9.4.1 性能脚本的生成

9.4.2 定义用户行为