**loadrunner总结**

**v1.0**

[版本变更 3](#_Toc433006800)

[1 webservice、socket、http 协议小记 4](#_Toc433006801)

[1.1 socket协议定义和其特点 4](#_Toc433006802)

[1.2 http协议定义和其特点 4](#_Toc433006803)

[2 LR基础知识 5](#_Toc433006804)

[2.1 HTML-based script与URL-based script脚本区别 5](#_Toc433006805)

[2.2 运行时的Pacing设置主要影响什么 6](#_Toc433006806)

[2.3 Enable logging 6](#_Toc433006807)

[2.4 VuGen会修改录制浏览器中的代理服务器设置吗 6](#_Toc433006808)

[2.5 深入理解Browser Emulation 6](#_Toc433006809)

[2.6 迭代大于2次失败 13](#_Toc433006810)

[2.7 Run-time Setting中线程和进程运行方式的区别 13](#_Toc433006811)

[2.8 处理Controller常规超时 13](#_Toc433006812)

[2.9 服务器压力过大超时。 14](#_Toc433006813)

[2.10 LoadRunner HTTP服务器状态代码：404/500 14](#_Toc433006814)

[2.11 录制Web协议脚本回放停止 14](#_Toc433006815)

[2.12 模拟的用户数量却上不去-27796 15](#_Toc433006816)

[2.13 造数据的方式 15](#_Toc433006817)

[2.14 录制脚本中包含中文，出现乱码 16](#_Toc433006818)

[2.15 回放脚本时，在浏览器显示的中文是乱码 16](#_Toc433006819)

[2.16 Analysis设置思考时间 17](#_Toc433006820)

[2.17 RTMP协议用 flex协议录制 17](#_Toc433006821)

[2.18 修改LR自带的例子端口号 17](#_Toc433006822)

[2.19 用Web\_reg\_find查找中文字符串时查找不到 18](#_Toc433006823)

[2.20 Web\_find失效 18](#_Toc433006824)

[2.21 如何从命令行调用LoadRunner脚本 18](#_Toc433006825)

[2.22 LR启动controller报错“transaction monitor not available” 18](#_Toc433006826)

[2.23 Exception ACCESS\_VIOLATION received 18](#_Toc433006827)

[2.24 如果C代码函数在LR中，部分类型不读取 19](#_Toc433006828)

[2.25 LoadRunner监控远程机器Windows资源，提示“net use failed” 19](#_Toc433006829)

[3 关联函数 20](#_Toc433006830)

[3.1 抓取所有相同左右边界的动态value 20](#_Toc433006831)

[4 参数化 21](#_Toc433006832)

[5 Socket协议 22](#_Toc433006833)

[5.1 如何忽略Socket接收数据的验证 22](#_Toc433006834)

[6 WebServices协议 22](#_Toc433006835)

[6.1 web\_service\_call（） 22](#_Toc433006836)

[6.2 soap\_request（） 24](#_Toc433006837)

[7 LoadRunner性能测试时遇到的问题汇总 27](#_Toc433006838)

[7.1 Controller提示信息 27](#_Toc433006839)

[7.2 脚本为空的解决方法： 30](#_Toc433006840)

[7.3 录制脚本时为什么不弹出IE浏览器 30](#_Toc433006841)

[8 常见的性能问题与分析 30](#_Toc433006842)

[8.1 常见问题 30](#_Toc433006843)

[8.2 基础问题检查 31](#_Toc433006844)

[9 性能测试方法与实践 33](#_Toc433006845)

[9.1 性能测试目的 33](#_Toc433006846)

[9.2 性能测试类型 33](#_Toc433006847)

[9.3](#_Toc433006848) **[性能测试基本过程](#_Toc433006848)** [39](#_Toc433006848)

[9.4 性能评估模型 41](#_Toc433006849)

[9.4.1 性能评估模型概述 41](#_Toc433006850)

[9.4.2 闭环流程图 45](#_Toc433006851)

[9.4.3 TPS计算公式 46](#_Toc433006852)

[9.4.4 影响因子 47](#_Toc433006853)

[9.4.5 响应时间 48](#_Toc433006854)

[9.4.6 稳定性要求 48](#_Toc433006855)

[9.4.7 性能测试环境要求 48](#_Toc433006856)

[9.4.8 并发量与吞吐量（TPS）的关系 49](#_Toc433006857)

[9.4.9 线上性能数据采集 49](#_Toc433006858)

[9.5 性能测试需求分析 50](#_Toc433006859)

[9.5.1 业务调研 50](#_Toc433006860)

[9.5.2 性能需求评估 51](#_Toc433006861)

[9.5.3 性能测试指标 53](#_Toc433006862)

[9.5.4 压力与容量预估 57](#_Toc433006863)

[9.6 性能测试准备 58](#_Toc433006864)

[9.6.1 性能测试环境申请 58](#_Toc433006865)

[9.6.2 环境清理 59](#_Toc433006866)

[9.6.3 环境搭建及数据准备 59](#_Toc433006867)

[9.6.4 压力工具选择 59](#_Toc433006868)

[9.6.5 资源监控工具部署 62](#_Toc433006869)

[9.7 性能测试执行 62](#_Toc433006870)

[9.7.1 无人值守执行性能测试 62](#_Toc433006871)

[9.7.2 动态调优 63](#_Toc433006872)

[9.7.3 边执行边思考 63](#_Toc433006873)

[9.8 性能分析与调优 64](#_Toc433006874)

[9.9 性能测试报告与总结 64](#_Toc433006875)

[9.10 性能准入标准与建议 65](#_Toc433006876)

[10 性能分析与调优 67](#_Toc433006877)

[10.1 性能分析 67](#_Toc433006878)

[10.1.1 性能分析的目的 67](#_Toc433006879)

[10.1.2 常见性能瓶颈征兆 67](#_Toc433006880)

[10.1.3 性能数据解读建议 68](#_Toc433006881)

[10.1.4 如何定位性能问题 70](#_Toc433006882)

[10.1.5 常见性能问题参考 72](#_Toc433006883)

[10.2 性能调优 73](#_Toc433006884)

[10.2.1 性能调优目标与策略 73](#_Toc433006885)

[10.2.2 调优的原则 74](#_Toc433006886)

[10.2.3 调优的基本步骤 74](#_Toc433006887)

[10.2.4 性能调优的层次 75](#_Toc433006888)

[10.2.5 配置项性能调优参考 87](#_Toc433006889)

[11 性能测试模板 93](#_Toc433006890)

[11.1 COD系统性能测试报告范例 93](#_Toc433006891)

[12 性能测试工具 110](#_Toc433006899)

[12.1 JMeter参考手册 110](#_Toc433006900)

[12.1.1 JMeter介绍 110](#_Toc433006901)

[12.1.2 JMeter安装 111](#_Toc433006902)

[12.1.3 JMeter运行 111](#_Toc433006903)

[12.1.4 JMeter常用协议 113](#_Toc433006904)

[12.1.5 JMeter常用场景 117](#_Toc433006905)

[12.1.6 JMete主要组件介绍 121](#_Toc433006906)

[12.1.7 JMeter函数 127](#_Toc433006907)

[12.1.8 JMeter扩展 128](#_Toc433006908)

[12.1.9 JMeter持续集成 131](#_Toc433006909)

[12.1.10 JMeter+Selenium 135](#_Toc433006910)

[12.2 Loadrunner 139](#_Toc433006911)

[12.2.1 loadrunner简介 139](#_Toc433006912)

[12.2.2 录制及调试脚本 140](#_Toc433006913)

[12.2.3 场景设置 154](#_Toc433006914)

[12.2.4 结果分析 157](#_Toc433006915)

[13 参考文档 159](#_Toc433006916)

版本变更

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **要点** | **修改人** |
| 2015-10-12 | 1.0 | loadrunner总结 |  |
|  |  |  |  |
|  |  |  |  |

# webservice、socket、http 协议小记

webservice是基于http协议的soap协议的封装和补充，如果你考虑公共稳定的接口，可以考虑一下。

## socket协议定义和其特点

1 数据传输方式  
1.1 socket传输的定义和其特点  
    所谓socket通常也称作"套接字",实现服务器和客户端之间的物理连接，并进行数据传输，主要有udp和tcp两个协议。socket处于网络协议的传输层。  
    udp协议：广播式数据传输，不进行数据验证  
    tcp协议：传输控制协议，一种面向连接的协议，给用户进程提供可靠的全双工的字节流，  
  
**socket传输的特点：**  
  **优点**  
   1) 传输数据为字节级，传输数据可自定义，数据量小（对于手机应用讲：费用低）  
   2) 传输数据时间短，性能高  
   3) 适合于客户端和服务器端之间信息实时交互  
   4) 可以加密,数据安全性强  
**缺点：**  
   1) 需对传输的数据进行解析，转化成应用级的数据  
   2) 对开发人员的开发水平要求高  
   3) 相对于http协议传输，增加了开发量

## http协议定义和其特点

1.2 基于http协议传输的定义和其特点

   目前基于http协议传输的主要有http协议 和基于http协议的soap协议（web service）,常见的方式是 http 的post 和get 请求，web 服务。

**基于http协议的传输特点**  
  **优点：**  
   1) 基于应用级的接口使用方便  
   2) 程序员开发水平要求不高，容错性强  
**缺点：**  
   1) 传输速度慢，数据包大（http协议中包含辅助应用信息）

   2) 如实时交互，服务器性能压力大。

   3) 数据传输安全性差

2 各数据传输方式的适用范围  
 2.1 socket传输适用范围  
  基于socket传输的特点:socket 传输方式适合于对传输速度，安全性，实时交互，费用等要求高的应用中，如网络游戏，手机应用，银行内部交互等  
 2.2 基于http协议传输的适用范围  
  基于http协议传输的特点：基于http协议传输方式适合于对传输速度，安全性，要求不是很高，且需要快速开发的应用。如公司oa系统，互联网服务等

# LR基础知识

## HTML-based script与URL-based script脚本区别

使用“HTML-based script”的模式录制脚本，VuGen为用户的每个HTML操作生成单独的步骤，这种脚本看上去比较直观;

使用“URL-based script”模式录制脚本时，VuGen可以捕获所有作为用户操作结果而发送到服务器的HTTP请求，然后为用户的每个请求分别生成对应方法。

通常，基于浏览器的Web应 用会使用“HTML-based script”模式来录制脚本;而没有基于浏览器的Web应用、Web应用中包含了与服务器进行交互的Java Applet、基于浏览器的应用中包含了向服务器进行通信的JavaScript/VBScript代码、基于浏览器的应用中使用了HTTPS安全协议， 这时使用“URL-based script”模式进行录制。

## 运行时的Pacing设置主要影响什么

    Pacing主要用来设置重复迭代脚本的间隔时间。共有三种方法：A:上次迭代结束后立刻开始、 B:上次迭代结束后等待固定时间、C:按固定或随机的时间间隔开始执行新的迭代。

     根据实际需要设置迭代即可。通常，没有时间间隔会产生更大的压力。

## Enable logging

 运行时设置Log标签中，如果没有勾选“Enable logging”，则手工消息可以发送吗?

Enable logging选项仅影响自动日志记录和通过lr\_log\_message发送的消息。即使没有勾选，虚拟用户脚本中如果使用lr\_message、lr\_output\_message、lr\_error\_message,仍然会记录其发出的消息。

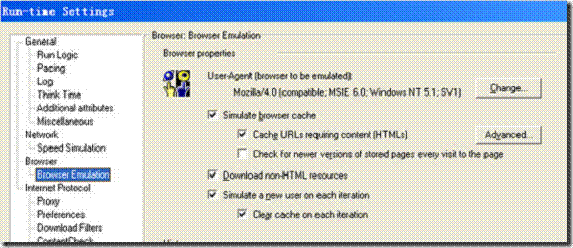
## VuGen会修改录制浏览器中的代理服务器设置吗

会修改。在开始录制基于浏览器的Web Vuser脚 本时，VuGen首先会启动指定的浏览器。然后，VuGen会指示浏览器访问VuGen代理服务器。为此，VuGen会修改录制浏览器上的代理服务器设 置。默认情况下，VuGen会立即将代理服务器设置更改为Localhost:7777。录制之后，VuGen会将原始代理服务器设置还原到该录制浏览器 中。因此，在VuGen进行录制的过程中，不可以更改代理服务器设置，否则将无法正常进行。

## 深入理解Browser Emulation

**一：基本介绍**

在Loadrunner的使用中，对于Run-time Settings下的browser emulation设置是比较容易让人产生困惑的地方。下面我们结合sniffer来具体看看每个选项的用途，以及对测试的影响。

[](http://www.blogjava.net/images/blogjava_net/tacy/WindowsLiveWriter/LoadrunnerBrowserEmulation_461/clip_image002_2.gif)

Browser Emulation 图

**二：案例和工具**

**1. 测试案例：**

打开网站首页两次，对比不同Browser Emulation设置下loadrunner的行为，脚本如下。

Action()

{

web\_url("www.primeton.com",

"URL=http://www.primeton.com/",

"Resource=0",

"RecContentType=text/html",

"Referer=",

"Snapshot=t2.inf",

"Mode=HTML",

LAST);

web\_url("www.primeton.com",

"URL=http://www.primeton.com/",

"Resource=0",

"RecContentType=text/html",

"Referer=",

"Snapshot=t2.inf",

"Mode=HTML",

LAST);

return 0;

}

**2. sniffer工具**

开源工具：Wireshark(前身是ethereal)（www.wireshark.org）

**三：测试过程**

为了方便描述，我们约定用：

*A代表Simulate browser cache*

*B代表Cache URLs requiring content(HTMLs)*

*C代表Check for newer versions of stored pages every visit to the page*

*D代表Download non-HTML resources*

*E代表Simulate a new user on each iteratioin*

*F代表Clear cache on each iteration*

首先设置Run Logic中的iteration为2。让Action运行两次，看看循环运行脚本两次，数据包和连接数的变化。

**1. 去掉所有选项**

结果：共获取数据包95个，建立连接1个（红色标识），断开连接1个（蓝色标识）

No. Time Source Destination Protocol Info

1 0.000000 192.168.1.61 203.81.29.137 TCP 13835 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

2 0.036053 203.81.29.137 192.168.1.61 TCP http > 13835 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

92 1.415887 192.168.1.61 203.81.29.137 TCP 13835 > http [FIN, ACK] Seq=817 Ack=71762 Win=257760 Len=0

94 1.449960 203.81.29.137 192.168.1.61 TCP http > 13835 [FIN, ACK] Seq=71762 Ack=818 Win=16464 Len=0

在这种情况下，数据包非常少（没有选择下载资源文件入css,js,gif等），而且你可以看到，打开4次首页，只建立了一个tcp连接。

这时，你即使选择A，发现数据包的数量量页没有变化，因为cache主要还是针对资源文件

**2. 选择E(F)**

结果：共获取数据包102个，建立连接2个（红色标识），断开连接2个（蓝色标识）

No. Time Source Destination Protocol Info

1 0.000000 192.168.1.61 203.81.29.137 TCP 13886 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

2 0.037013 203.81.29.137 192.168.1.61 TCP http > 13886 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

48 0.618117 192.168.1.61 203.81.29.137 TCP 13886 > http [FIN, ACK] Seq=409 Ack=35882 Win=257760 Len=0

49 0.644106 192.168.1.61 203.81.29.137 TCP 13887 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

51 0.651919 203.81.29.137 192.168.1.61 TCP http > 13886 [FIN, ACK] Seq=35882 Ack=410 Win=16872 Len=0

53 0.676377 203.81.29.137 192.168.1.61 TCP http > 13887 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

99 1.310379 192.168.1.61 203.81.29.137 TCP 13887 > http [FIN, ACK] Seq=409 Ack=35882 Win=257760 Len=0

101 1.347949 203.81.29.137 192.168.1.61 TCP http > 13887 [FIN, ACK] Seq=35882 Ack=410 Win=16872 Len=0

在这种情况下，数据包非常少（没有选择下载资源文件入css,js,gif等），对比第一种情况，你会发现它建立了两个连接，这就是E的作用，它对于每次迭代都当成一个新的用户，需要重新建立连接。

**3. 选择DE(F)**

结果：共获取数据包1782个，建立连接6个（红色标识），断开连接6个（蓝色标识）

No. Time Source Destination Protocol Info

1 0.000000 192.168.1.61 203.81.29.137 TCP 14016 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

2 0.037911 203.81.29.137 192.168.1.61 TCP http > 14016 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

6 0.107432 192.168.1.61 203.81.29.137 TCP 14017 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

9 0.141816 203.81.29.137 192.168.1.61 TCP http > 14017 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

426 3.334889 192.168.1.61 203.81.29.137 TCP 14017 > http [FIN, ACK] Seq=1852 Ack=150284 Win=257484 Len=0

428 3.372253 203.81.29.137 192.168.1.61 TCP http > 14017 [FIN, ACK] Seq=150284 Ack=1853 Win=16998 Len=0

448 4.395488 192.168.1.61 203.81.29.137 TCP 14020 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

457 4.439604 203.81.29.137 192.168.1.61 TCP http > 14020 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

859 7.593610 192.168.1.61 203.81.29.137 TCP 14016 > http [FIN, ACK] Seq=2849 Ack=377404 Win=257484 Len=0

870 7.659680 203.81.29.137 192.168.1.61 TCP http > 14016 [FIN, ACK] Seq=377404 Ack=2850 Win=15935 Len=0

888 8.511308 192.168.1.61 203.81.29.137 TCP 14020 > http [FIN, ACK] Seq=1602 Ack=208150 Win=257760 Len=0

890 8.549451 203.81.29.137 192.168.1.61 TCP http > 14020 [FIN, ACK] Seq=208150 Ack=1603 Win=17280 Len=0

892 8.566246 192.168.1.61 203.81.29.137 TCP 14022 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

893 8.601893 203.81.29.137 192.168.1.61 TCP http > 14022 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

899 8.702628 192.168.1.61 203.81.29.137 TCP 14023 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

904 8.741807 203.81.29.137 192.168.1.61 TCP http > 14023 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

1298 11.809456 192.168.1.61 203.81.29.137 TCP 14022 > http [FIN, ACK] Seq=1550 Ack=159770 Win=257484 Len=0

1310 11.878665 203.81.29.137 192.168.1.61 TCP http > 14022 [FIN, ACK] Seq=159770 Ack=1551 Win=17280 Len=0

1341 12.771707 192.168.1.61 203.81.29.137 TCP 14026 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

1348 12.813950 203.81.29.137 192.168.1.61 TCP http > 14026 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

1759 16.032952 192.168.1.61 203.81.29.137 TCP 14023 > http [FIN, ACK] Seq=3151 Ack=367918 Win=257484 Len=0

1761 16.068296 203.81.29.137 192.168.1.61 TCP http > 14023 [FIN, ACK] Seq=367918 Ack=3152 Win=17280 Len=0

1779 16.983042 192.168.1.61 203.81.29.137 TCP 14026 > http [FIN, ACK] Seq=1602 Ack=208150 Win=257760 Len=0

1781 17.016836 203.81.29.137 192.168.1.61 TCP http > 14026 [FIN, ACK] Seq=208150 Ack=1603 Win=17280 Len=0

在这种情况下，数据包的数量非常大，连接也很多，由于没有cache功能，每次打开页面都需要重新下载所有的资源文件。

**4. 选择ADE**

结果：共获取数据包525个，建立连接3个，断开连接3个(不再标识了，syn即为连接请求，fin即为断开请求）

No. Time Source Destination Protocol Info

1 0.000000 192.168.1.61 203.81.29.137 TCP 14189 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

2 0.033657 203.81.29.137 192.168.1.61 TCP http > 14189 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

6 0.100636 192.168.1.61 203.81.29.137 TCP 14190 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

9 0.133703 203.81.29.137 192.168.1.61 TCP http > 14190 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

429 3.383748 192.168.1.61 203.81.29.137 TCP 14190 > http [FIN, ACK] Seq=1852 Ack=150284 Win=257484 Len=0

431 3.418556 203.81.29.137 192.168.1.61 TCP http > 14190 [FIN, ACK] Seq=150284 Ack=1853 Win=16998 Len=0

471 4.352071 192.168.1.61 203.81.29.137 TCP 14189 > http [FIN, ACK] Seq=1504 Ack=235576 Win=257760 Len=0

472 4.380312 192.168.1.61 203.81.29.137 TCP 14192 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

474 4.389778 203.81.29.137 192.168.1.61 TCP http > 14189 [FIN, ACK] Seq=235576 Ack=1505 Win=17280 Len=0

476 4.413220 203.81.29.137 192.168.1.61 TCP http > 14192 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

522 5.078068 192.168.1.61 203.81.29.137 TCP 14192 > http [FIN, ACK] Seq=409 Ack=35882 Win=257760 Len=0

524 5.115099 203.81.29.137 192.168.1.61 TCP http > 14192 [FIN, ACK] Seq=35882 Ack=410 Win=16872 Len=0

在这种情况下，cache发挥作用，数据包对比第三种情况大大减少，几乎等于打开一次首页的数据量（449个数据包），只有第一次打开页面需要完整下载页面（包括资源文件），后面的三次打开页面都只要下载HTML页面（不包括资源文件）。

**5. 选择ADEF**

选择F之后我们看看结果：共获取数据包942个，建立连接4个，断开连接4个

No. Time Source Destination Protocol Info

1 0.000000 192.168.1.61 203.81.29.137 TCP 14292 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

2 0.034524 203.81.29.137 192.168.1.61 TCP http > 14292 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

6 0.102314 192.168.1.61 203.81.29.137 TCP 14294 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

9 0.139752 203.81.29.137 192.168.1.61 TCP http > 14294 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

426 3.791111 192.168.1.61 203.81.29.137 TCP 14294 > http [FIN, ACK] Seq=1852 Ack=150284 Win=257484 Len=0

428 3.824970 203.81.29.137 192.168.1.61 TCP http > 14294 [FIN, ACK] Seq=150284 Ack=1853 Win=16998 Len=0

468 6.213276 192.168.1.61 203.81.29.137 TCP 14292 > http [FIN, ACK] Seq=1504 Ack=235576 Win=257760 Len=0

469 6.244052 192.168.1.61 203.81.29.137 TCP 14297 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

471 6.249564 203.81.29.137 192.168.1.61 TCP http > 14292 [FIN, ACK] Seq=235576 Ack=1505 Win=17280 Len=0

473 6.279647 203.81.29.137 192.168.1.61 TCP http > 14297 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

479 6.374967 192.168.1.61 203.81.29.137 TCP 14298 > http [SYN] Seq=0 Len=0 MSS=1460 WS=2

484 6.419597 203.81.29.137 192.168.1.61 TCP http > 14298 [SYN, ACK] Seq=0 Ack=1 Win=17280 Len=0 MSS=1440 WS=0

897 9.858493 192.168.1.61 203.81.29.137 TCP 14297 > http [FIN, ACK] Seq=1550 Ack=159770 Win=257484 Len=0

899 9.895188 203.81.29.137 192.168.1.61 TCP http > 14297 [FIN, ACK] Seq=159770 Ack=1551 Win=17280 Len=0

939 12.840029 192.168.1.61 203.81.29.137 TCP 14298 > http [FIN, ACK] Seq=1806 Ack=226090 Win=257760 Len=0

941 12.876120 203.81.29.137 192.168.1.61 TCP http > 14298 [FIN, ACK] Seq=226090 Ack=1807 Win=17076 Len=0

在这种情况下，由于选择了F，在迭代的时候清除了cache，所以每次迭代都需要重新下载资源文件。数据包差不多等于第三种情况的一半，约等于打开两次首页的数据量（449×2个数据包）。

**6. 关于BC选项**

**C的解释（*Check for newer versions of stored pages every visit to the page）***

C比较容易理解，类似IE设置中的每次检查，如果不设置C，LR对于已经cache的文件就不会重新向服务器请求，如果选择C，你就可以在数据包中发现很多304信息。

**B的解释（*Cache URLs requiring content(HTMLs)）***

LR对于资源文件的cache并不会真正cache在内存中或者在磁盘上，这个选项表示：对于一些需要用到的关联，校验，页面解析内容真正cache在内存中，减少客户端的重复工作。

当然如果你想把GIF也cache到内存中，你可以在Advanced中设置，选择Specify URL requiring content in addition to HTML pages，加入条目image/gif，并勾选。当Vuser运行的时候，你可以对比一下mmdrv.exe进程的内存消耗（内存占用会更多）。

**四： 结论**

通过上面的测试分析，我们大概知道了每个选项的真正含义，你需要根据你的测试目的来选择合适的设置：

1、 对于一个具体的应用测试，对于前端Web Server不可忽略，缺省设置非常合适，不需要调整（有时候需要考虑把C选上）

注意：很多人在录制脚本的时候，习惯把登入操作放到vuser\_init中，这时候缺省设置可能会抛错，建议把这类的操作都放入到action中

2、 如果你更关注后端应用服务器的性能或者说做一些架构的验证分析，那你缺省设置对于你来说就不合适了，你需要选择取消所有的设置项。

当然你也可以根据自己的具体情况做不同调整，但是一定要真正理解这些选项的具体含义才能做到不犯错误

## 迭代大于2次失败

一些Web虚拟用户脚本录制后立刻回放没有任何问题，但是当设置迭代次数大于1时，如果进行回放则只能成功迭代一次。为什么从第二次迭代开始发生错误?

这种现象多是由于在“Run-time Setting”的“Browse Emulation”的设置中，勾选了“Simulate a new user on each iteration”及其下面的选项“Clear cache on each iteration”这两个选项的含义是每次迭代时模拟一个新的用户及每次迭代时清除缓存。

由于脚本迭代时，init和end只能执行一次，如果每次迭代都模拟一个新的用户并清除缓存，则用户登录信息将一并清除，因此迭代时可能会发生错误。

## Run-time Setting中线程和进程运行方式的区别

如果选择“Run Vuser as a process”，则场景运行时会为每一个虚拟用户创建一个进程;选择“Run Vuser as a thread”则将每个虚拟用户作为一个线程来运行，在任务管理器中只看到一个mmdrv.exe，这种方式的运行效率更高，能造成更大的压力，时默认选项。

另外，如果启用了IP欺骗功能，则先在Controller中选中Tools菜单下的“Expert Mode”，然后将Tools菜单下的“Options>General”标签页中的IP地址分配方式也设置为与Vuser运行方式一致，同为线程或进程方式。

## 处理Controller常规超时

在Controller中运行Web相关测试场景时，经常会有很多超时错误提示，如何处理这类问题?

这主要有脚本的默认超时设置引起。当回放Web脚本时，有时候由于服务器响应时间较长，会产生超时的错误。这时需要修改脚本的运行时配置。

vuser\_init.c(12): Error -27728: Step download timeout (120 seconds) has expired when **downloading non-resource(s)**（出现个别，可以忽略）

vuser\_init.c(12): Error -27727: Step download timeout (120 seconds) has expired when **downloading resource(s).** Set the "Step Timeout caused by resources is a warning" Run-Time Setting to Yes/No to have this message as a warning/error, respectively

vuser\_init.c(12): Error -27796

1、如果觉得下载一个页面超过2分钟不是错误的话，可以在Run-Time设置中选择Preferences->Options，**修改Step download timeout(sec)的时间。**

2、或者把“Step timeout caused by resources is a warning”**设置为Yes**，这样下载资源超时也只是作为警告，不作为错误提示，但是对于非资源的下载超时，则总是会提示错误.

3、如果还有超时现象，需要在“Runtime Setting”>“Internet Protocol：Preferences”>“Advanced”区域中设置一个“winlnet replay instead of sockets”选项，再回放是否成功。

## 服务器压力过大超时。

错误现象 ：Action.c(81):Continuing after Error -27498: Timed out while processing URL=http://172.18.20.70:7001/workflow/bjtel/leasedline/ querystat/ subOrderQuery.do

错误分析：这种错误常常是因为并发压力过大，服务器端太繁忙，无法及时响应客户端的请求而造成的，所以这个错误是正常现象，是压力过大造成的。

如果压力很小就出现这个问题，可能是脚本某个地方有错误，要仔细查看脚本，提示的错误信息会定位某个具体问题发生的位置。

**解决办法：**

**1、**例如上面的错误现象问题定位在某个URL上，需要再次运行一下场景，同时在[其他](http://blog.csdn.net/loadrunn/article/details/7886918" \t "_self)机器上访问此URL。如果不能访问或时间过长，可能是服务器或者此应用不能支撑如此之大的负载。分析一下服务器，最好对其性能进行优化。

如果再次运行场景后还有超时现象，就要在各种图形中分析一下原因，例如可以查看是否服务器、DNS、网络等方面存在问题。

2、增加一下运行时的超时设置，在“Run-Time Settings”>“Internet Protocol:Preferences”中，单击“options”，增加“HTTP-request connect timeout” 或者“HTTP-request receive”的值。

## LoadRunner HTTP服务器状态代码：404/500

在录制Web协议脚本回放脚本的过程中，会出现HTTP服务器状态代码，例如常见的页面-404错误提示、-500错误提示。

**错误现象1**：-404 Not Found服务器没有找到与请求URI相符的资源，但还可以继续运行直到结束。

错误分析：此处与请求URI相符的资源在录制脚本时已经被提交过一次，回放时不可再重复提交同样的资源，而需要更改提交资源的内容，每次回放一次脚本都要改变提交的数据，保证模拟实际环境，造成一定的负载压力。

解决办法：在出现错误的位置进行脚本关联，在必要时插入相应的函数。

**错误现象2**：-500 Internal Server Error服务器内部错误，脚本运行停止。

错误分析：服务器碰到了意外情况，使其无法继续回应请求。

解决办法：出现此错误是致命的，说明问题很严重，需要从问题的出现位置进行检查，此时需要此程序的开发人员配合来解决，而且产生的原因根据实际情况来定，测试人员无法单独解决问题，而且应该尽快解决，以便于后面的测试。

## 录制Web协议脚本回放停止

LoadRunner请求无法找到：在录制Web协议脚本回放脚本的过程中，会出现请求无法找到的现象，而导致脚本运行停止。

错误现象：Action.c(41): Error -27979: Requested form. not found [MsgId: MERR-27979]

Action.c(41): web\_submit\_form. highest severity level was "ERROR",0 body bytes, 0 header bytes [MsgId: MMSG-27178]"

这时在tree view中看不到此组件的相关URL。

错误分析：所选择的录制脚本模式不正确，通常情况下，基于浏览器的Web应用会使用“HTML-based script”模式来录制脚本；而没有基于浏览器的Web应用、Web应用中包含了与服务器进行交互的Java Applet、基于浏览器的应用中包含了向服务器进行通信的JavaScript/VBScript代码、基于浏览器的应用中使用HTTPS安全协议，这 时则使用“URL-based script”模式进行录制。

**解决办法：**打开录制选项配置对话框进行设置，在“Recording Options” 的“Internet Protocol”选项里的“Recording”中选择“Recording Level”为“HTML-based script”，单击“HTML Advanced”，选择“Script. Type”为“A script. containing explicit”。然后再选择使用“URL-based script”模式来录制脚本。

## 模拟的用户数量却上不去-27796

为什么Windows系统中的CPU、内存等资源仍然充足，但是模拟的用户数量却上不去-27796?

     在Windows计算机的标准设置下，操作系统的默认限制只能使用几百个Vuser，这个限制与CPU或内存无关，主要是操作系统本身规定了默认的最大线程数所导致。要想突破Windows这个限制，须修改Windows注册表。以Windows XP Professional为例。

       (1)打开注册表后，进入注册表项HKEY\_LOCAL\_MACHINE中的下列关键字：SystemCurrentControlSetControlSession ManagerSubSystems。

        (2)找到Windows关键字，Windows关键字如下所示：

%SystemRoot%system32csrss.exe bjectDirectory=Windows

SharedSection=1024,3072,512 Windows=On SubSystemType=Windows ServerDll=basesrv,1

ServerDll=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitialization,2

ProfileControl=Off MaxRequestThreads=16

SharedSection=1024,3072,512关键字的格式为xxxx,yyyy,zzz。其中，xxxx定义了系统范围堆的最大值(以KB为单位)，yyyy定义每个桌面堆得大小。

       (3)将yyyy的设置从3072更改为8192(即8MB)，增加SharedSection参数值。

 通过对注册表的更改，系统将允许运行更多的线程，因而可以在计算机上运行更多的Vuser。这意味着能够模拟的最大并发用户数量将不受Windows操作系统的限制，而只受硬件和内部可伸缩性限制的约束。

## 造数据的方式

**12.[性能测试](http://www.besttest.cn/article/www.besttest.cn) 往往需要准备大批量的数据，大批量数据的生成方法有很多种，常见的有：**

（1）编写SQL语句来插入数据

（2）使用DataFactory等专业的数据生成工具

（3）通过LoadRunner录制回放的方式重复执行生成大批量数据

        Tips:

        使用DataFactory插入nchar数据类型的数据时会出现空值的情况，可以先修改数据库的数据类型设置，插完数据后再改回来

## 录制脚本中包含中文，出现乱码

 把录制选项中的Support charset选中UTF-8

 录制脚本后，切换到树视图中，打开相应的脚本页面。在右侧的PageView中录制的脚本呈现中文版式，但是当切换到Server Response中，所有的中文全部换成的乱码，如“勌缞仫訆”。

原因是服务器端没有把响应的编码设置为gb2312

在IIS中找到 Web.Config文件，在<system.web>….</system.web>节加 入<globalization requestEncoding="gb2312" responseEncoding="gb2312" fileEncoding="gb2312"/>后再次录制脚本，乱码变中文。

## 回放脚本时，在浏览器显示的中文是乱码

解决办法（1）：

首先设置Run-Time Settings – Browser – Browser Emulation – User-Agent

然后设置IE：

查看－编码－钩上“自动选择”和Unicode（UTF-8）。

解决办法（2）：

使用lr\_convert\_string\_encoding函数来转换编码

下面是一个使用的例子：

lr\_convert\_string\_encoding(lr\_eval\_string("{ReplyContents}"),LR\_ENC\_SYSTEM\_LOCALE,LR\_ENC\_UTF8,"ReplyMessage");

web\_submit\_data("postreply.aspx\_2",

"Action=http://127.0.0.1/postreply.aspx?infloat=1&topicid=2&&inajax=1",

"Method=POST",

"EncType=multipart/form-data",

"RecContentType=text/xml",

"Referer=http://127.0.0.1/showtopic-2.aspx",

"Snapshot=t9.inf",

"Mode=HTML",

ITEMDATA,

"Name=iconid", "Value=0", ENDITEM,

"Name=title", "Value=", ENDITEM,

"Name=wysiwyg", "Value=0", ENDITEM,

"Name=checkbox", "Value=0", ENDITEM,

"Name=message", "Value=[localimg=180,112]1[/localimg]rnrn{ReplyMessage}", ENDITEM,

"Name=e\_mediatyperadio", "Value=on", ENDITEM,

"Name=sl\_attachdesc", "Value=", ENDITEM,

"Name=attachid", "Value=", ENDITEM,

"Name=attachdesc", "Value=", ENDITEM,

"Name=localid", "Value=", ENDITEM,

"Name=attachdesc", "Value=", ENDITEM,

"Name=localid", "Value=1", ENDITEM,

"Name=attachdesc", "Value=", ENDITEM,

"Name=localid", "Value=2", ENDITEM,

"Name=emailnotify", "Value=on", ENDITEM,

"Name=postreplynotice", "Value=on", ENDITEM,

"Name=postfile", "Value=", "File=Yes", ENDITEM,

"Name=postfile", "Value=D:图片收集It's about time.JPG", "File=Yes", ENDITEM,

"Name=postfile", "Value=", "File=Yes", ENDITEM,

"Name=uploadallowmax", "Value=10", ENDITEM,

"Name=uploadallowtype", "Value=jpg,gif", ENDITEM,

"Name=thumbwidth", "Value=300", ENDITEM,

"Name=thumbheight", "Value=250", ENDITEM,

"Name=noinsert", "Value=0", ENDITEM,

LAST);

## Analysis设置思考时间

17.在场景设置中不忽略思考时间,但是在查看响应时间的时候怎样让LR自动在响应时间里减去思考时间？

  在analysis中找到了设置是否在报告中包含思考时间的地方做相应的设置即可：

## RTMP协议用 flex协议录制

18.测试RTMP协议应该在LoadRunner选择什么协议来录制？

 用flex协议

有这几个函数可用：

flex\_rtmp\_connect  Connects a client to an RTMP server and sets connection options.

flex\_rtmp\_disconnect  Disconnects a client from an RTMP server.

flex\_rtmp\_send  Sends messages to an RTMP server.

flex\_rtmp\_receive  Receives responses from an RTMP server

Flex can record and replay scripts involving RTMP (Real Time Messaging Protocol). In order to enable RTMP simulation, you must configure the recording options for the Flex protocol.

To enable RTMP:

1 Open the Recording Options dialog box by selecting Tools > Recording Options or clicking the Options button in the Start Recording dialog box.

2 In the Network > Port Mapping node click Options.

3 Set the Send-Receive buffer size threshold to 1500.

## 修改LR自带的例子端口号

LR自带的例子端口号怎么修改？

   LR自带的例子端口号是1080,我怎么样把这个端口设置我自己想用的端口号8088,在什么地方设置在LR安装目录下，找到Xitami.config文件，找到portbase，可以修改它（默认是1000）；

默认的端口号是portbase+80；要把端口号改成8088，就把portionbase改为8008，保存之后就是了（8088=8008+80）。

## 用Web\_reg\_find查找中文字符串时查找不到

脚本文件里有个default.cfg，里面有个参数是 UTF8InputOutput ，将其值改为0

## Web\_find失效

LoadRunner不执行检查方法错误现象在脚本中插入函数Web\_find，在脚本中设置文本以及图像的检查点，但是在回放过程中并没有对设置的检查点进行检查，即Web\_find失效。

错误分析：  
由于检查功能会消耗一定的资源，因此LoadRunner默认关闭了对文本以及图像的数据执行保护检查，所以在设置检查点后，需要开启检查功能。  
解决办法打开运行环境设置对话框进行设置，在“Run-time Settings”的“Internet Protocol”选项里的“Perference”中勾选“Check”下的“Enable Image and text check”选项。  
  需要注意的是，回放脚本时出现的错误有时是程序自身的原因导致的，因此在解决脚本回放问题前必须保证程序录制出的脚本是正确的。

## 如何从命令行调用LoadRunner脚本

 Here is the command line that you need to execute to run a VuGen script from the command prompt:

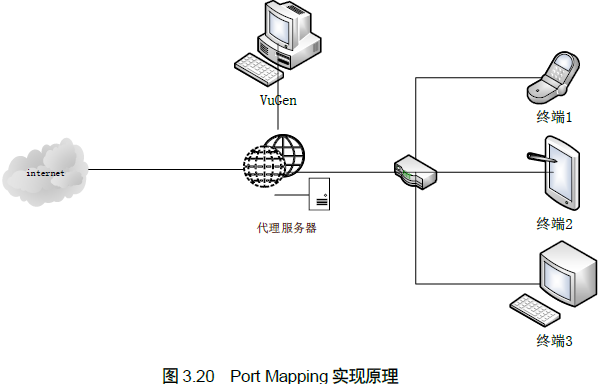
   <LoadRunner>binmmdrv.exe -usr <path to usr file>

Note:

In order to get all the other options that go with the command, run mmdrv.exe from the command prompt without any options.

## 录制Chrome或者**Linux**下的应用

Port Mapping提供了对于端口映射的处理规则，通过其内置的代理服务器，可以将通过代理服务器的协议内容录制下来，其实现原理如图3.20。



终端只要设置代理的地址为代理服务器的地址，在访问Internet的时候，VuGen会自动将经过代理服务器的协议请求录制下来，从而实现对任意应用的录制。

这里我们来模拟一个基于远程电脑Chrome访问百度首页的脚本录制，在Port Mapping中新建一个端口映射策略，注意在Allow forwarding to garget server form local port中选中并设置一个代理的端口，如图3.21。

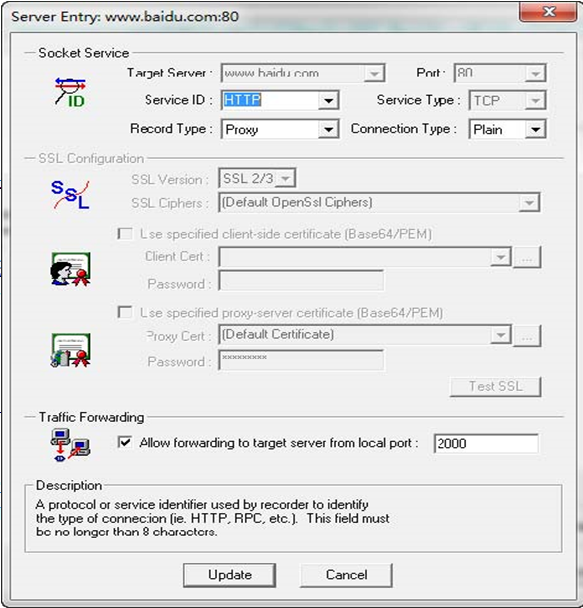
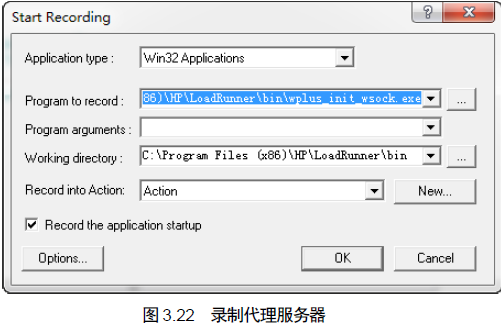


图3.21 Port Mapping代理设置

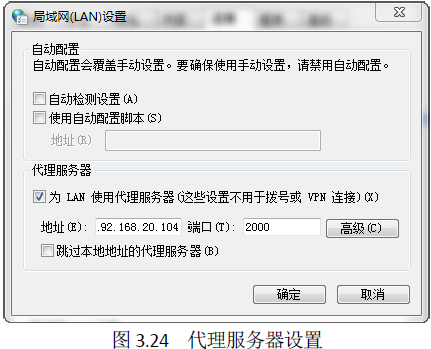
该设置的作用是在本机上启动一个端口为2000的代理，将这个端口映射成为本机的80端口访问外网（有点网桥的概念），只能通过HTTP协议和访问www.baidu.com才会被录制。接着我们启动录制，录制的对象为LR安装目录下的bin\wplus\_init\_wsock.exe程序，使用Win32 Applications录制，如图3.22。



启动录制后，我们会看到wplus\_init\_scock代理服务器会被启动，如图3.23。这个时候所有通过该代理服务器的匹配请求都会被录制下来。



接着我们来设置一下远程的Chrome浏览器，我们知道默认情况下LR11是无法启动录制Chrome浏览器的，现在通过代理模式来录制，就要设置Chrome浏览器的代理地址，让Chrome走的协议通过我们正在录制的代理服务应用，如图3.24。



Chrome使用了IE的代理服务器设置，这里将地址设置为启动代理的服务器IP地址，端口为2000，这样Chrome浏览器的所有协议都会从代理服务器上走过。在Chrome上访问www.baidu.com时就能看到VuGen将这些内容录制下来，停止录制后可以看到对应脚本。注意这个时候回放VuGen会自动将浏览器模拟成IE4.0。

## LR启动controller报错“transaction monitor not available”

1.多是OS系统问题，修复试试或升级

2.或修复LR试试

## Exception ACCESS\_VIOLATION received

Loadrunnber 报错误：Error -- memory violation : Exception ACCESS\_VIOLATION received.的一种情况

Loadrunner脚本，最后一步是点击“退出”按钮退出登录状态，如下：

web\_text\_link("[退出]",

    "Snapshot=t18.inf",

    DESCRIPTION,

    "Text=[退出]",

    ACTION,

    "UserAction=Click",

    LAST);

return 0;

但是会报错误：

Action1.c(94): Error: C interpreter run time error: Action1.c (94): Error -- memory violation : Exception ACCESS\_VIOLATION received.

Action1.c(94): Notify: CCI trace: Action1.c(94): web\_text\_link(0x02080870, 0x0208085f "Snapshot=t18.inf", 0x02080a42 "DESCRIPTION", 0x02080853, 0x02080a3b "ACTION", 0x020809de "UserAction=Click", 0x02080a6a "LAST")

Action1.c(94): Notify: CCI trace: Compiled\_code(0): Action1()

经过试验，发现是因为 退出后有个自动跳转。

如果退出结束代码修改为下面就没有问题了。

    web\_text\_link("[退出]",

       "Snapshot=t18.inf",

       DESCRIPTION,

       "Text=[退出]",

       ACTION,

       "UserAction=Click",

       LAST);

    web\_browser("Sync",

       "Snapshot=t18.inf",

       DESCRIPTION,

       ACTION,

        "Sync",

       LAST);

     return 0;

结论：

web\_text\_link如果有中间跳转，不能放在代码的最后，最后要有一个 Sync。

## 如果C代码函数在LR中，部分类型不读取

请问"int64这个类型,在LR中怎么表示"。我将一段C的代码放在LR中,LR不认int64这个类型,怎么解决?

 把那段C代码做成DLL，然后在LR中调用

## LoadRunner监控远程机器Windows资源，提示“net use failed”

解决办法：

1、在被监控机器上更改本地帐户的共享和安全模式为经典模式。控制面板->管理工具->本地安全策略->网络访问：本地帐户的共享和安全模式->经典模式。

监视连接前的准备工作：

2、首先保证被监视的windows系统开启以下二个服务Remote Procedure Call(RPC) 和Remote Registry Service

3、被监视的WINDOWS机器:右击我的电脑,选择管理->共享文件夹->共享在这里面要有C$这个共享文件夹,(要是没有自己手动加)

4、然后保证在安装LR的机器上使用运行.输入被监视机器IPC$ 然后输入管理员帐号和密码,如果能看到被监视机器的C盘了,就说明你得到了那台机器的管理员权限,可以使用LR去连接了

## [日志级别的选择：Debug、Info、Warn、Error还是Fatal](http://fengshen-xia.iteye.com/blog/295961)

        软件中总免不了要使用诸如 Log4net, Log4j, Tracer 等东东来写日志，不管用什么，这些东东大多是大同小异的，一般都提供了这样5个日志级别：  
    × Debug  
    × Info  
    × Warn  
    × Error  
    × Fatal  
        一个等级比一个高，但是在具体开发中，关于应该如何选择适应的等级，却没有找到好的文章进行说明。记录一下自己的一些看法，以便日后使用吧。  
=== Debug ===  
        这个级别最低的东东，一般的来说，在系统实际运行过程中，一般都是不输出的。  
        因此这个级别的信息，可以随意的使用，任何觉得有利于在调试时更详细的了解系统运行状态的东东，比如变量的值等等，都输出来看看也无妨。  
        当然，在每一个 Debug 调用之前，一定要加上 If 判断。  
=== Info ===  
        这个应该用来反馈系统的当前状态给最终用户的，所以，在这里输出的信息，应该对最终用户具有实际意义，也就是最终用户要能够看得明白是什么意思才行。  
        从某种角度上说，Info 输出的信息可以看作是软件产品的一部分（就像那些交互界面上的文字一样），所以需要谨慎对待，不可随便。  
=== Warn、Error、Fatal ===  
        警告、错误、严重错误，这三者应该都在系统运行时检测到了一个不正常的状态，他们之间的区别，要区分还真不是那么简单的事情。我大致是这样区分的：

        所谓警告，应该是这个时候进行一些修复性的工作，应该还可以把系统恢复到正常状态中来，系统应该可以继续运行下去。  
        所谓错误，就是说可以进行一些修复性的工作，但无法确定系统会正常的工作下去，系统在以后的某个阶段，很可能会因为当前的这个问题，导致一个无法修复的错误（例如宕机），但也可能一直工作到停止也不出现严重问题。  
        所谓Fatal，那就是相当严重的了，可以肯定这种错误已经无法修复，并且如果系统继续运行下去的话，可以肯定必然会越来越乱。这时候采取的最好的措施不是试图将系统状态恢复到正常，而是尽可能地保留系统有效数据并停止运行。  
        也就是说，选择 Warn、Error、Fatal 中的具体哪一个，是根据当前的这个问题对以后可能产生的影响而定的，如果对以后基本没什么影响，则警告之，如果肯定是以后要出严重问题的了，则Fatal之，拿不准会怎么样，则 Error 之。  
=== 一些疑惑 ===  
        不过在实际使用中，基于上面的这种考虑，也还是有一些具体问题。最常见的就是要在最终产品中将输出日志打开到那种级别才算好呢？  
        例如在应用中有一个输出窗口，一些系统状态信息将被输出到这个输出窗口中。因为 Info 的级别是如此之低，所以为了让用户能够看到有效的输出信息，必须将日志级别开放到 Info 级别。但是 Warn 的级别比 Info 要高，所以用户不得不被迫看到一些 Warn 的信息。而我们其实已经假定，Warn 信息其实并不影响系统的正常运行，这一般只代表系统中存在一些还没有被发现或者修改的小 Bug。这些 Warn 信息会让最终用户困惑甚至恐慌，系统发出警告了，该怎么办？  
        个人观点，Info 的级别应该比 Warn 更高才对，Warn 信息和 Debug 一样，应该在产品测试和调试时使用，而 Info、Erro 以及 Fatal 则在产品发布后需要继续使用。  
        目前我所采用的解决方法是，对于 Warn、Error、Fatal 都添加一个相应的系统断言，这样，可以保证当发生这种问题时，在调试阶段，可以立即得到提示。在软件发布以后，这些信息也能被记录到日志文件中去。

**Java代码  [收藏代码](javascript:void())**

1. log.Warn("message");
2. System.Diagnostics.Debug.Fail("警告", "message");

         Debug.Fail 将导致编译为 Debug 输出时，会弹出一个消息警告窗口，这可保证在测试、调试阶段不漏过任何一个潜在的错误。而在发布时，Release 编译的输出不会包括 Debug 语句，这就不会打扰最终用户，而错误信息仍然能通过 log 记录到日志中。

**以上观点定义不是特别准确，以后在补充详细。**

## 字符串替换

#include "lr\_replace.h"

Action()

{

//times格式：20150706102524.218

//取参数化变量的当前值并保存到LRParam参数

lr\_save\_string( lr\_eval\_string("{times}"), "LRParam");

//查找参数里面的点替换成空

lr\_replace( "LRParam", ".", "" );

//取参数值并打印

lr\_output\_message( "包含毫秒的字符串1 %s", lr\_eval\_string("{LRParam}") );

return 0;

}

Action : 20150706102524218



## 取参数长度

Action()

{

lr\_save\_string("Aqwe位个", "Param");

//loadrunner：取参数长度，可以按照10进制和16进制进行取长度

web\_save\_param\_length("Param", "Base=Decimal", LAST );

lr\_output\_message ("获取参数长度： %s",lr\_eval\_string ("{Param\_Length}"));

return 0;

}

Action.c(8): 获取参数长度： 8

参数解释：Param 需要获得的参数名称，Base 参数大小的格式包括Hexadecimal(十六进制) 和 Decimal(十进制)

## 简单判读语句

if (atoi(lr\_eval\_string("{cs}")) > 0){ //判断如果Welcome字符串出现次数大于0

lr\_output\_message("登录成功");

lr\_output\_message ("我登录的用户名是：%s",lr\_eval\_string ("{cs}"));

}else{ //如果出现次数小于等于

lr\_error\_message("登录失败"); //在日志中输出Log on failed

lr\_output\_message ("我登录的用户名是：%s",lr\_eval\_string ("{cs}"));

return -1 ; }

## 在C语言中，&有三种作用

分别如下：

1. 取地址。单目运算符。用来取一个变量的地址。

比如

int i, \*p;  
p = &i;//这里的&作用是取变量i的地址。

2. 位操作中的与操作运算符。

也就是常说的and操作，双目运算符。

计算的时候按位计算，&两边操作数对应位上全为1时，结果的该位值为1。否则该位值为0

比如

0x12&0x23 转为二进制为:

B00010010&B00100011，

按位计算结果为B00000010，

即结果为0x02。

3. 当两个&&一起用的时候，表示为逻辑运算中的与运算。

逻辑运算结果只有0和1两种结果。

一般在&&两边是两个逻辑表达式。

当两个表达式值均为1时，&&运算结果为1，否则为0。

比如

2>1 && 0<1 这个的值为1&&1 = 1。

1>2 && 2>0 这个值为0&&1 = 0。

1>2 && 2>3 这个值为0&&0=0。

# 关联函数

## 抓取所有相同左右边界的动态value

怎样抓取有相同左右边界的动态value？例如：

stateID="d7lg0ehmjkkm6uin3s4boei7oq">

stateID="cvopakp46ftsf8mh6l37ti3ubm">

stateID="bv9mja8gtgr39ddibm5t9163re">

web\_reg\_save\_param里的ORD应该怎样设置？

ORD: Indicates the ordinal position or instance of the match. The default instance is 1. If you specify "All," it saves the parameter values in an array.

例子：

char outFlightParam[50]; // The name of the parameter for correlation

char outFlightParamVal[50]; // The formatted value of outFlightParam

web\_reg\_save\_param("outFlightVal",

"LB=outboundFlight value=", "RB=>",

"ORD=ALL",

"SaveLen=18",

LAST);

web\_submit\_form("reservations.pl",

"Snapshot=t4.inf",

ITEMDATA,

"Name=depart", "Value=London", ENDITEM,

"Name=departDate", "Value=11/20/2003", ENDITEM,

"Name=arrive", "Value=New York", ENDITEM,

"Name=returnDate", "Value=11/21/2003", ENDITEM,

"Name=numPassengers", "Value=1", ENDITEM,

"Name=roundtrip", "Value=<OFF>", ENDITEM,

"Name=seatPref", "Value=None", ENDITEM,

"Name=seatType", "Value=Coach", ENDITEM,

"Name=findFlights.x", "Value=83", ENDITEM,

"Name=findFlights.y", "Value=16", ENDITEM,

LAST);

sprintf(outFlightParam, "{outFlightVal\_%s}", lr\_eval\_string("{outFlightVal\_count}"));

sprintf(outFlightParamVal, "Value=%s", lr\_eval\_string(outFlightParam));

lr\_message("The value argument is : %s", outFlightParamVal);

web\_submit\_form("reservations.pl\_2",

"Snapshot=t5.inf",

ITEMDATA,

"Name=outboundFlight",outFlightParamVal, ENDITEM,

"Name=reserveFlights.x", "Value=92", ENDITEM,

"Name=reserveFlights.y", "Value=10", ENDITEM,

LAST);

# 参数化

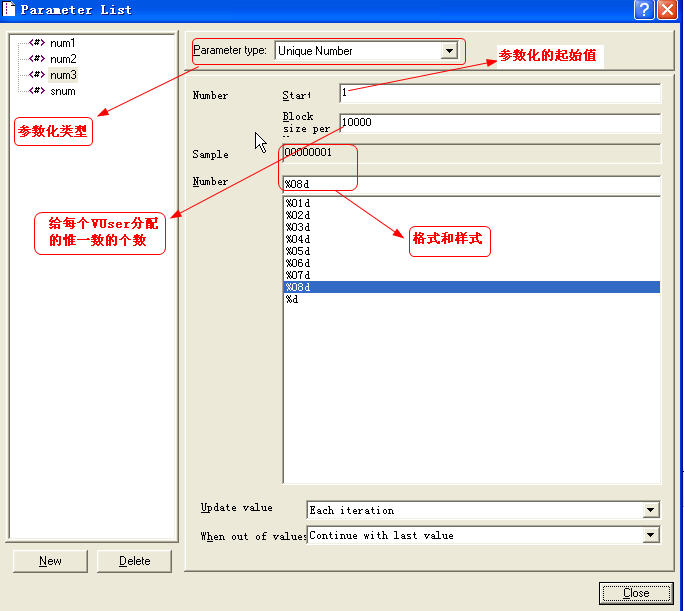
Unique Number

要求：

1、所有的链接进来的线程不能关闭并且不能重复，所以socket这块就要设置惟一数

2、所有用户的use\_id当然也不能重复，所以这也要一个惟一数

3、游戏的座位号也要求惟一数



例如：

如果start=1，Block size per Vuser=10000时，就相当于给每个用户分配了10000个惟一数字，第一个用户从1开始，第二个用户从10001开始，第三个用户从20001开始，依次加10000；

但是如何设置好start 和Block size per Vuser这也是一门学问，设置不好就会出现数字不够用的情况，刚开始我就在这块走了很多弯路，经过百度发现如果一下个公式供大家参考：

设：块大小设置为:B(block)，起始值为：S（start)，当前用户为：Ｎ(number)，当前取值为Ｖ(value)

则：

V = B\*(N-1)+S

# Socket协议

## 如何忽略Socket接收数据的验证

在LR中对Socket进行性能测试时，LR会自己判断lrs\_receive回来的数据的长度，而如果长度不符的话会有时间延迟的情况(这是性能测试完全不能接受的事情)，如果做到这一点呢，经过反复尝试，发现一种简单的方法(用\*代替具体的长度)：

类似于将：

recv buf1 12

"Hello, Denny"

改为：

recv buf1 \*

"Hello, Denny"

一切OK。

## 项目实例：

lr\_load\_dll("LRAssist.dll"); //这之后就可以调用LRAssist.dll中的函数了。

lrs\_set\_send\_buffer ("socket0", buf, strlen(buf));

lrs\_send("socket0", "buf0", LrsLastArg);

//这两句话要在一起用，第一句定义了下一句要发送的数据包buf和数据包的长度strlen（buf），buf0被忽略。  
lrs\_receive("socket0", "buf1", LrsLastArg);

buf=lrs\_get\_received\_buffer ("socket0", 0, -1, NULL);

//这两句话也要一起用，第一句说要接收一个数据包buf1，第二句说接收了一个数据包并赋值给字符串buf。buf的声明是char \*buf。

### ATM\_120\_2401\_余额查询

Action()

{

unsigned char BitMap[17]; // 声明存放bitmap字符数组

unsigned char DESTemp[512]; // 声明存放2~49号域报文体的字符数组

unsigned char PassWord[9]; // 声明存放52号域（PIN码）的字符数组

unsigned char DES128[9]; // 声明存放128号域的字符数组

unsigned char sendbuf[512]; // 声明存放整个发送报文的字符数组

unsigned char respcode[9]; // 声明存放返回报文39、41域的字符数组，用于判断交易处理成功与否

char \* ReturnData = NULL; // 存放返回结果的字符指针

char mystr[8]; // 声明存放socket连接名称的字符数组

int intConn = -1; // 存放建立socket连接返回结果

// 内存字符数组初始化

memset(BitMap, 0, 17);

memset(DESTemp, 0, 512);

memset(PassWord, 0, 9);

memset(DES128, 0, 9);

memset(sendbuf, 0, 512);

memset(respcode, 0, 9);

memset(mystr, 0, 8);

// 组报文体

sprintf(DESTemp, "%s", lr\_eval\_string(

"19<P\_CardNo>" // P-2 账号 N..19(LLVAR)

"300000" // P-3 交易码 N6

"<P\_DE15\_Date><P\_DE12\_DateTime>" // P-7 交易日期时间 N10

"<P\_DE11\_Rand6>" // P-11 系统跟踪号 N6

"<P\_DE12\_DateTime>" // P-12 本地交易时间 N6

"<P\_DE15\_Date>" // P-13 本地交易日期 N4

"001" // P-24 Network International ID N3

"0808241000" // P-33 授权机构标识码 N..11(LLVAR)

"316210330710010447195=49121202030" // P-35 二磁道数据 Z…37(LLVAR)

"000" // P-36 三磁道数据 Z...104(LLVAR)

// "000114000049121=000000000000=000000000000=00000001"

// "2606000"

"982607" // P-41 收单单位终端标识码 ANS6

"1182 " // P-42 收单商户定义码 ANS15

"001" // P-49 交易货币代码 AN3

));

//sprintf的作用是将一个格式化的字符串输出到一个目的字符串中，而printf是将一个格式化的字符串输出到屏幕。sprintf的第一个参数应该是目的字符串，如果不指定这个参数，执行过程中出现 "该程序产生非法操作,即将被关闭...."的提示。

sprintf(DES128, "%s", lr\_eval\_string("00000000")); // P-128 信息确认码（MAC） B8

// P-52 用户密码(8)

PassWord[0] = '\x1d';

PassWord[1] = '\x61';

PassWord[2] = '\x1d';

PassWord[3] = '\x16';

PassWord[4] = '\xeb';

PassWord[5] = '\x60';

PassWord[6] = '\x09';

PassWord[7] = '\xe3';

// 组bitmap 16位

BitMap[0] = '\xe2';

BitMap[1] = '\x38';

BitMap[2] = '\x01';

BitMap[3] = '\x00';

BitMap[4] = '\xB0';

BitMap[5] = '\xC0';

BitMap[6] = '\x90';

BitMap[7] = '\x00';

BitMap[8] = '\x00';

BitMap[9] = '\x00';

BitMap[10] = '\x00';

BitMap[11] = '\x00';

BitMap[12] = '\x00';

BitMap[13] = '\x00';

BitMap[14] = '\x00';

BitMap[15] = '\x01';

memcpy( sendbuf, "01680000000200", 14 ); // 将报文长度（4位）+000000（6位）+MTI（4位）COPY进总的发送报文sendbuf

memcpy( sendbuf+14, BitMap, 16 ); // 将BitMap（16位）COPY进总的发送报文sendbuf

memcpy( sendbuf+30, DESTemp, 126 ); // 将报文体中2~49域COPY进总的发送报文sendbuf

memcpy( sendbuf+156, PassWord, 8 ); // 将报文体中52域（8位）COPY进总的发送报文out

memcpy( sendbuf+164, DES128, 8 ); // 将报文体中128域（8位）COPY进总的发送报文out

sprintf( mystr, "%s", lr\_eval\_string("ATM<P\_SockId>") ); // socket连接名称

intConn = lrs\_create\_socket( mystr, "TCP", "RemoteHost=<P\_PBHostIp>", LrsLastArg ); // 建立socket连接

lr\_start\_transaction( "ATM\_120\_2401\_余额查询" ); // 事务开始

if( 0 == intConn ) // 如果socket连接创建成功

{

lrs\_set\_send\_buffer( mystr, (char\*)sendbuf, 172 ); // 缓冲区数据重定义

lrs\_set\_send\_timeout( 30, 0 ); // 设置发送报文的超时时间

lrs\_length\_send( mystr, "buf0", 0, LrsLastArg ); // 发送报文

lrs\_set\_recv\_timeout( 30, 0 ); // 设置接收返回报文的超时时间

lrs\_receive( mystr, "buf1", LrsLastArg ); // 接收返回报文

ReturnData = lrs\_get\_received\_buffer( mystr, 0, - 1, NULL ); // 抓取返回数据

if( ReturnData == NULL ){

lr\_end\_transaction( "ATM\_120\_2401\_余额查询", LR\_STOP );

lr\_error\_message( "TimeOut: No Answer==[CardNo=%s][Traceid=%s]", lr\_eval\_string( "<P\_CardNo>" ), lr\_eval\_string( "<P\_DE11\_Rand6>" ) );

} // 如果没有接收到返回结果报文事务标识未STOP

else{

memcpy( respcode, ReturnData+99, 8 ); // 将系统处理返回状态码赋值给respcode变量

if( 0 == memcmp( respcode, "00982607", 8 ) ) // 如果系统处理返回码表示交易处理成功

lr\_end\_transaction( "ATM\_120\_2401\_余额查询", LR\_PASS ); // 事务结束，返回状态为PASS

else // 系统处理返回码表示交易处理不成功

{

lr\_error\_message ( "TRANSACTION FAIL![CardNo=%s][Traceid=%s]", lr\_eval\_string( "<P\_CardNo>" ), lr\_eval\_string( "<P\_DE11\_Rand6>" )); // 将系统返回报文中的错误代状态码输出到LR执行的log文件和OutPut窗口

lr\_end\_transaction( "ATM\_120\_2401\_余额查询", LR\_FAIL ); // 事务结束，返回状态为FAIL

}

} // 如果接收到返回报文

lrs\_close\_socket( mystr ); // 关闭socket连接

ReturnData = NULL; // 释放ReturnData变量空间

} // 如果socket连接创建成功

else

{

lr\_error\_message( "SOCKET CREATE FAIL! ERROR CODE=[%d]", intConn ); // 将创建socket连接不成功的返回码输出到LR执行的log文件和OutPut窗口

lr\_end\_transaction( "ATM\_120\_2401\_余额查询", LR\_STOP ); // 事务结束，返回状态为STOP

}

return 0;

}

### 12\_控管\_柜面更新PIN密钥

**Action文件内容**

#include "lrs.h"

char \*sRecvBuf;

int iRecvLen;

char saveBuf[1024];

char sendBuf[1024];

char checkBuf[6+1];

char \*p\_temp = 0;

Action()

{

lr\_start\_transaction("12\_控管\_柜面更新PIN密钥");

lrs\_send("socket0", "buf0", LrsLastArg);

lrs\_receive("socket0", "buf1", LrsLastArg);

//lrs\_get\_last\_received\_buffer("socket0",&sRecvBuf,&iRecvLen);

sRecvBuf=lrs\_get\_received\_buffer("socket0",0,-1,"ascii");

//丢弃头15个字节

//lr\_save\_string(sRecvBuf+2,"lastBuf");

memset(saveBuf,0x00,1024);

p\_temp = sRecvBuf + 17;

memcpy(saveBuf,p\_temp,62);

//memcpy(saveBuf,lr\_eval\_string("<lastBuf>")+15,62);

memset(sendBuf,0x00,1024);

sprintf(sendBuf,"DA0041003%s2040004",saveBuf);

//lrs\_free\_buffer(sRecvBuf);

//lrs\_save\_param\_ex("socket1","user",sendBuf,0,strlen(sendBuf),"ascii","Param\_buf1");

lr\_save\_string(sendBuf,"Param\_buf1");

lrs\_send("socket1", "buf2", LrsLastArg);

lrs\_receive("socket1", "buf3", LrsLastArg);

//lrs\_get\_last\_received\_buffer("socket1",&sRecvBuf,&iRecvLen);

sRecvBuf=lrs\_get\_received\_buffer("socket1",0,-1,"ascii");

//判断是否接收成功

if(strlen(saveBuf+2)==0){

lr\_end\_transaction("12\_控管\_柜面更新PIN密钥", LR\_FAIL);

}

else{

//解析接收到的报文

lr\_save\_string(sRecvBuf+2,"lastBuf");

memset(checkBuf,0x00,7);

memcpy(checkBuf,lr\_eval\_string("<lastBuf>")+6,6);

if(strcmp(checkBuf,"000000")==0){

lr\_end\_transaction("12\_控管\_柜面更新PIN密钥", LR\_PASS);

}

else{

lr\_end\_transaction("12\_控管\_柜面更新PIN密钥", LR\_FAIL);

lr\_error\_message("12\_控管\_柜面更新PIN密钥失败，发送的密钥:【%s】,返回报文：【%s】",lr\_eval\_string("<miyao>"),lr\_eval\_string("<lastBuf>"));

}

}

//lrs\_free\_buffer(sRecvBuf);

return 0;

}

**data.ws文件内容**

;WSRData 2 1

send buf0 30

"\x00\x1c"

"TA28110010010012ABS.<miyao>.zpk"

recv buf1 79

send buf2 84

"\x00\x52"

"<Param\_buf1><miyao>"

recv buf3 14

-1

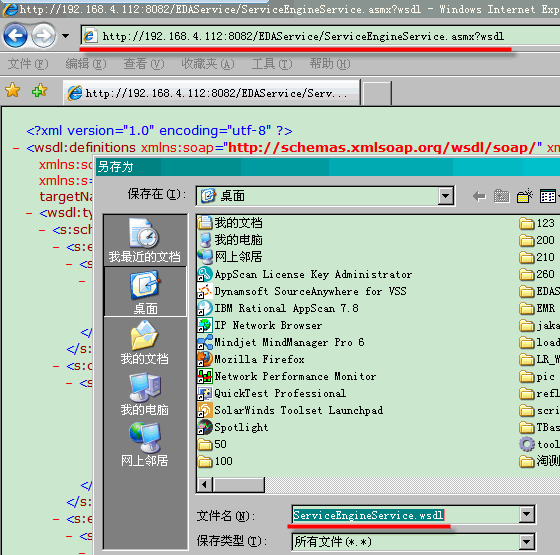
# [WebServices](http://www.cnblogs.com/spo22/archive/2010/06/17/1759795.html)协议

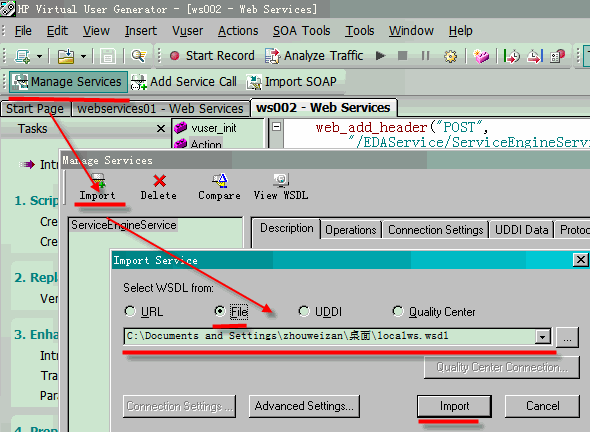
有三种方式：常用的有2种web\_service\_call（）方式，需要准备wsdl文件；第二种就是soap\_request()方式，需要准备xml格式的请求文件；第三种就是web\_custom\_request()模拟http方式发请求（post/get）

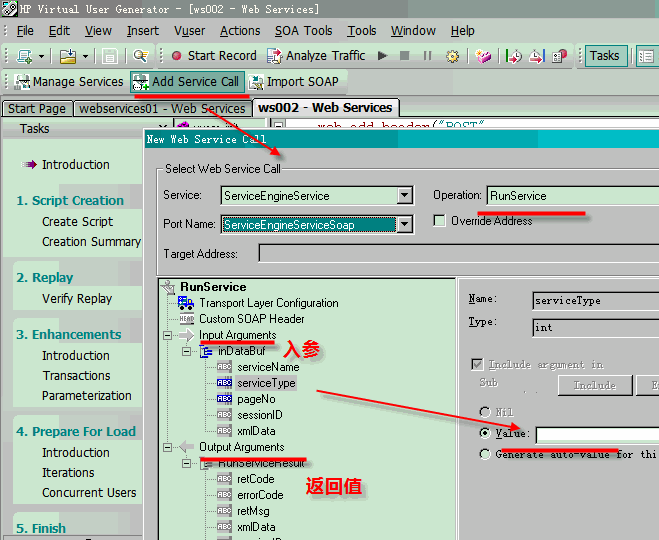
使用loadrunner测试Web Services的程序大致可以使用两种方法，分别为web\_service\_call（）和soap\_request（）。两者的使用很相识，我们录制时都使用Web Services的协议。下面分别介绍一下使用方法：

## web\_service\_call（）

**web\_service\_call（）使用步骤如下：**  
**1、保存WSDL文件。在IE中打开Web Services的地址，并加上“？wsdl”。**比如：“[http://192.168.4.112:8082/EDASer ... neService.asmx?wsdl](http://192.168.4.112:8082/EDAService/ServiceEngineService.asmx?wsdl" \t "_blank)”。然后将打开的XML文件另存为扩展名为".wsdl"的文件。如下图

  
**2、在loadrunner中导入刚才保存的WSDL文件，**如下图

  
**3、增加调用函数，**如下图

  
**4、完成后将自动生成以下代码。**

web\_service\_call( "StepName=RunService\_101", "SOAPMethod=ServiceEngineService|ServiceEngineServiceSoap|RunService",

"ResponseParam=response",

"Service=ServiceEngineService",

"ExpectedResponse=SoapResult",

"Snapshot=t1273650512.inf",

BEGIN\_ARGUMENTS,

"xml:inDataBuf="

"<inDataBuf>"

"<serviceName>2307</serviceName>"

"<serviceType>3</serviceType>"

"<pageNo></pageNo>"

"<sessionID></sessionID>"

"<xmlData><DataSet></xmlData>"

"</inDataBuf>",

END\_ARGUMENTS,

BEGIN\_RESULT,

"RunServiceResult=Param\_RunServiceResult",

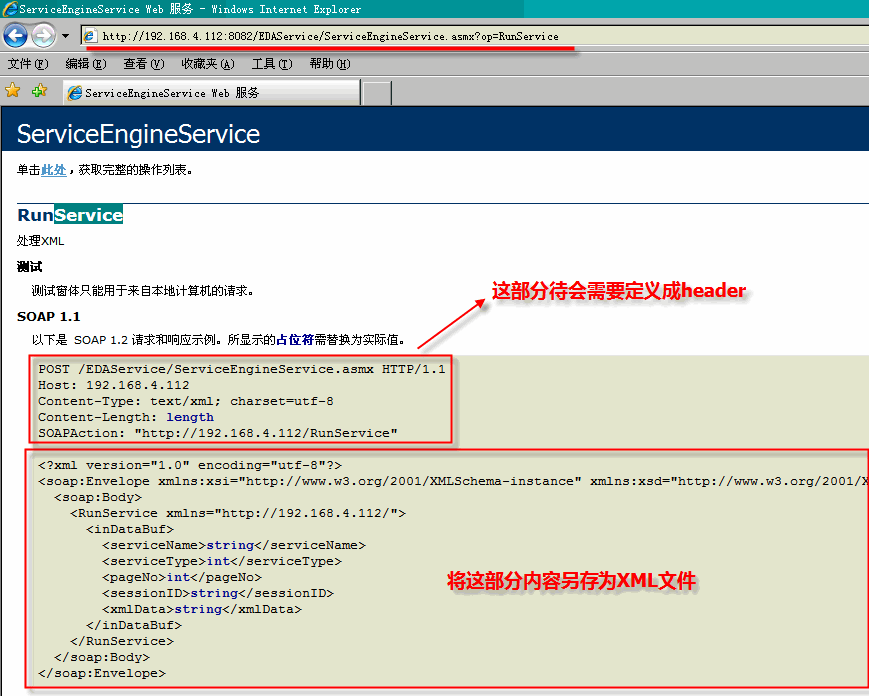
END\_RESULT,

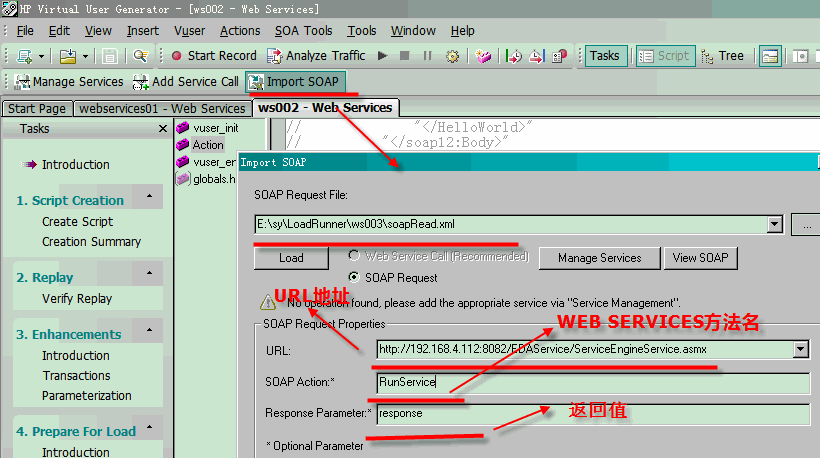
LAST);

lr\_log\_message("result = %s", lr\_eval\_string("{Param\_RunServiceResult}")); --可以输出返回值

## soap\_request（）

**soap\_request（），使用步骤如下：**  
**1、定义SOAP REQUEST FILE。**在IE中打开Web Services的页面，比如：[http://192.168.4.112:8082/EDASer ... .asmx?op=RunService](http://192.168.4.112:8082/EDAService/ServiceEngineService.asmx?op=RunService" \t "_blank)。IE中将显示这个文件的信息，复制保存为XML文件。如图下图：

  
**2、导入刚才定义的XML文件。**

  
**3、导入后自动以下代码。**  
soap\_request("StepName=SOAP Request",

"URL=http://192.168.4.112:8082/EDAService/ServiceEngineService.asmx",

"SOAPEnvelope="

"<?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>"

"<RunService xmlns=\"http://192.168.4.112/\">"

"<inDataBuf>"

"<serviceName>2307</serviceName>" --入参需要改成自己需要的值

"<serviceType>3</serviceType>" --入参需要改成自己需要的值

"<pageNo></pageNo>"

"<sessionID></sessionID>"

"<xmlData></xmlData>"

"</inDataBuf>"

"</RunService>"

"</soap12:Body>"

"</soap12:Envelope>",

"SOAPAction=RunService",

"ResponseParam=response",

"Snapshot=t1273722181.inf",

LAST);  
**4、在刚才自动生成的代码前，增加header信息。**需要增加的内容见步骤1里的截图。“Content-Length”不需要加。本例中我们增加的代码为：  
web\_add\_header("POST",

"/EDAService/ServiceEngineService.asmx HTTP/1.1");

web\_add\_header("Host",

"192.168.4.112");

web\_add\_header("Content-Type",

"application/soap+xml; charset=utf-8");  
这样2种方式就介绍完了。  
**补充说明：**  
**1、如果入参的字符串需要使用“<”或“>”,则必须传递“& l t ;”或“& g t ;”**(这几个字符需要去空格)  
**2、如果WEB SERVICES返回的是XML文件，则可以使用lr\_xml\_get\_values来获取。**比如：  
int NumOfValues;

NumOfValues= lr\_xml\_get\_values("XML={response}", --response为soap\_request函数的返回值

"ValueParam=OutputParam", --定义lr\_xml\_get\_values的返回值

"Query=/soap:Envelope/soap:Body/HelloWorldResponse/HelloWorldResult", --XML的节点名称，需要包含父节点的名称，并用"/"隔开。

"SelectAll=yes", LAST);

for ( i = 0; i < NumOfValues; i++) { /\* Print multiple values of OutputParam \*/ --输出刚才获取的XML的值

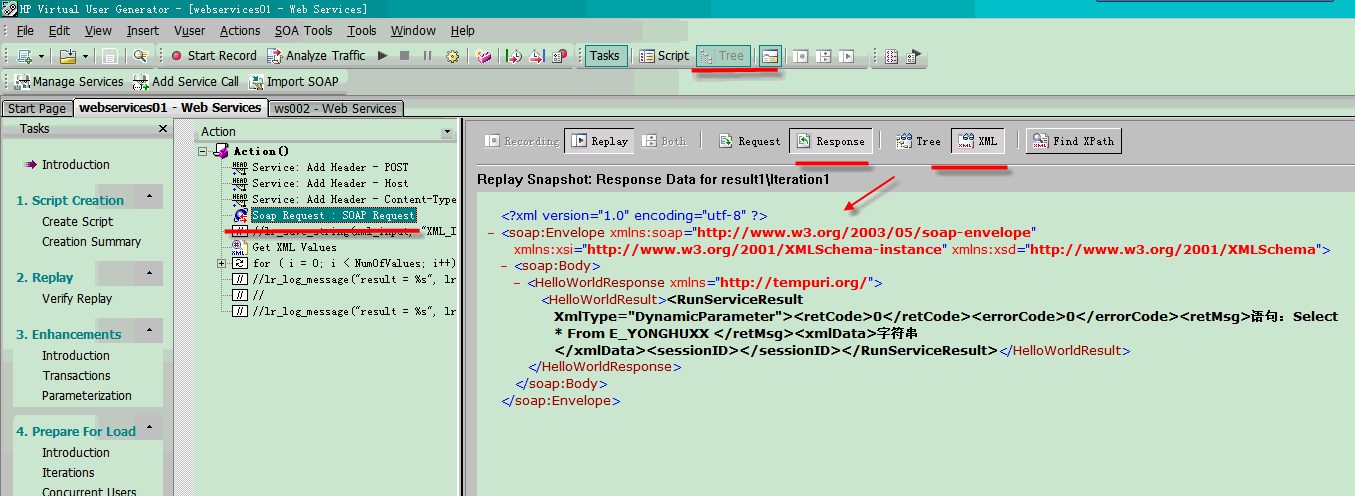
sprintf (buf, "{OutputParam\_%d}",i+1);

lr\_output\_message("result = %s",lr\_eval\_string(buf));

re = strncmp(lr\_eval\_string(buf),string1,85);

lr\_output\_message("return : %d",re);

}  
**3、可以切换到TREE界面查看我们定义的函数的请求，以及服务器的返回值。**

****

# LoadRunner性能测试时遇到的问题汇总

## Controller提示信息

1. Action.c(4): Error -27796: Failed to connect to server "10.10.10.2:81": [10048] Address already in use  
Try changing the registry value   
HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\TcpTimedWaitDelay to 30  
and HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\MaxUserPort to 65534  
and rebooting the machine  
See the readme.doc file for more information

2.Action.c(4): Error -26610: HTTP Status-Code=502 (Bad Gateway) for "<http://10.10.10.2:81/pa/newstock/hq/index.jsp>"

3.Error: Exception was raised when calling event-notify Vuser function in extension lrwreplaymain.dll: System Exceptions: EXCEPTION\_ACCESS\_VIOLATION

解决办法：在C盘下新建一个TEMP目录，把环境变量TMP,TEMP的值设置成C:\TEMP就ok

4.   Action.c(33): Error: C interpreter run time error: Action.c (33):  Error -- Unresolved symbol : vuser\_end.

5.Action.c(77): Continuing after Error -27728: Step download timeout (120 seconds) has expired when downloading non-resource(s)

解决办法：设置loadrunner9.1的设置过程，run-time setting---internet protocal----preferences----advanced--打开options--修改General中Step download timeout(sec)：600。  
这个值根据自己的实际需要而设置。我这里的场景是要下载一个附件，手工操作大概是3.4分钟。

6.Action.c(33): Error: C interpreter run time error: Action.c (33):  Error -- Unresolved symbol : vuser\_end.

--------意思是：user\_end部分的语法错误。比如缺少下面这段中的某行：

user\_end()

{

     return 0;

}

7.Connection to load generator failed.  
Error: Process "lr\_bridge.exe" was not created on remote host "192.168.170.161". Reason: communication error. Make sure the Agent process or service is running on the remote machine..    [MsgId: MERR-29987]  
Error: Communication error: Failed to connect to remote host [server IP: 192.168.170.161].    [MsgId: MERR-10343

---------原因在于：使用controller加载另外的负载机，提示failed。

解决办法：出现这种情况都是因为防火墙等之内的限制了。负载机是windows2008，装的是LoadRunner9.5，上面没有另外装防火墙或者是杀毒软件。所以关闭掉windows防火墙（控制面板---防火墙---关闭）。

**注：一下问题均出现在windows server 2003，loadrunner 11**

**问题1：**

Action.c(4): Error -27796: Failed to connect to server "ip:port": [10048] Address already in use  
Try changing the registry value   
HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\TcpTimedWaitDelay to 30  
and HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\MaxUserPort to 65534  
and rebooting the machine  
See the readme.doc file for more information

解决办法：

在负载生成器的注册表HKEY\_LOCAL\_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters里，有如下两个键值：

TcpTimedWaitDelay

MaxUserPort

1,这里的TcpTimedWaitDelay默认值应该中是30s，所以这里，把这个值调小为5s（按需要调整）。

2,也可以把MaxUserPort调大（如果这个值不是最大值的话）。

问题2：

Action.c(4): Error -27796: Failed to connect to server"ip:port": [10060] Connection timed out

解决办法：

出现这个问题很可能是访问应用部署的web容器允许最大连接量太低，我使用的是tomcat容器，所以修改tomcat的配置文件server.xml，如下

<connector   port="8080"     
      maxThreads="150"  
</connector   port="8080"

...................把maxThreads值再改大一点

顺便解释一下tomcat中标签中各个属性的含义：

<connector port="8080"

maxThreads="150"

minSpareThreads="25"

maxSpareThreads="75"

acceptCount="100"

/>

maxThreads="150" 表示最多同时处理150个连接

minSpareThreads="25" 表示即使没有人使用也开这么多空线程等待

maxSpareThreads="75" 表示如果最多可以空75个线程，例如某时刻有80人访问，之后没有人访问了，则tomcat不会保留80个空线程，而是关闭5个空 的。

acceptCount="100" 当同时连接的人数达到maxThreads时，还可以接收排队的连接，超过这个连接的则直接返回拒绝连接。</connector port="8080"

问题3：

Failed to Initialize. Reason: TimeOut

问题4：

Failed to Run. Reason: TimeOut

问题5：

Error: Communication error: Linked List client failed to initialize server pointer.  
  
1. 服务器没有响应  
2. 没有目标提供这个服务  
3. 发送包失败，原因是通讯错误  
4. 应该是收发包失败吧  
5. 超时  
综上，确实是服务器down掉了

①Error: Communication error: No response from the server. The server has shut down.

这个错误是不是表示，机器扛不住了。

## 脚本为空的解决方法：

1）如果安装了IE以外的浏览器，并且IE不是默认浏览器，则无法生成录制脚本

2）如果录制脚本时IE不能打开，则需要将浏览器的IE工具高级选项中，将“启用第三方浏览器扩展”的

勾选去掉

3）lr自带的webtours系统，如果点击Flights选项，出现空白页面，请检查本机oracle的 PERL5LIB环境

## 录制脚本时为什么不弹出IE浏览器

当一台主机上安装多个浏览器时，LoadRunner录制脚本经常遇到不能打开浏览器的情况，可以用下面的方法来解决。

启动浏览器，打开Internet选项对话框，切换到高级标签，去掉“启用第三方浏览器扩展(需要重启动)”的勾选，然后再次运行VuGen即可解决问题

提示：通常安装Firefox等浏览器后，都会勾选上面得选项，导致不能正常录制。因此建议运行

LoadRunner得主机上保持一个干净的测试环境。

# 常见的性能问题与分析

## 常见问题

a) 前台访问很慢，请帮忙分析优化

b) 用户对性能很不满意，再不解决就要投诉

c) 数据库负载很重，请帮忙分析一下

d) XXX功能打开需要1分钟，请帮忙分析一下。而等我访问这个功能的时候，可能几秒钟就返回；等你满怀困惑的找到问题提出人员，如果足够幸运的话，可能他告诉你要选择什么查询条件，问题能够重现；

a)系统性的问题? 比如CPU利用率，SWAP利用率或者IO过高导致的整体性能下降？

b)功能性问题？ 整体性能良好，个别功能时延很长

c)新出现问题？什么时候开始的，之前系统有哪些变动？（升级或者管理的资源大量增加）

d)不规律问题？有时候快，有时候慢，没有特定规律

1) 系统整体负载正常，但/nms/res/devicelist\_down.jsp目前经常出现35046毫秒的访问时延，请协助排查一下

2) 系统SWAP利用率经常会超过50%，这时候系统响应很慢，杀掉GetCGFlux.pl后恢复正常，请分析一下

3) 数据库服务器目前CPU利用率居高不下，已经持续了一段时间，请分析一下

## 基础问题检查

1. 主机基础故障问题

磁盘空间是否空闲为0？

SWAP利用率是否超过40%?

CPU利用率是否长时间超过85%？

网络是否持续丢包？

工作磁盘IO的利用率是否持续100%?

上述状况通常意味着系统有较严重问题，需要进一步从程序或者数据库上查找原因。

2. resin的JVM检查

Web应用的前台程序jsp和class都是运行在resin的JVM里，JVM(Java虚拟机)类似于oracle数据库，jsp和class类似于SQL,都可以看作一个系统软件，那么仅仅是看java进程在不在，前台能不能访问是不够的。就像没有sqlplus,PLSQL我们就无法维护数据库，同样的JVM也有相应的维护工具,，都在JAVA\_HOME/bin下

a. 查看JVM的内存占用情况

jstat -gcutil <pid> 3s 5

这条命令间隔3秒钟查看JVM的内存利用率，取样5次，

S0 S1 E O P YGC YGCT FGC FGCT GCT

0.00 98.51 44.95 39.41 63.43 9 0.070 2 0.195 0.266

永久内存区利用率63.43%, Elden和old区分别是44.95%和39.41%

b. 查看JVM的堆栈调用情况

jstack <pid>

这条命令把当前JVM里所有的线程调用堆栈输出。在前台访问无响应的时候，排查故障根源时特别有用。

上述Thread-831被38f06e88的线程所阻塞，而根据调用堆栈，可以准确的定位到执行程序，进行排查。

c. 查看错误日志是否有内存溢出错误

日志在resin/logs/error.log或者resin/log/jvm-default.log

如果有java.lang.OutOfMemoryError: PermGen space 说明JVM的永久内存区不足

-XX:MaxPermSize=256m 可根据永久内存区利用率调整到256M

java.lang.OutOfMemoryError说明JVM的堆内存不足

-Xmx2048m -Xms2048m 把堆内存调大到2048G

如果把Xmx加到2G，仍然会出现上述错误，那可能是有内存泄漏，需要开发人员排查

3. 数据库检查

oracle排查比较复杂，我只能从两方面简单举几个例子。

a) 系统参数层面优化

1) sga是否充分利用了系统内存，sga可以配系统内存的一半. 而我遇到过主机64G内存，sga\_target设置为5G

2) db\_cache\_size最好在sga\_target-3G，因为我们的程序很多没有使用绑定变量,如果不设置db\_cache\_size,则渐渐的SGA都有被share\_pool占用的趋势，数据被缓存的越来越少，获取数据需要从磁盘读取，这样整体性能肯定会下降。

3) shared\_servers设置为0，让数据库运行在专有模式而不是共享服务器模式

虽然系统参数调整会在整体上带来一定的性能改善，但相比于糟糕的SQL或者程序设计以及失效的索引和过期的统计数据对性能起到的作用，还是很有限的。

b) 应用优化层面

1) 定位问题SQL

这个SQL能够列出数据库当前正在执行的所有SQL,

select distinct s.sid,s.serial#,s.blocking\_session,p.spid PID,to\_char(s.logon\_time,'YYYY-MM-DD HH24:MI:SS') logontime ,

substr(s.machine,1,15) smachine,substr(s.program,1,20) sprogram,q.sql\_id,substr(q.sql\_text,1,200) sql from v$sql q,v$session s,v$process p

where q.hash\_value=s.sql\_hash\_value and q.address=s.sql\_address and p.addr=s.paddr

基于两个判断标准我们能很快的找到问题SQL

第一种是某个进程执行的SQL占用的CPU非常高，CPU利用率从Top命令获得，进程ID即PID

第二种是某类进程执行的SQL非常多，单个CPU不高，但合并起来就非常高了。

针对SQL就可以找支撑人员进一步判断是否需要找开发人员优化。

2) 是否存在session被其他session阻塞的情况

查看上一SQL结果的blocking\_session字段，如果被阻塞的进程都被某一会话锁定，需要把该session杀掉

alter system kill session 'sid,serail#';

遇到过几次系统非常慢的情况，经查看都是开发或者维护用plsql把某表锁住，导致相关会话都被阻塞

3) 对该SQL所涉及的表进行表分析，更新其统计信息

# [性能分析与调优](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1968332)

本章重点介绍性能分析，包括性能数据解读、性能问题发生的原因，如何分析等。对于性能调优，这里只是给一些简单的建议与参考。

## 性能分析

性能结果分析是性能测试中的一个重要部分，同时也是一个难点。由于不同的软件系统，不同的性能指标，结果分析方法都是不一样的。需要具体问题具体分析。下面将阐述一些性能分析的方法与建议。

### 性能分析的目的

1）找出系统瓶颈（硬件、软件）

2）提出性能优化方案

3）达到合理的硬件和软件配置

4）使系统资源使用达到最大平衡

### 常见性能瓶颈征兆

在性能测试执行过程中，我们需要观察和了解系统的运行状态，如果出现以下征兆，则表示系统可能存在瓶颈。

1)持续缓慢：应用程序一直特别慢，改变负载，对整体响应时间影响很少；

2) 随着时间推进越来越慢：负载不变，随着时间推进越来越慢，可能到达某个阈值，系统被锁定或出现大量错误而崩溃；

3)随着负载增加响应时间越来越慢：每增加若干用户，系统明显变慢，用户离开系统，系统恢复原状；

4)零星挂起或异常错误：可能是负载或某些原因，用户看到页面无法完成并挂起，无法消除；

5)可预见的锁定：一旦出现挂起或错误，就加速出现，直到系统完全锁定。通常要重启系统才解决。

6)突然混乱：系统一直运行正常，可能是一个小时或三天之后，系统突然出项大量错误或锁定。

### 性能数据解读建议

性能分析过程也是一个解读数据的过程，读懂了数据你就能知道问题出在何处。随着经验的累积将会很容易判断问题的根源，甚至在开发阶段就能对可能出现问题的点打预防针。

|  |  |  |  |
| --- | --- | --- | --- |
| **性能指标类型** | **标准** | **性能瓶颈征兆** | **分析工具** |
| TPS及其波动范围 | 1.Tps符合性能目标  2.Tps轨迹波动平稳 | 1.TPS有明显的大幅波动，不稳定。例如TPS轨迹缓慢下降，缓慢上升后骤降，呈瀑布型，呈矩形，分时间段有规律的波动，无规律的波动等。这些TPS的波动轨迹反映出被测试的性能点存在性能瓶颈，需要性能测试工程师与开发工程师查找性能瓶颈的原因。  2. TPS轨迹比较平稳，但是也存在波动现象。该类波动不明显，很难直接确定是否存在性能瓶颈。我们需要根据其他指标来进行判断。 | Jmeter/loadrunner |
| 响应时间 | 90%平均事务响应时间<性能目标 | 1.关注高峰负载时，用户操作响应时间；   2.关注数据库增量，对用户操作响应时间的影响。 | Jmeter/loadrunner |
| Web\DB服务器内存 |  | 1.很高的换页率  2.进程进入不活动状态;  3.交换区所有磁盘的活动次数过高;  4.过高的全局系统CPU利用率;  5.内存不够出错(out of memory errors) | Nmon/top/vmstat |
| WEB\DB服务器CPU | 合理使用的范围在60%至70% | 1.很慢的响应时间  2.CPU空闲时间为零  3.过高的用户占用CPU时间  4.过高的系统占用CPU时间  5.长时间的有很长的运行进程队列 | Nmon/vmstat/top |
| WEB\DB服务器磁盘I/O | Iowait<30% | 1.过高的磁盘利用率；  2.太长的磁盘等待队列；  3.等待磁盘I/O的时间所占的百分率太高；  4.太高的物理I/O速率；  5.过低的缓存命中率；  6.太长的运行进程队列，但CPU却空闲； | Nmon/sar/iostat |
| Oracle数据库 |  | 1.缓存命中率小于0.90  2.top 10sql耗时高  Oracle数据库的分析和优化，是一门专门的技术，进一步的分析可查相关资料，也可以查看数据库性能分析工具AWR章节。 | AWR |

### **如何定位性能问题**

性能问题的定位排查过程比较复杂，可以采用“拆分问题，隔离分析”的方法进行分析，即逐步定位、从外到内、从表及里、逐层分解、隔离排除。**以下分析顺序可供参考。**

**日志分析--->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，web服务器等）---〉数据库及应用瓶颈（SQL语句、数据库设计、业务逻辑、算法等）。**

以上过程并不是每个分析中都需要的，要根据测试目的和要求来确定分析的深度。整个过程中，要配套使用一些健康工具和日志进行。如： JDK自带的Jconsole，或者JProfiler，来监控服务器性能，oracle的监控工具awr等，具体工具可以参考工具篇。

另外，做性能测试的时候，我们一定要确保瓶颈不要发生在自己的测试脚本和测试工具上。

基于上述思想的指导，在具体执行层面**，可以参考如下分析过程：**

首先，当我们系统有问题的时候，我们不要急于去调查我们代码，这个毫无意义。我们首要需要看的是操作系统的报告。看看操作系统的CPU利用率，看看内存使用率，看看操作系统的IO，还有网络的IO，网络链接数，等等。通过观察这些数据，我们就可以知道我们的软件的性能基本上出在哪里。比如：

　　1）先看CPU利用率，如果CPU利用率不高，但是系统的Throughput和Latency上不去了，这说明我们的程序并没有忙于计算，而是忙于别的一些事，比如IO。（另外，CPU的利用率还要看内核态的和用户态的，内核态的一上去了，整个系统的性能就下来了。而对于多核CPU来说，CPU 0是相当关键的，如果CPU 0的负载高，那么会影响其它核的性能，因为CPU各核间是需要有调度的，这靠CPU0完成）

　　2）然后，我们可以看一下IO大不大，IO和CPU一般是反着来的，CPU利用率高则IO不大，IO大则CPU就小。关于IO，我们要看三个事，一个是磁盘文件IO，一个是驱动程序的IO（如：网卡），一个是内存换页率。这三个事都会影响系统性能。

　　3）然后，查看一下网络带宽使用情况，在Linux下，你可以使用iftop，iptraf，ntop，tcpdump这些命令来查看。或是用Wireshark来查看。

　　4）**如果CPU不高，IO不高，内存使用不高，网络带宽使用不高。但是系统的性能上不去。这说明你的程序有问题，比如，你的程序被阻塞了。可能是因为等那个锁，可能是因为等某个资源，或者是在切换上下文**。

通过了解操作系统的性能，我们才知道性能的问题，比如：带宽不够，内存不够，TCP缓冲区不够，等等，很多时候，不需要调整程序的，只需要调整一下硬件或操作系统的配置就可以了。具体配置项的调优，可以参考配置项调优参考章节。

接下来，我们需要使用性能检测工具，也就是使用某个Profiler来查看一下我们程序的运行性能。如：Java的JProfiler/TPTP/CodePro Profiler，GNU的gprof，IBM的PurifyPlus，Intel的VTune，AMD的CodeAnalyst，还有Linux下的OProfile/perf，后面两个可以让你对你的代码优化到CPU的微指令级别，如果你关心CPU的L1/L2的缓存调优，那么你需要考虑一下使用VTune。使用这些Profiler工具，可以让你程序中各个模块函数甚至指令的很多东西，如：运行的时间，调用的次数，CPU的利用率，等等。这些东西对我们来说非常有用。

我们重点观察运行时间最多，调用次数最多的那些函数和指令。这里注意一下，对于调用次数多但是时间很短的函数，你可能只需要轻微优化一下，你的性能就上去了（比如：某函数一秒种被调用100万次，你想想如果你让这个函数提高0.01毫秒的时间，这会给你带来多大的性能）

使用Profiler有个问题我们需要注意一下，因为Profiler会让你的程序运行的性能变低，像PurifyPlus这样的工具会在你的代码中插入很多代码，会导致你的程序运行效率变低，从而没发测试出在高吞吐量下的系统的性能，对此，一般有两个方法来定位系统瓶颈：

1）在你的代码中自己做统计，使用微秒级的计时器和函数调用计算器，每隔10秒把统计log到文件中。

2）分段注释你的代码块，让一些函数空转，做Hard Code的Mock，然后再测试一下系统的Throughput和Latency是否有质的变化，如果有，那么被注释的函数就是性能瓶颈，再在这个函数体内注释代码，直到找到最耗性能的语句。

最后再说一点，对于性能测试，不同的Throughput会出现不同的测试结果，不同的测试数据也会有不同的测试结果。所以，用于性能测试的数据非常重要，性能测试中，我们需要观测试不同Throughput的结果。

### 常见性能问题参考

下面是整理收集的一些常见问题列表，不全，也许并不对，大家可以补充指正。

|  |  |
| --- | --- |
| 类别 | 常见性能问题 |
| 操作系统类 | 1. Sys的CPU使用率过高  2. User的CPU使用率过高，持续大于80%以上  3.可用物理内存不足导致内存溢出  4. 磁盘空间不足导致交易处理失败，性能下降  5. TCP/IP连接数限制导致用户请求失败  6.磁盘IO使用比较繁忙，持续大于70% |
| 中间件类 | 常用主流中间件：Tomcat、apache、nginx、Weblogic、Jboss等  1.线程不回收导致溢出，引发宕机  2.数据库连接池不释放导致溢出  3.JVM内存参数设置不合理，新生代过大或偏小  4.永久代设置过小，导致栈溢出  5.其它问题 |
| 应用程序类 | 1.程序响应时间超长  2. JAVA程序内存溢出，内存中存放大量数据对象  3. JAVA程序循环嵌套过多，过于精细的查询条件，子查询间等待超时  4.程序中存在死循环引起线程死锁，导致CPU使用率达到100%  5.某些返回结果未定义处理方式，导致线程等待，不释放，CPU使用率高 |
| 数据库类 | 1.SGA分配不合理，需要具体情况具体分析  2.使用全表扫描  3.对于查询业务比较多的表，未建立索引，或建立的索引不合理，在索引列上使用IS NULL和IS NOT NULL  4.存在数据库死锁导致数据库连接超时或不释放。  5.存在过于复杂的计算，导致CPU、内存和IO使用率较高。  6. 数据库读写过于频繁，导致IO使用率比较高 |
| 其他问题 | 1.网络问题，被测试环境网络环境小于100M  2. 客户端问题等等 |

## 性能调优

性能调优是一个非常大的议题，更多的是开发人员进行，对于测试人员，可以了解一些通用的调优方法，并根据性能分析过程中发现的问题，给出一些建议。

### 性能调优目标与策略

性能优化的目标不外乎两个：1.时间性能：减小系统执行的时间；2.空间性能：减小系统占用的空间。

一般来说，性能优化也就是下面的几个策略：

用空间换时间。各种cache如CPU L1/L2/RAM到硬盘，都是用空间来换时间的策略。这样策略基本上是把计算的过程一步一步的保存或缓存下来，这样就不用每次用的时候都要再计算一遍，比如数据缓冲，CDN，等。这样的策略还表现为冗余数据，比如数据镜象，负载均衡什么的。

用时间换空间。有时候，少量的空间可能性能会更好，比如网络传输，如果有一些压缩数据的算法，这样的算法其实很耗时，但是因为瓶颈在网络传输，所以用时间来换空间反而能省时间。

简化代码。最高效的程序就是不执行任何代码的程序，所以，代码越少性能就越高。如：减少循环的层数，减少递归，在循环中少声明变量，少做分配和释放内存的操作，尽量把循环体内的表达式抽到循环外，条件表达的中的多个条件判断的次序，尽量在程序启动时把一些东西准备好，注意函数调用的开销（栈上开销），注意面向对象语言中临时对象的开销，小心使用异常，等等，这连东西需要我们非常了解编程语言和常用的库。

并行处理。如果CPU只有一个核，你要玩多进程，多线程，对于计算密集型的软件会反而更慢（因为操作系统调度和切换开销很大），CPU的核多了才能真正体现出多进程多线程的优势。并行处理需要我们的程序有Scalability，不能水平或垂直扩展的程序无法进行并行处理。从架构上来说，是否可以做到不改代码只是加加机器就可以完成性能提升？

总之，根据2：8原则来说，20%的代码耗了你80%的性能，找到那20%的代码，你就可以优化那80%的性能。

### 调优的原则

* 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。
* 确定清晰明确的性能目标是关键。
* 必须保证调优后的程序运行正确。
* 性能更大程度是取决于良好的设计，调优技巧只是一个辅助手段。
* 调优过程是叠代渐进的过程，每次调优的结果要反馈到后续的代码开发中去。
* 性能调优不能以牺牲代码的可读性和维护性为代价。

### 调优的基本步骤

1. 确定清晰的性能目标，并按优先级排列
2. 利用科学的测试工具对应用程序进行测试，并记录测试结果。
3. 把分布式系统拆分成组件：Web层、业务层、集成层、以及网络传输时间，分别进行调优。
4. 有系统的科学调优
   * 遵循一定的程序：测试性能→找出瓶颈→假设造成瓶颈的因素→测试假设是否成立→修改应用→再次测试性能
   * 确定影响性能的因素：CPU、内存还是IO。
   * 找出主要的瓶颈，首先解决最容易的，再重复测试。
   * 一次修改一个瓶颈，不要对不需要的地方进行调优
   * 提高CPU性能：更快的代码，更好的算法，减少短期生存的对象。
   * 提高内存性能：减少或减小长期生存的对象。
   * 提高IO性能：重新设计应用，减少IO的交互。

5.优化完成之后，进行QA测试。

6.在代码中记录优化的地方，并对旧代码进行注释。

### 性能调优的层次

为了提升系统性能，开发人员可以从系统的各个角度和层次对系统进行优化。除了最常见的代码优化外，在软件架构上、JVM虚拟机层、数据库以及操作系统层面都可以通过各种手段进行调优，从而在整体上提升系统的性能。

#### 设计调优

设计调优处于所有调优手段的上层，它往往需要在软件开发之前进行。在软件开发之初，架构师就应该评估系统可能存在的各种潜在问题，并给出合理的设计方案。由于软件设计和架构对软件整体质量有决定性的影响，所以，设计调优对系统性能的影响也是最大的。如果说，代码优化、JVM优化都是对系统微观层面上"量"的优化，那么设计优化就是对系统在宏观层面上"质"的优化。

设计优化的一大显著特点是，它可以规避某一个组件的性能问题，而非改良该组件的实现。比如，系统中组件A需要等待某事件E才能触发一个行为。如果组件A通过循环监控不断监测事件E是否发生，其监测行为必然会占用部分系统资源，因此，开发人员必须在监测频率和资源消耗间取得平衡。如果监测频率太低，虽然减少了资源消耗，但是系统实时反应性就会降低。如果进行代码层的调优，就需要优化监测方法的实现以及求得一个最为恰当的监测频率。

而若将此问题预留在设计层解决，便可以使用事件通知的方式将系统行为进行倒置。如使用第2章中提到的观察者模式，在事件E发生的时刻，由事件E通知组件A，从而触发组件A的行为。这种设计方法弃用了存在性能隐患的循环监控，从根本上解决了这一问题。

从某种程度上说，设计优化直接决定了系统的整体品质。如果在设计层考虑不周，留下太多问题隐患，那么这些"质"上的问题，也许无法再通过代码层的优化进行弥补。因此，开发人员必须在软件设计之初，认真仔细考虑软件系统的性能问题。

进行设计优化时，设计人员必须熟悉常用的软件设计方法、设计模式、基本性能组件和常用优化思想，并将其有机地集成在软件系统中。

注意：一个良好的系统设计可以规避很多潜在的性能问题。因此，尽可能多花些时间在系统设计上，是创建高性能程序的关键。

#### 算法调优

算法非常重要，好的算法会有更好的性能。举几个例子，大家可以感觉一下。

一个是过滤算法。系统需要对收到的请求做过滤，我们把可以被filter in/out的东西配置在了一个文件中，原有的过滤算法是遍历过滤配置，后来，我们找到了一种方法可以对这个过滤配置进行排序，这样就可以用二分折半的方法来过滤，系统性能增加了50%。

　　一个是哈希算法。计算哈希算法的函数并不高效，一方面是计算太费时，另一方面是碰撞太高，碰撞高了就跟单向链表一个性能（可参看Hash Collision DoS 问题）。我们知道，算法都是和需要处理的数据很有关系的，就算是被大家所嘲笑的“冒泡排序”在某些情况下（大多数数据是排好序的）其效率会高于所有的排序算法。哈希算法也一样，广为人知的哈希算法都是用英文字典做测试，但是我们的业务在数据有其特殊性，所以，对于还需要根据自己的数据来挑选适合的哈希算法。

　　分而治之和预处理。如某程序为了生成月报表，每次都需要计算很长的时间，有时候需要花将近一整天的时间。于是我们把我们找到了一种方法可以把这个算法发成增量式的，也就是说我每天都把当天的数据计算好了后和前一天的报表合并，这样可以大大的节省计算时间，每天的数据计算量只需要20分钟，但是如果我要算整个月的，系统则需要10个小时以上（SQL语句在大数据量面前性能成级数性下降）。这种分而治之的思路在大数据面前对性能有很帮助，就像merge排序一样。SQL语句和数据库的性能优化也是这一策略，如：使用嵌套式的Select而不是笛卡尔积的Select，使用视图，等等。

#### 代码调优

代码调优是在软件开发过程中，或者在软件开发完成后，软件维护过程中进行的对程序代码的改进和优化。代码优化涉及诸多编码技巧，需要开发人员熟悉相关语言的API，并在合适的场景中正确使用相关API或类库。同时，对算法、数据结构的灵活使用，也是代码优化的重要内容。

虽然代码优化是从微观上对性能进行调整，但是一个"好"的实现和一个"坏"的实现对系统的影响也是非常大的。比如，同样作为List的实现，LinkedList和ArrayList在随机访问上的性能却可以相差几个数量级；又如，同样是文件读写的实现，使用Stream方式与Java NIO的方式，其性能可能又会相差一个数量级。

因此，虽然与设计优化相比，这里将代码优化称为在微观层面上的优化，但是它却是对系统性能产生最直接影响的优化方法。

#### JVM调优

由于Java软件总是运行在JVM虚拟机之上，对JVM虚拟机进行优化也能在一定程度上提升Java程序的性能。JVM调优通常可以在软件开发后期进行，如在软件开发完成，或者在软件开发的某一里程碑阶段。

作为Java软件的运行平台，JVM的各项参数将会直接影响Java程序的性能。比如，JVM的堆大小、垃圾回收策略等。

要进行JVM层面的调优，需要开发人员对JVM的运行原理和基本内存结构有一定了解。如，堆内存的结构、GC的种类等。然后，依据应用程序的特点，设置合理的JVM启动参数。

#### 网络调优

　　关于网络调优，尤其是TCP Tuning，这里面有很多很多东西可以说。看看Linux下TCP/IP的那么多参数就知道了。强烈建议大家看看《TCP/IP详解卷1:协议》这本书。在这里只讲一些概念上的东西。

　　A)TCP调优

　　我们知道TCP链接是有很多开销的，一个是会占用文件描述符，另一个是会开缓存，一般来说一个系统可以支持的TCP链接数是有限的，我们需要清楚地认识到TCP链接对系统的开销是很大的。正是因为TCP是耗资源的，所以，很多攻击都是让你系统上出现大量的TCP链接，把你的系统资源耗尽。比如著名的SYNC Flood攻击。所以，我们要注意配置KeepAlive参数，这个参数的意思是定义一个时间，如果链接上没有数据传输，系统会在这个时间发一个包，如果没有收到回应，那么TCP就认为链接断了，然后就会把链接关闭，这样可以回收系统资源开销。（注：HTTP层上也有KeepAlive参数）对于像HTTP这样的短链接，设置一个1-2分钟的keepalive非常重要。这可以在一定程度上防止DoS攻击。有下面几个参数（下面这些参数的值仅供参考）：

　　net.ipv4.tcp\_keepalive\_probes = 5

　　net.ipv4.tcp\_keepalive\_intvl = 20

　　net.ipv4.tcp\_fin\_timeout = 30

　　对于TCP的TIME\_WAIT这个状态，主动关闭的一方进入TIME\_WAIT状态，TIME\_WAIT状态将持续2个MSL(Max Segment Lifetime)，默认为4分钟，TIME\_WAIT状态下的资源不能回收。有大量的TIME\_WAIT链接的情况一般是在HTTP服务器上。对此，有两个参数需要注意，

　　net.ipv4.tcp\_tw\_reuse=1

　　net.ipv4.tcp\_tw\_recycle=1

　　前者表示重用TIME\_WAIT，后者表示回收TIME\_WAIT的资源。

　　TCP还有一个重要的概念叫RWIN（TCP Receive Window Size），这个东西的意思是，一个TCP链接在没有向Sender发出ack时可以接收到的最大的数据包。为什么这个很重要？因为如果Sender没有收到Receiver发过来ack，Sender就会停止发送数据并会等一段时间，如果超时，那么就会重传。这就是为什么TCP链接是可靠链接的原因。重传还不是最严重的，如果有丢包发生的话，TCP的带宽使用率会马上受到影响（会盲目减半），再丢包，再减半，然后如果不丢包了，就逐步恢复。相关参数如下：

　　net.core.wmem\_default = 8388608

　　net.core.rmem\_default = 8388608

　　net.core.rmem\_max = 16777216

　　net.core.wmem\_max = 16777216

　　一般来说，理论上的RWIN应该设置成：吞吐量\*回路时间。Sender端的buffer应该和RWIN有一样的大小，因为Sender端发送完数据后要等Receiver端确认，如果网络延时很大，buffer过小了，确认的次数就会多，于是性能就不高，对网络的利用率也就不高了。也就是说，对于延迟大的网络，我们需要大的buffer，这样可以少一点ack，多一些数据，对于响应快一点的网络，可以少一些buffer。因为，如果有丢包（没有收到ack），buffer过大可能会有问题，因为这会让TCP重传所有的数据，反而影响网络性能。（当然，网络差的情况下，就别玩什么高性能了）所以，高性能的网络重要的是要让网络丢包率非常非常地小（基本上是用在LAN里），如果网络基本是可信的，这样用大一点的buffer会有更好的网络传输性能（来来回回太多太影响性能了）。

　　另外，我们想一想，如果网络质量非常好，基本不丢包，而业务上我们不怕偶尔丢几个包，如果是这样的话，那么，我们为什么不用速度更快的UDP呢？你想过这个问题了吗？

　　B)UDP调优

　　说到UDP的调优，有一些事我想重点说一样，那就是MTU——最大传输单元（其实这对TCP也一样，因为这是链路层上的东西）。所谓最大传输单元，你可以想像成是公路上的公交车，假设一个公交车可以最多坐70人，带宽就像是公路的车道数一样，如果一条路上最多可以容下100辆公交车，那意味着我最多可以运送7000人，但是如果公交车坐不满，比如平均每辆车只有20人，那么我只运送了2000人，于是我公路资源（带宽资源）就被浪费了。所以，我们对于一个UDP的包，我们要尽量地让他大到MTU的最大尺寸再往网络上传，这样可以最大化带宽利用率。对于这个MTU，以太网是1500字节，光纤是4352字节，802.11无线网是7981。但是，当我们用TCP/UDP发包的时候，我们的有效负载Payload要低于这个值，因为IP协议会加上20个字节，UDP会加上8个字节（TCP加的更多），所以，一般来说，你的一个UDP包的最大应该是1500-8-20=1472，这是你的数据的大小。当然，如果你用光纤的话，这个值就可以更大一些。（顺便说一下，对于某些NB的千光以态网网卡来说，在网卡上，网卡硬件如果发现你的包的大小超过了MTU，其会帮你做fragment，到了目标端又会帮你做重组，这就不需要你在程序中处理了）

　　再多说一下，使用Socket编程的时候，你可以使用setsockopt() 设置SO\_SNDBUF/SO\_RCVBUF的大小，TTL和KeepAlive这些关键的设置，当然，还有很多，具体你可以查看一下Socket的手册。

　　最后说一点，UDP还有一个最大的好处是multi-cast多播，这个技术对于你需要在内网里通知多台结点时非常方便和高效。而且，多播这种技术对于机会的水平扩展（需要增加机器来侦听多播信息）也很有利。

　　C）网卡调优

　　对于网卡，我们也是可以调优的，这对于千兆以及网网卡非常必要，在Linux下，我们可以用ifconfig查看网上的统计信息，如果我们看到overrun上有数据，我们就可能需要调整一下txqueuelen的尺寸（一般默认为1000），我们可以调大一些，如：ifconfig eth0 txqueuelen 5000。Linux下还有一个命令叫：ethtool可以用于设置网卡的缓冲区大小。在Windows下，我们可以在网卡适配器中的高级选项卡中调整相关的参数（如：Receive Buffers, Transmit Buffer等，不同的网卡有不同的参数）。把Buffer调大对于需要大数据量的网络传输非常有效。

　　D）其它网络性能

　　关于多路复用技术，也就是用一个线程来管理所有的TCP链接，有三个系统调用要重点注意：一个是select，这个系统调用只支持上限1024个链接，第二个是poll，其可以突破1024的限制，但是select和poll本质上是使用的轮询机制，轮询机制在链接多的时候性能很差，因主是O(n)的算法，所以，epoll出现了，epoll是操作系统内核支持的，仅当在链接活跃时，操作系统才会callback，这是由操作系统通知触发的，但其只有Linux Kernel 2.6以后才支持（准确说是2.5.44中引入的），当然，如果所有的链接都是活跃的，过多的使用epoll\_ctl可能会比轮询的方式还影响性能，不过影响的不大。

　　另外，关于一些和DNS Lookup的系统调用要小心，比如：gethostbyaddr/gethostbyname，这个函数可能会相当的费时，因为其要到网络上去找域名，因为DNS的递归查询，会导致严重超时，而又不能通过设置什么参数来设置time out，对此你可以通过配置hosts文件来加快速度，或是自己在内存中管理对应表，在程序启动时查好，而不要在运行时每次都查。另外，在多线程下面，gethostbyname会一个更严重的问题，就是如果有一个线程的gethostbyname发生阻塞，其它线程都会在gethostbyname处发生阻塞，这个比较变态，要小心。这种到网上找信息的东西很多，比如，如果你的Linux使用了NIS，或是NFS，某些用户或文件相关的系统调用就很慢，所以要小心。

#### 系统调优

　　A）I/O模型

　　前面说到过select/poll/epoll这三个系统调用，我们都知道，Unix/Linux下把所有的设备都当成文件来进行I/O，所以，那三个操作更应该算是I/O相关的系统调用。说到I/O模型，这对于我们的I/O性能相当重要，我们知道，Unix/Linux经典的I/O方式是（关于Linux下的I/O模型，大家可以读一下这篇文章《使用异步I/O大大提高性能》）：

　　第一种，同步阻塞式I/O，这个不说了。

　　第二种，同步无阻塞方式。其通过fctnl设置O\_NONBLOCK来完成。

　　第三种，对于select/poll/epoll这三个是I/O不阻塞，但是在事件上阻塞，算是：I/O异步，事件同步的调用。

　　第四种，AIO方式。这种I/O模型是一种处理与I/O并行的模型。I/O请求会立即返回，说明请求已经成功发起了。在后台完成I/O操作时，向应用程序发起通知，通知有两种方式：一种是产生一个信号，另一种是执行一个基于线程的回调函数来完成这次I/O处理过程。

　　第四种因为没有任何的阻塞，无论是I/O上，还是事件通知上，所以，其可以让你充分地利用CPU，比起第二种同步无阻塞好处就是，第二种要你一遍一遍地去轮询。Nginx之所所以高效，是其使用了epoll和AIO的方式来进行I/O的。

　　当然，真正提高I/O性能方式是把和外设的I/O的次数降到最低，最好没有，所以，对于读来说，内存cache通常可以从质上提升性能，因为内存比外设快太多了。对于写来说，cache住要写的数据，少写几次，但是cache带来的问题就是实时性的问题，也就是latency会变大，我们需要在写的次数上和相应上做权衡。

　　B）多核CPU调优

　　关于CPU的多核技术，我们知道，CPU0是很关键的，如果0号CPU被用得过狠的话，别的CPU性能也会下降，因为CPU0是有调整功能的，所以，我们不能任由操作系统负载均衡，因为我们自己更了解自己的程序，所以，我们可以手动地为其分配CPU核，而不会过多地占用CPU0，或是让我们关键进程和一堆别的进程挤在一起。

　　对于Linux来说，可以使用taskset命令来设置（你可以通过安装schedutils来安装这个命令：apt-get install schedutils）

　　多核CPU还有一个技术叫NUMA技术（Non-Uniform Memory Access）。传统的多核运算是使用SMP(Symmetric Multi-Processor )模式，多个处理器共享一个集中的存储器和I/O总线。于是就会出现一致存储器访问的问题，一致性通常意味着性能问题。NUMA模式下，处理器被划分成多个node，每个node有自己的本地存储器空间。关于NUMA的一些技术细节，你可以查看一下这篇文章《Linux的NUMA技术》，在Linux下，对NUMA调优的命令是：numactl 。如下面的命令：（指定命令“myprogram arg1 arg2”运行在node 0上，其内存分配在node 0 和1上）

　　numactl --cpubind=0 --membind=0,1 myprogram arg1 arg2

　　当然，上面这个命令并不好，因为内存跨越了两个node，这非常不好。最好的方式是只让程序访问和自己运行一样的node，如：

　　$ numactl --membind 1 --cpunodebind 1 --localalloc myapplication

　　C）文件系统调优

　　关于文件系统，因为文件系统也是有cache的，所以，为了让文件系统有最大的性能。首要的事情就是分配足够大的内存，这个非常关键，在Linux下可以使用free命令来查看 free/used/buffers/cached，理想来说，buffers和cached应该有40%左右。然后是一个快速的硬盘控制器，SCSI会好很多。最快的是Intel SSD固态硬盘，速度超快，但是写次数有限。

　　接下来，我们就可以调优文件系统配置了，对于Linux的Ext3/4来说，几乎在所有情况下都有所帮助的一个参数是关闭文件系统访问时间，在/etc/fstab下看看你的文件系统有没有noatime参数（一般来说应该有），还有一个是dealloc，它可以让系统在最后时刻决定写入文件发生时使用哪个块，可优化这个写入程序。还要注间一下三种日志模式：data=journal、data=ordered和data=writeback。默认设置data=ordered提供性能和防护之间的最佳平衡。

　　当然，对于这些来说，ext4的默认设置基本上是最佳优化了。

　　其它还有一些关于NFS、XFS的调优，大家可以上google搜索一些相关优化的文章看看。关于各文件系统，大家可以看一下这篇文章——《Linux日志文件系统及性能分析》。

#### 数据库调优

对绝大部分应用系统而言，数据库是必不可少的一部分。Java程序可以使用JDBC的方式连接数据库。对数据库的调优可以分为3个部分：在应用层对SQL语句进行优化；对数据库进行优化；对数据库软件进行优化。

数据库调优是一个很大的话题，下面的这些东西并不一定正确，因为在不同的业务场景，不同的数据库设计下可能会得到完全相反的结论，所以，在这里做一些一般性的说明，具体问题还要具体分析。

　　A）数据库引擎调优

　　数据库的锁的方式。这个非常非常地重要。并发情况下，锁是非常非常影响性能的。各种隔离级别，行锁，表锁，页锁，读写锁，事务锁，以及各种写优先还是读优先机制。性能最高的是不要锁，所以，分库分表，冗余数据，减少一致性事务处理，可以有效地提高性能。NoSQL就是牺牲了一致性和事务处理，并冗余数据，从而达到了分布式和高性能。

　　数据库的存储机制。不但要搞清楚各种类型字段是怎么存储的，更重要的是数据库的数据存储方式，是怎么分区的，是怎么管理的，比如Oracle的数据文件，表空间，段，等等。了解清楚这个机制可以减轻很多的I/O负载。比如：MySQL下使用show engines;可以看到各种存储引擎的支持。不同的存储引擎有不同的侧重点，针对不同的业务或数据库设计会让你有不同的性能。

　　数据库的分布式策略。最简单的就是复制或镜像，需要了解分布式的一致性算法，或是主主同步，主从同步。通过了解这种技术的机理可以做到数据库级别的水平扩展。

　　B）SQL语句优化

　　关于SQL语句的优化，首先也是要使用工具，比如：MySQL SQL Query Analyzer，Oracle SQL Performance Analyzer，或是微软SQL Query Analyzer，基本上来说，所有的RMDB都会有这样的工具，来让你查看你的应用中的SQL的性能问题。 还可以使用explain来看看SQL语句最终Execution Plan会是什么样的。

　　还有一点很重要，数据库的各种操作需要大量的内存，所以服务器的内存要够，优其应对那些多表查询的SQL语句，那是相当的耗内存。

　　下面我根据我有限的数据库SQL的知识说几个会有性能问题的SQL：

　　全表检索。比如：select \* from user where lastname = “xxxx”，这样的SQL语句基本上是全表查找，线性复杂度O(n)，记录数越多，性能也越差（如：100条记录的查找要50ms，一百万条记录需要5分钟）。对于这种情况，我们可以有两种方法提高性能：一种方法是分表，把记录数降下来，另一种方法是建索引（为lastname建索引）。索引就像是key-value的数据结构一样，key就是where后面的字段，value就是物理行号，对索引的搜索复杂度是基本上是O(log(n)) ——用B-Tree实现索引（如：100条记录的查找要50ms，一百万条记录需要100ms）。

　　索引。对于索引字段，最好不要在字段上做计算、类型转换、函数、空值判断、字段连接操作，这些操作都会破坏索引原本的性能。当然，索引一般都出现在Where或是Order by字句中，所以对Where和Order by子句中的子段最好不要进行计算操作，或是加上什么NOT之类的，或是使用什么函数。

　　多表查询。关系型数据库最多的操作就是多表查询，多表查询主要有三个关键字，EXISTS，IN和JOIN（关于各种join，可以参看图解SQL的Join一文）。基本来说，现代的数据引擎对SQL语句优化得都挺好的，JOIN和IN/EXISTS在结果上有些不同，但性能基本上都差不多。有人说，EXISTS的性能要好于IN，IN的性能要好于JOIN，我各人觉得，这个还要看你的数据、schema和SQL语句的复杂度，对于一般的简单的情况来说，都差不多，所以千万不要使用过多的嵌套，千万不要让你的SQL太复杂，宁可使用几个简单的SQL也不要使用一个巨大无比的嵌套N级的SQL。还有人说，如果两个表的数据量差不多，Exists的性能可能会高于In，In可能会高于Join，如果这两个表一大一小，那么子查询中，Exists用大表，In则用小表。这个，我没有验证过，放在这里让大家讨论吧。另，有一篇关于SQL Server的文章大家可以看看《IN vs JOIN vs EXISTS》

　　JOIN操作。有人说，Join表的顺序会影响性能，只要Join的结果集是一样，性能和join的次序无关。因为后台的数据库引擎会帮我们优化的。Join有三种实现算法，嵌套循环，排序归并，和Hash式的Join。（MySQL只支持第一种）

　　（1）嵌套循环，就好像是我们常见的多重嵌套循环。注意，前面的索引说过，数据库的索引查找算法用的是B-Tree，这是O(log(n))的算法，所以，整个算法复法度应该是O(log(n)) \* O(log(m))这样的。

　　（2）Hash式的Join，主要解决嵌套循环的O(log(n))的复杂，使用一个临时的hash表来标记。

　　（3）排序归并，意思是两个表按照查询字段排好序，然后再合并。当然，索引字段一般是排好序的。

　　还是那句话，具体要看什么样的数据，什么样的SQL语句，你才知道用哪种方法是最好的。

　　部分结果集。我们知道MySQL里的Limit关键字，Oracle里的rownum，SQL Server里的Top都是在限制前几条的返回结果。这给了我们数据库引擎很多可以调优的空间。一般来说，返回top n的记录数据需要我们使用order by，注意在这里我们需要为order by的字段建立索引。有了被建索引的order by后，会让我们的select语句的性能不会被记录数的所影响。使用这个技术，一般来说我们前台会以分页方式来显现数据，Mysql用的是OFFSET，SQL Server用的是FETCH NEXT，这种Fetch的方式其实并不好是线性复杂度，所以，如果我们能够知道order by字段的第二页的起始值，我们就可以在where语句里直接使用>=的表达式来select，这种技术叫seek，而不是fetch，seek的性能比fetch要高很多。

　　字符串。正如我前面所说的，字符串操作对性能上有非常大的恶梦，所以，能用数据的情况就用数字，比如：时间，工号，等。

　　全文检索。千万不要用Like之类的东西来做全文检索，如果要玩全文检索，可以尝试使用Sphinx。

　　其它。

　　（1）不要select \*，而是明确指出各个字段，如果有多个表，一定要在字段名前加上表名，不要让引擎去算。

　　（2）不要用Having，因为其要遍历所有的记录。性能差得不能再差。

　　（3）尽可能地使用UNION ALL 取代UNION。

　　（4）索引过多，insert和delete就会越慢。而update如果update多数索引，也会慢，但是如果只update一个，则只会影响一个索引表。

### 配置项性能调优参考

下面的推荐调优配置都是基于下面的机器配置：

|  |  |
| --- | --- |
| OS | CentOS release 6.3 (Final) |
| IP | 192.168.197.138 |
| CPU | 4 核 3000.035 MHz |
| MEMORY | 16G |
| Additional info. | 虚拟机 |

1. Linux系统调优
2. Linux limit 配置修改

|  |
| --- |
| 修改limit 配置文件，提高linux 本身最大限制，都是在文件最后添加，如果有则是修改  vi /etc/security/limits.conf  \* soft nproc 102400000  \* hard nproc 102400000  \* soft nofile 1048576  \* hard nofile 1048576  vi /etc/security/limits.d/90-nproc.conf  \* soft nproc 102400000  \* hard nproc 102400000  \* soft nofile 1048576  \* hard nofile 1048576  重启生效  service sshd restart  为每个普通用户修改limit 限制(经过尝试，两台虚拟机最大的)：  vi /etc/profile  ulimit -u 102400000  ulimit -n 1048576  ulimit -d unlimited  ulimit -m unlimited  ulimit -s unlimited  ulimit -t unlimited  ulimit -v unlimited  ulimit -c unlimited |

1. TCP linux 内核参数设置

|  |
| --- |
| 修改配置文件 /etc/sysctl.conf  vi /etc/sysctl.conf  # Controls the use of TCP syncookies  net.ipv4.tcp\_syncookies = 1  net.ipv4.tcp\_tw\_reuse = 1  net.ipv4.tcp\_tw\_recycle = 1  net.ipv4.tcp\_fin\_timeout = 5  net.ipv4.tcp\_keepalive\_time = 1200  net.ipv4.ip\_local\_port\_range = 10000 65000  net.ipv4.tcp\_max\_syn\_backlog = 262144  net.core.netdev\_max\_backlog = 262144  net.ipv4.tcp\_max\_tw\_buckets = 5000  生效(执行会报错，需要执行中间两个命令)  /sbin/sysctl -p  modprobe bridge  lsmod|grep bridge  /sbin/sysctl –p |

1. **JVM 调优**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **推荐参数：**  JVM\_ARGS=  "-Xms4000M –Xmx4000M –Xmn1000M -XX:PermSize=1000M -XX:MaxPermSize=1000M \  -XX:+UseConcMarkSweepGC-XX:MaxTenuringThreshold=7 -XX:GCTimeRatio=19 \  -Xnoclassgc -Xloggc:log/gc.log -XX:+PrintGCDetails -XX:+PrintGCTimeStamps \  -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0 \  JVM 堆内存和 GC回收设置，堆内存大小推荐机器物理内存的 1/4, GC 回收用的是CMS内存并发回收。  **堆参数解释：**   |  |  |  | | --- | --- | --- | | 参数名称  含义 | 默认值 |  | | -Xms  初始堆大小 | 物理内存的1/64(<1GB) | 默认(MinHeapFreeRatio参数可以调整)空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制. | | -Xmx  最大堆大小 | 物理内存的1/4(<1GB) | 默认(MaxHeapFreeRatio参数可以调整)空余堆内存大于70%时，JVM会减少堆直到 -Xms的最小限制 | | -Xmn  年轻代大小(1.4or lator) |  | 注意：此处的大小是（eden+ 2 survivor space).与jmap -heap中显示的New gen是不同的。 | | 整个堆大小=年轻代大小 + 年老代大小 + 持久代大小. | | 增大年轻代后,将会减小年老代大小.此值对系统性能影响较大,Sun官方推荐配置为整个堆的3/8 | | -XX:NewSize  设置年轻代大小(for 1.3/1.4) |  |  | | -XX:MaxNewSize  年轻代最大值(for 1.3/1.4) |  |  | | -XX:PermSize  设置持久代(perm gen)初始值 | 物理内存的1/64 |  | | -XX:MaxPermSize  设置持久代最大值 | 物理内存的1/4 |  | |

1. **Tomcat调优**
   1. 增加JVM堆内存大小

|  |
| --- |
| JAVA\_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8  -server –Xms4000m –Xmx4000m  -XX:NewSize=1000m -XX:MaxNewSize=1000m -XX:PermSize=1000m  -XX:MaxPermSize=1000m -XX:+DisableExplicitGC"  -Xms – 指定初始化时化的栈内存, 推荐机器物理内存的 1/4  -Xmx – 指定最大栈内存, 推荐机器物理内存的 1/4  在重启你的Tomcat服务器之后，这些配置的更改才会有效。 |

* 1. 修复JRE内存泄漏

|  |
| --- |
| <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" /> |

* 1. 线程池设置

|  |
| --- |
| <Connector port="8080" address="localhost"  **maxThreads**="250" maxHttpHeaderSize="8192"  emptySessionPath="true" protocol="HTTP/1.1"  enableLookups="false" redirectPort="8181"acceptCount="100"  connectionTimeout="20000" disableUploadTimeout="true"/> |

* 1. 压缩

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8181" **compression**="500"  compressableMimeType="text/html,text/xml,text/plain,application/octet-stream" /> |

1. **Nginx 调优**

|  |
| --- |
| worker\_processes 8;  nginx进程数，建议按照cpu数目来指定，一般为它的倍数, 2倍左右比较合适。  worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00000001 00000010 00000100 00001000;  为每个进程分配cpu，上例中将8个进程分配到4个cpu，当然可以写多个，或者将一个进程分配到多个cpu。  worker\_rlimit\_nofile 102400;  这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n的值保持一致。  use epoll;  使用epoll的I/O模型， epoll 是 linux 内核对AIO的实现。  worker\_connections 102400;  每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为worker\_processes\*worker\_connections。  keepalive\_timeout 60;  keepalive超时时间。  client\_header\_buffer\_size 4k;  客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。  open\_file\_cache max=102400 inactive=20s;  这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。  open\_file\_cache\_valid 30s;  这个是指多长时间检查一次缓存的有效信息。  open\_file\_cache\_min\_uses 1;  open\_file\_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。 |

1. **MySQLInnoDB调优**

|  |
| --- |
| #如果将back\_log设定到比你操作系统限制更高的值,将会没有效果,  #对应 Linux tcp 内核参数中的net.core.netdev\_max\_backlog  back\_log = 30000  # MySQL 服务所允许的同时会话数的上限  max\_connections = 3000  # 每个客户端连接最大的错误允许数量  max\_connect\_errors = 30  # 所有线程所打开表的数量  table\_cache = 4096  # 允许外部文件级别的锁. 打开文件锁会对性能造成负面影响  #external-locking  # 服务所能处理的请求包的最大大小以及服务所能处理的最大的请求大小  max\_allowed\_packet = 32M  # 在一个事务中binlog为了记录SQL状态所持有的cache大小  binlog\_cache\_size = 4M  # 独立的内存表所允许的最大容量.  max\_heap\_table\_size = 128M  # 排序缓冲被用来处理类似ORDER BY以及GROUP BY队列所引起的排序  sort\_buffer\_size = 16M  # 此缓冲被使用来优化全联合(full JOINs 不带索引的联合).  join\_buffer\_size = 16M  # 我们在cache中保留多少线程用于重用  thread\_cache\_size = 16  # 此值只对于支持 thread\_concurrency() 函数的系统有意义( 例如Sun [Solaris](http://www.ha97.com/category/unix/solaris" \o "Solaris)) # 推荐 [CPU数量]\*(2~4) 来作为thread\_concurrency的值 thread\_concurrency = 8  # 查询缓冲常被用来缓冲 SELECT 的结果并且在下一次同样查询的时候不再执行直接返回结果  query\_cache\_size = 128M  # 此设置用来保护查询缓冲,防止一个极大的结果集将其他所有的查询结果都覆盖.  query\_cache\_limit = 4M  # 被全文检索索引的最小的字长.  ft\_min\_word\_len = 8  # 线程使用的堆大小. 此容量的内存在每次连接时被预留.  thread\_stack = 512K |

**附录参考：**

探秘Java虚拟机——内存管理与垃圾回收：<http://sunbean.blog.51cto.com/972509/768034>

JVM参数调：<http://blog.csdn.net/chenyi8888/article/details/4720489>

（总结）MySQL my.cnf参数配置优化详解：<http://www.ha97.com/4110.html>

Tomcat 生产服务器性能优化：<http://www.oschina.net/translate/tomcat-performance-tuning>

# 性能测试模板

## COD系统性能测试报告范例

**1.概述**

**1.1测试对象**

1. COD前置查询接口
2. COD前置确认接口
3. COD消费接口
4. COD前置撤销接口
5. COD前置退款接口

1.2测试环境

|  |  |
| --- | --- |
| 软件名称 | 版本 |
| Red Hat | 2.6.32-279.el6.x86\_64 |
| ORACLE | 11g |
| JDK | jdk1.6.0\_25 |
| TOMCAT | Apache Tomcat/7.0.42（COD前置）  Apache Tomcat/6.0.33(交易) |

|  |  |
| --- | --- |
| **服务器** | |
| CPU | AMD Opteron(TM) Processor 6220 |
| 内存 | 16G |
| 网络连接 | 局域网 |
| IP地址 | 192.168.195.167 |
| 负载端（2台PC服务器） | |
| 操作系统 | Windows 7 企业版64位 |
| 测试工具 | LoadRunner11 |



**1.3测试模型**

根据业务模型同比缩小后作为本次测试模型。

|  |  |  |  |
| --- | --- | --- | --- |
| 系统名称 | 编号 | 交易名称 | 本业务交易占比 |
| COD项目 | 1 | 消费 | 30 |
| 2 | COD前置查询 | 30 |
| 3 | COD前置确认 | 30 |
| 4 | COD前置撤销 | 5 |
| 5 | COD前置退款 | 5 |

**1.4测试指标**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试指标  系统名称 | 系统交易处理能力  （TPS：笔/秒） | 平均响应时间  （TRT：秒） | 系统资源使用率 | 交易失败率%  （不包括返回错误的正常交易） | 本次（Q4）全国日交易量:51（万） | 预计明年6-18日交易量300（万） |
| 消费 | 35 | 2 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 76.5 | 450 |
| COD前置查询 | 23 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 51 | 300 |
| COD前置确认 | 23 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 51 | 300 |
| COD前置撤销 | 0.5 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 1 | 1 |
| COD前置退款 | 0.5 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 1 | 1 |

说明：预期目标是根据Q4日交易量下进行测试，明年6-18的目标只是个预期的展望、通过测试可以比对与预期的差距。本次检测系统TPS能否达到47，如果达不到，将找到系统TPS稳定的情况下的最大值。

TPS是根据：2、8 原则估算出来的。  20%的时间80%的业务

例如510000\*0.8/（24\*0.2\*3600）=23/S

**1.5测试前的参数调优**

**ulimit：**

core file size (blocks, -c) unlimited

data seg size (kbytes, -d) unlimited

scheduling priority (-e) 0

file size (blocks, -f) unlimited

pending signals (-i) 124397

max locked memory (kbytes, -l) 64

max memory size (kbytes, -m) unlimited

open files (-n) 1048576

pipe size (512 bytes, -p) 8

POSIX message queues (bytes, -q) 819200

real-time priority (-r) 0

stack size (kbytes, -s) unlimited

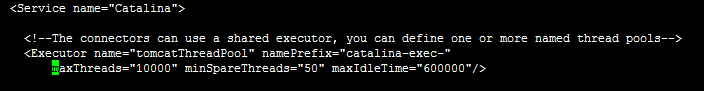
cpu time (seconds, -t) unlimited

max user processes (-u) 102400000

virtual memory (kbytes, -v) unlimited

file locks (-x) unlimited

**tomcat线程池：**



**2测试方法**

**2.1负载测试**

**测试目的：**容量测试是针对多支交易，在被测系统测试模型下逐步增加并发用户数进行负载测试，获取各系统处理的最优性能。

**测试方法：**

1. 按照并发梯度加压的方法，逐步提升对被测系统的压力；
2. 直到tps不再上升、系统出现容量拐点或瓶颈后结束负载测试，获取其系统最优处理性能；
3. 观察应用服务器与数据库服务器CPU、内存及磁盘的占用情况；

**2.2疲劳强度测试（混合场景）**

**测试目的：**

检测系统长时间在高负载情况下的性能表现。

**测试方法：**

1. 按照并发梯度加压的方法，逐步提升对被测系统的压力；
2. 当系统达到较高的负载时（最高负载的80%左右），不再提升压力，保持此压力较长时间（10个小时）。
3. 观察被测系统CPU、内存及磁盘的占用情况；

**3.测试结果**

总体情况

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 测试项 | 用户数 | 执行时间 | TPS(笔/秒) | 平均响应时间(秒） | 成功数 | 失败数 | 成功率(%) | 备注 |
| 查询接口 | 100 | 15min | 51 | 1.1 | 67645 | 18 | 99 |  |
| 70 | 15min | 61.2 | 1.26 | 59827 | 0 | 100 |  |
| 50 | 15min | 45 | 1.1 | 45345 | 0 | 100 |  |
| 确认接口 | 50 | 15min | 36.4 | 1.4 | 36453 | 0 | 100 |  |
| 30 | 15min | 26.1 | 1.2 | 23882 | 0 | 100 |  |
| 70 | 15min | 50.3 | 2 | 37028 | 0 | 100 |  |
| 100 | 15min | 60 | 1.28 | 80524 | 0 | 100 | 用户再增加的时候，tps骤降 |
| 撤销接口 | 50 | 15min | 41.9 | 1.36 | 37438 | 0 | 100 |  |
| 70 | 15min | 55.2 | 1.23 | 61520 | 0 | 100 |  |
| 100 | 15min | 34.8 | 1.23 | 52052 | 2 | 100 |  |
| 退款接口 | 50 | 15min | 45.4 | 1.1 | 46233 | 0 | 100 |  |
| 70 | 15min | 7 | 1 | 28397 | 0 | 100 | 10分钟的时候tps骤降 |
| 消费接口 | 10 | 15min | 5 | 1.96 | 4603 | 0 | 100 |  |
| 15 | 15min | 4.9 | 3 | 4652 | 0 | 100 |  |
| 混合 | 50 | 7.5小时 | 查询：1.094确认：1.107撤销：1.095退款：1.087消费：2.317 |  | 1010381 | 709 | 99 | 50vuser。消费30%、查询30%、确认30%，退款5%撤销5% |

应用服务器情况：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **测试项** | **用户数** | **cpu%（平均）** | **cpu%（最大）** | **服务器load（最大）** | **FullGC次数** | **Disk（Iowait）** | **SWAP** | **thread blocked** | **jconsole内存图** | **备注** |
| 查询接口 | 50 | 12.8 | 50 | 2.03 | 0 | 0 |  | 无 | 正常 |  |
| 70 | 20.4 | 100 | 5.94 | 0 | 0 | 0 | 无 | 正常 | CPU最大100%。平均负载过高，应用服务器成为瓶颈 |
| 100 | 17.5 | 39 | 3.74 | 0 | 0 |  | 无 | 正常 |  |
| 确认接口 | 50 | 15.6 | 81.7 | 3.09 | 0 | 0 | 0 | 无 | 正常 |  |
| 30 | 10.9 | 96.3 | 0.54 | 0 | 0 | 0 | 无 | 正常 |  |
| 70 | 13.1 | 92 | 1.05 | 0 | 0 | 0 | 无 | 正常 |  |
| 100 | 22.1 | 100 | 3.18 | 0 | 0 | 0 | 无 | 正常 | 事物失败的时候是负载:2.58.比较大 |
| 撤销接口 | 50 | 34.3 | 100 | 6.4 | 0 | 0 | 0 | 无 | 正常 | 事物失败的时候是负载最大 |
| 70 | 10.6 | 45.3 | 4.4 | 0 | 0 | 0 | 无 | 正常 |  |
| 100 | 22.4 | 100 |  | 0 | 0 | 0 | 无 | 正常 |  |
| 退款接口 | 50 | 12.7 | 94.1 | 0.52 | 0 | 0 | 0 | 无 | 正常 |  |
| 70 | 8.9 | 25.3 | 2.69 | 0 | 0 | 0 | 无 | 正常 |  |
| 消费 | 10 | 5.2 | 31.8 | 3.98 | 1 | 0 | 0 | 无 | 正常 |  |
| 15 | 90.3 | 99.9 | 10.98 | 2 | 0 | 0 | 无 | 正常 |  |
| 混合 | 50 | 61.3 | 99.9 | 13.21 | 8 | 0 | 0 | 无 |  |  |

数据库服务器情况：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试项** | **用户数** | **cpu** | **Iowait（最大）** | **问题** |
| 查询接口 | 100 | 23 | 9.34 | 硬解析比较多，建议变量绑定 |
| 确认接口 | 100 | 28.69 | 7.59 | 硬解析比较多，建议变量绑定 |
| 撤销接口 | 100 | 53.47 | 5.61 | 无 |
| 退款接口 | 50 | 37.37 | 4.49 | 无 |
| 消费接口 | 15 | 34.08 | 1.63 | T\_POS\_ORDER走的是全表扫描 |
| 混合 | 50 | 61.4 | 2.99 | 压到7个多小时的时候，后台报错，具体见后面的分析 |

说明:

1、用户加载方式：每分钟加载10个用户

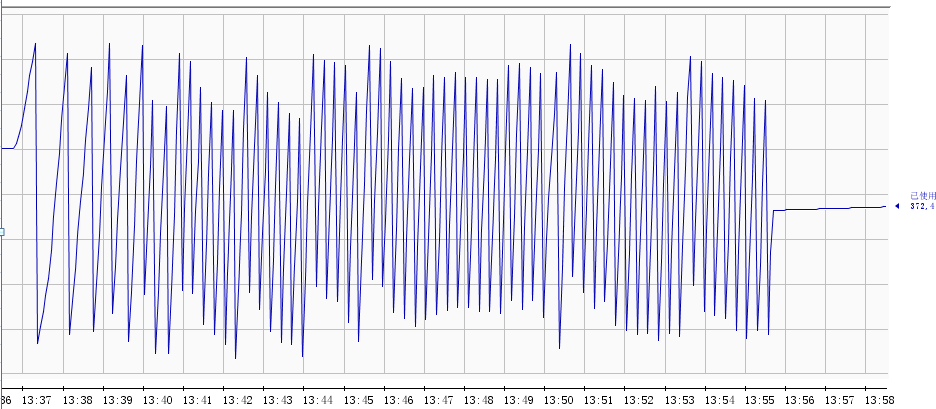
2、使用2台压力机，且压力机负载不高

3、单场景每个接口tps最高的时候。cpu高、负载大，服务瓶颈。

**4.测试结果分析**

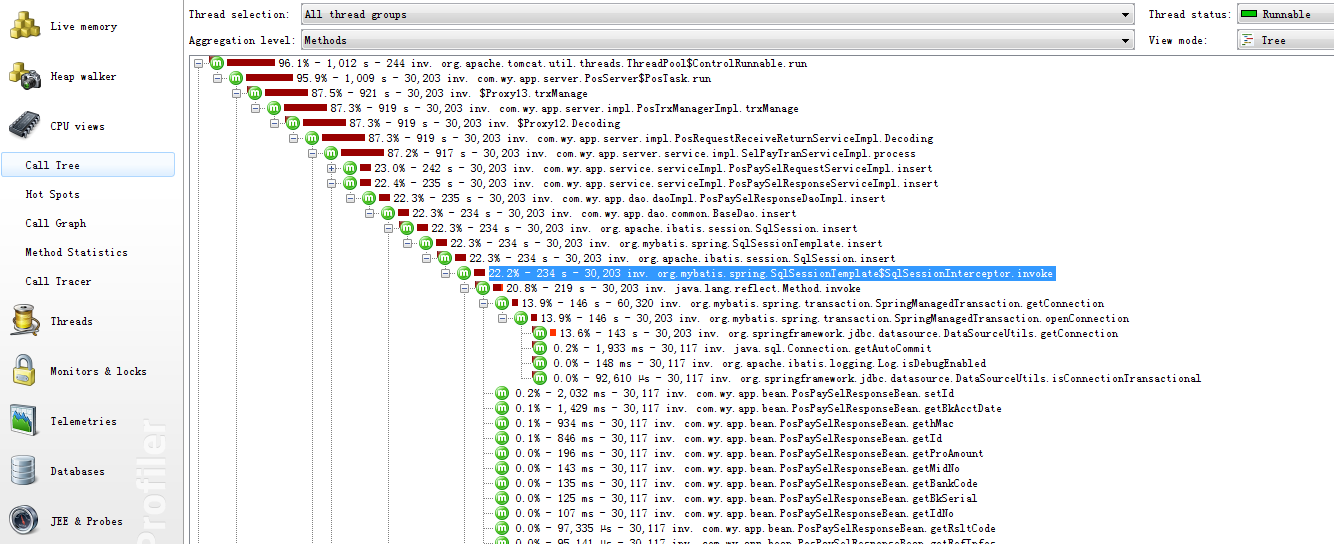
**4.1查询接口**

**jconsole图：**

****

**结论：JVM回收彻底，无Full GC。内存正常。**

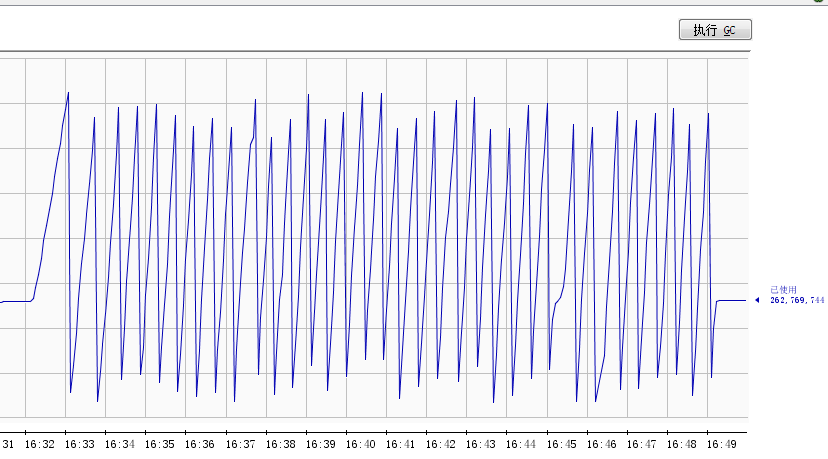
**jprofiler图（最消耗cpu方法占比）：**

****

分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding。开发进行调优的时候可以参考此图。

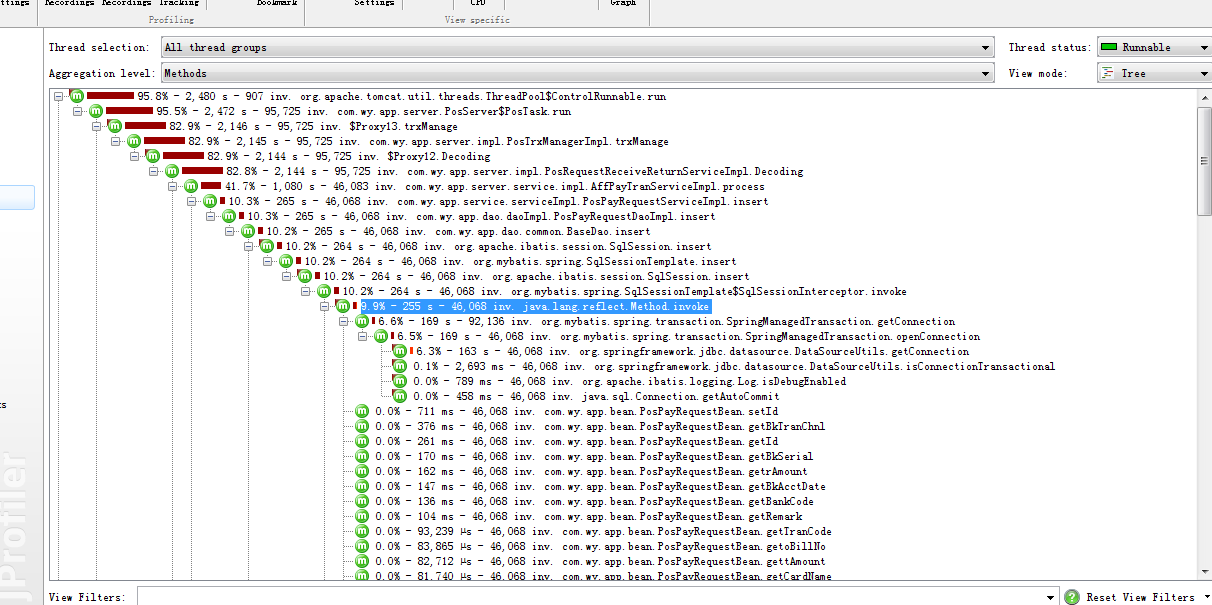
**4.2确认接口**

**jconsole图：**

****

**结论：JVM回收彻底，无Full GC。内存正常。**

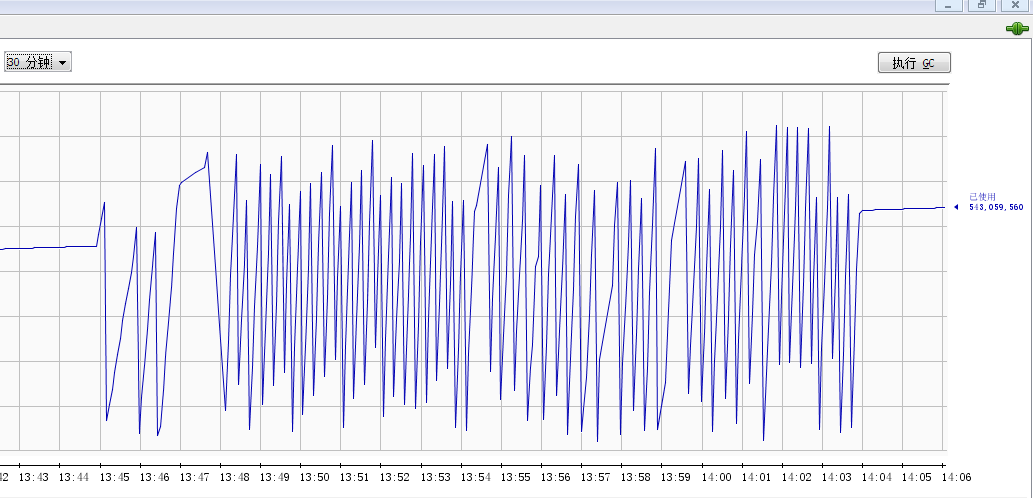
**jprofiler图（最消耗cpu方法占比）：**

****

分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding、com.wy.app.server.service.impl.AffPayTranServiceImpl.process。开发进行调优的时候可以参考此图。

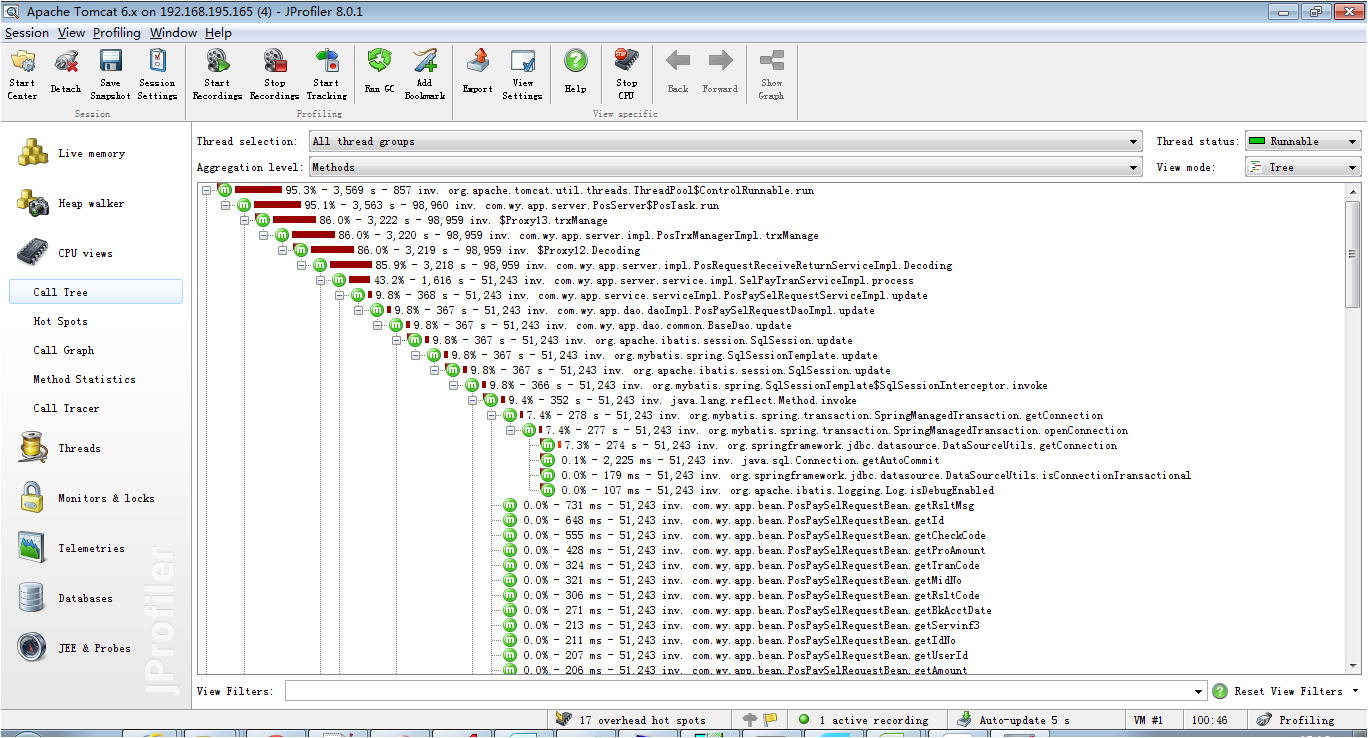
**4.3撤销接口**

**jconsole图：**

****

**结论：JVM回收彻底，无Full GC。内存正常。**

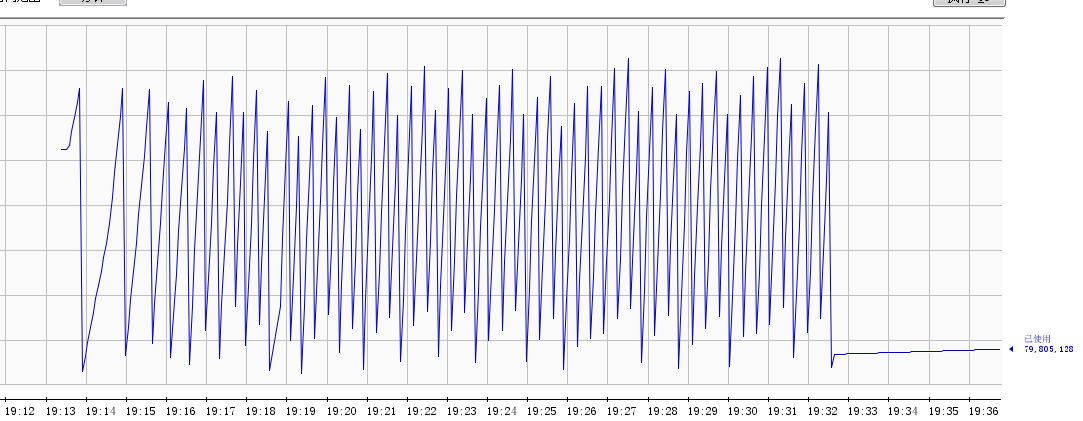
**jprofiler图（最消耗cpu方法占比）：**

****

分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding、com.wy.app.server.service.impl.SelPayTranServiceImpl.process开发进行调优的时候可以参考此图。

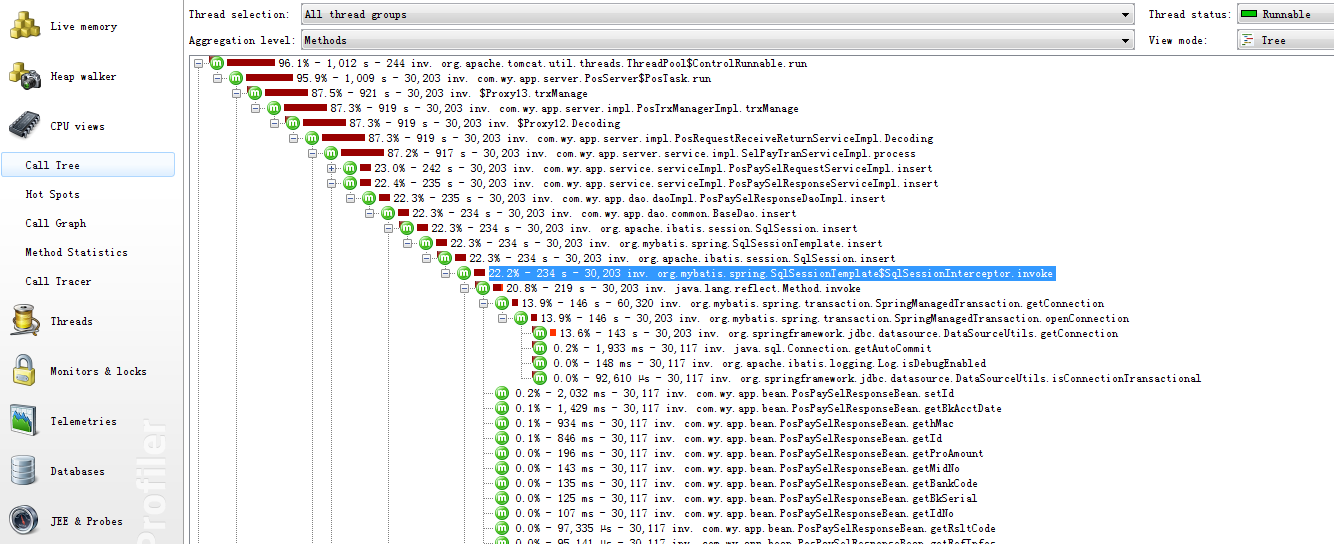
**4.4退款接口**

**jconsole图：**

****

**结论：JVM回收彻底，无Full GC。内存正常。**

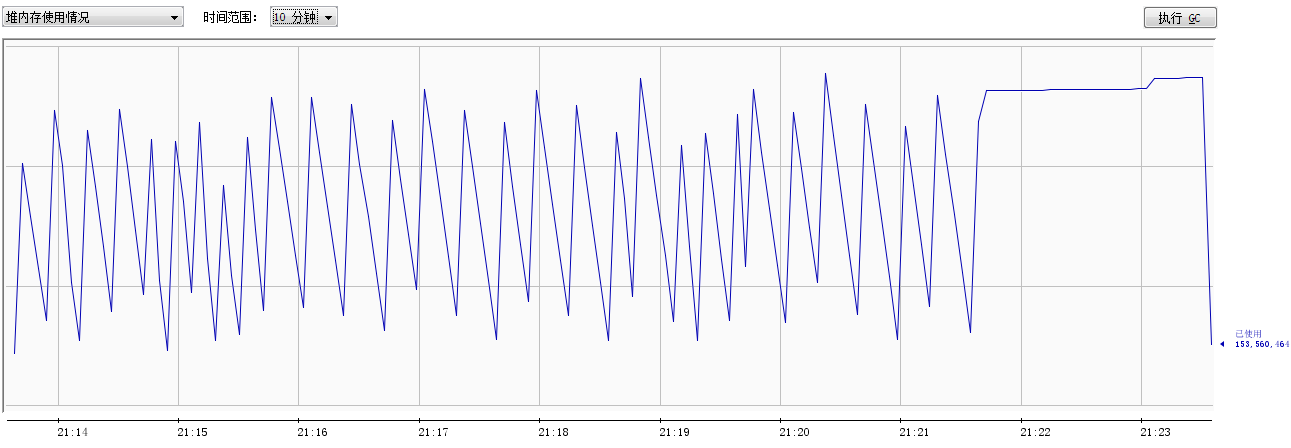
**jprofiler图（最消耗cpu方法占比）：**

****

分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding、com.wy.app.server.service.impl.BackPayTranServiceImpl.process、com.wy.app.server.service.impl.BackPayTranServiceImpl.processPayBack。开发进行调优的时候可以参考此图。

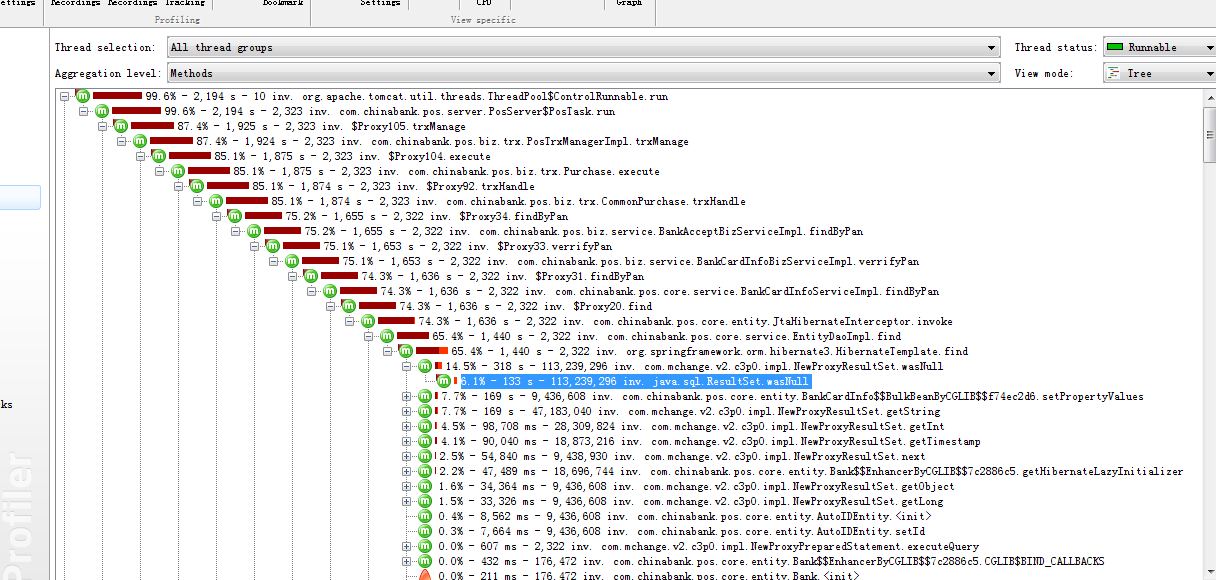
**4.5消费接口**

**jconsole图：**

****

**结论：JVM回收彻底，无Full GC。内存正常。**

**jprofiler图（最消耗cpu方法占比）：**

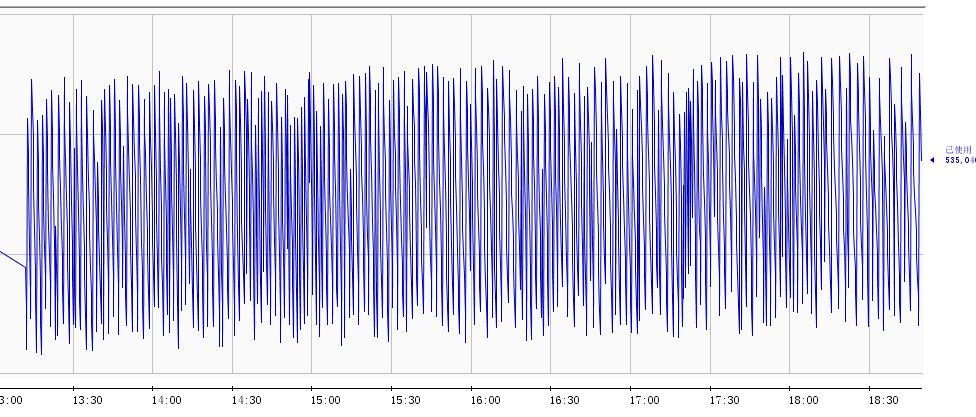
****

分析：cpu占比比较高的方法为：com.chinabank.pos.server.PosSERVER..PosTask.run、com.chinabank.pos.biz.trx.PosTrxManagerImpl.trxManage、com.chinabank.pos.biz.trx.Purchase.execute、com.chinabank.pos.biz.trx.CommonPurchase.trxHandle、com.chinabank.pos.biz.service.BankAcceptBizServiceImpl.findByPan。开发进行调优的时候可以参考此图。

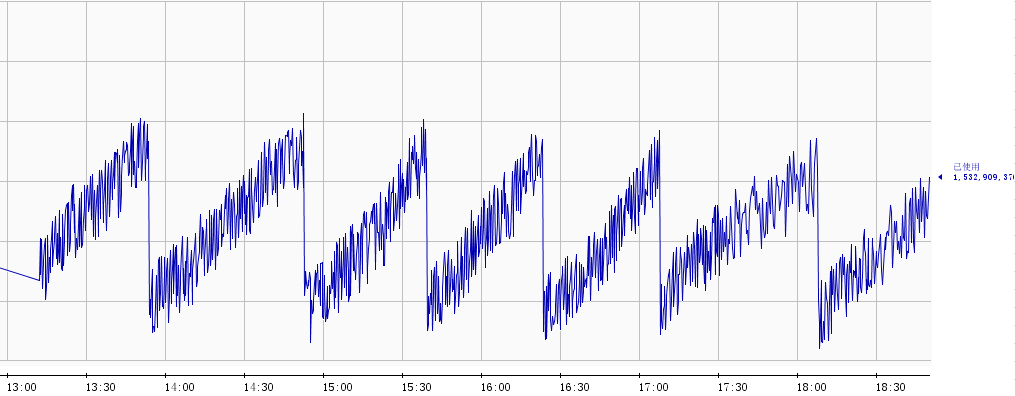
**4.6混合场景接口**

**jconsole图：**

**COD:**

****

**消费：**

****

**结论：cod前置接口：JVM回收彻底，无Full GC。内存正常。**

**消费接口：JVM回收彻底，8次Full GC，属于高负载，1个小时左右一次Full GC。**

**后台问题：在压到7个多小时的时候后台报错：**

### Error updating database. Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Could not get JDBC Connection; nested exception is java.sql.SQLException: Listener refused the connection with the following error:

ORA-12516, TNS:listener could not find available handler with matching protocol stack

The Connection descriptor used by the client was:

192.168.201.41:1521/payment

### Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Could not get JDBC Connection; nested exception is java.sql.SQLException: Listener refused the connection with the following error:

ORA-12516, TNS:listener could not find available handler with matching protocol stack

The Connection descriptor used by the client was:

192.168.201.41:1521/payment

atorg.mybatis.spring.MyBatisExceptionTranslator.translateExceptionIfPossible(MyBatisExceptionTranslator.java:73)

at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:365)

at $Proxy7.update(Unknown Source)

at org.mybatis.spring.SqlSessionTemplate.update(SqlSessionTemplate.java:251)

at com.wy.app.dao.common.BaseDao.update(BaseDao.java:49)

at com.wy.app.dao.daoImpl.PosPaySelRequestDaoImpl.update(PosPaySelRequestDaoImpl.java:22)

at com.wy.app.service.serviceImpl.PosPaySelRequestServiceImpl.update(PosPaySelRequestServiceImpl.java:25)

at com.wy.app.server.service.impl.SelPayTranServiceImpl.process(SelPayTranServiceImpl.java:138)

at com.wy.app.server.impl.PosRequestReceiveReturnServiceImpl.Decoding(PosRequestReceiveReturnServiceImpl.java:49)

at sun.reflect.GeneratedMethodAccessor92.invoke(Unknown Source)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)

at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:198)

at $Proxy12.Decoding(Unknown Source)

at com.wy.app.server.impl.PosTrxManagerImpl.trxManage(PosTrxManagerImpl.java:18)

at sun.reflect.GeneratedMethodAccessor89.invoke(Unknown Source)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)

at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:198)

at $Proxy13.trxManage(Unknown Source)

at com.wy.app.server.PosServer$PosTask.run(PosServer.java:85)

at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:687)

at java.lang.Thread.run(Thread.java:662)

Caused by: org.apache.ibatis.exceptions.PersistenceException:

报该错后，tps急剧下降，最终为0，但服务没有宕掉。

**5.测试结论**

1、查询接口支持的最大tps：61.2。响应时间：1.26。支持最大虚拟并发用户数：70。

确认接口支持的最大tps：60。响应时间：1.28。支持最大虚拟用户数：100。

撤销接口支持的最大tps：55.2。响应时间：1.23。支持最大虚拟用户数：70。

退款接口支持的最大tps：45.4。响应时间：1.1。支持最大虚拟用户数：50。

消费接口支持的最大tps：5。响应时间：1.96。支持最大虚拟用户数：10。

2、COD前置查询、确认、撤销、退款接口测试能达到Q4，距明年618相差甚远。但本次依据的是Q4,这4个接口---通过；消费tps（5）达不到预期目标（35）---不通过；

3、混合场景在稳定性测试，报数据库连接异常（COD部分）具体报错见4.6。是否因为没使用数据库连接池所导致？---混合场景测试不通过。

4、建议优化

a、T\_POS\_ORDER表走的是全表扫描，建议加索引。

b、测试查询、确认接口的时候，数据库走的几乎是硬解析，建议变量绑定。

c、消耗CPU较高的方法也建议优化，如4。

# 性能测试工具

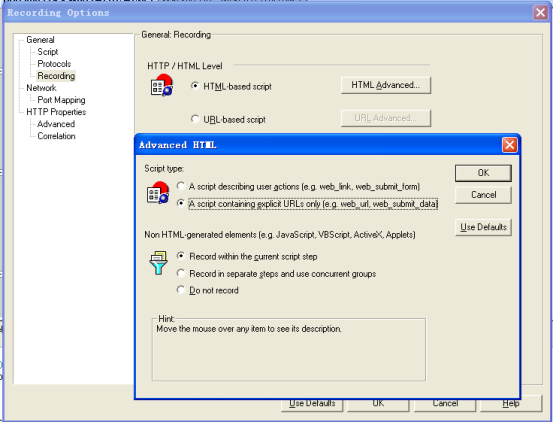
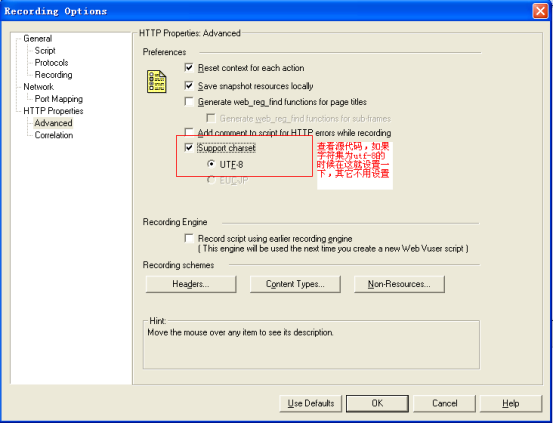
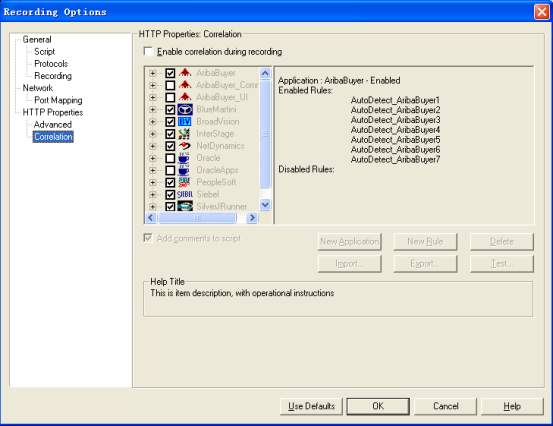
## Loadrunner

### loadrunner简介

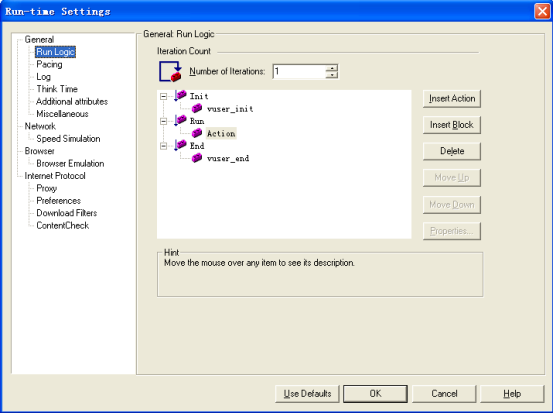
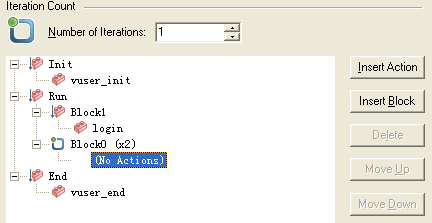
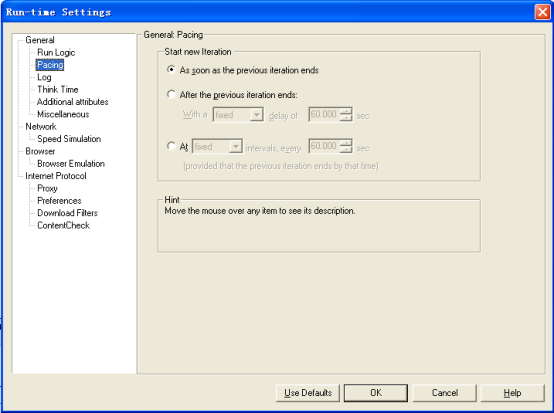
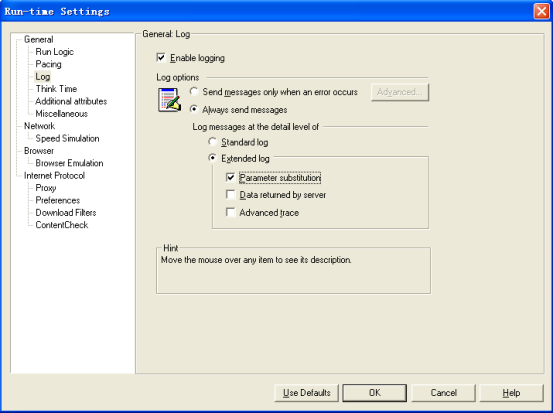
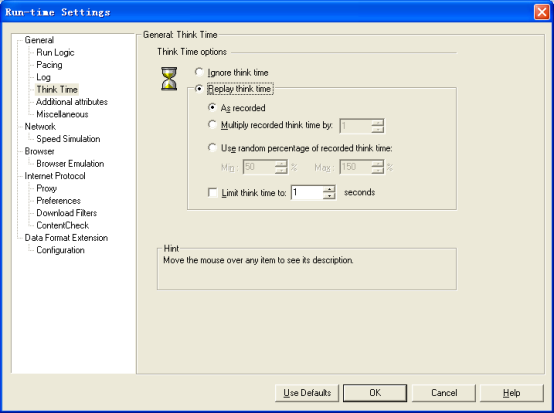
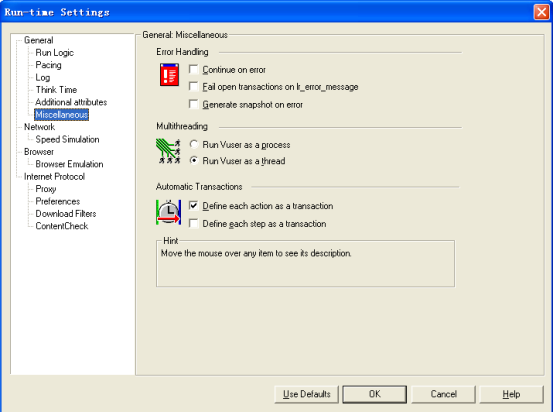
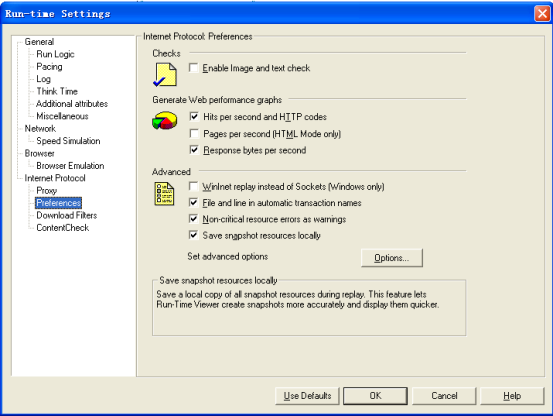
LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试。通过使用LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。企业的网络应用环境都必须支持大量用户，网络体系架构中含各类应用环境且由不同供应商提供软件和硬件产品。难以预知的用户负载和愈来愈复杂的应用环境使公司时时担心会发生用户响应速度过慢，系统崩溃等问题。这些都不可避免地导致公司收益的损失。Mercury Interactive 的 LoadRunner 能让企业保护自己的收入来源，无需购置额外硬件而最大限度地利用现有的IT 资源，并确保终端用户在应用系统的各个环节中对其测试应用的质量，可靠性和可扩展性都有良好的评价。LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来帮助您更快的查找和发现问题。此外，LoadRunner 能支持广范的协议和技术，为您的特殊环境提供特殊的解决方案。

### 录制及调试脚本

#### 录制脚本前的设置：

  
一般默认选择下面的。  
web URL 函数根据精确连接地址来找的。它不依赖于上下文。  
web link 类似QTP 先找文字再根据文字找连接。当所点击的字不在的时候，它就找不到了。它依赖于上下文  
  
  
关联，一般不做自动关联，即不选中Enable correlation

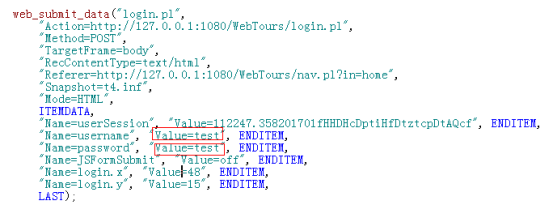
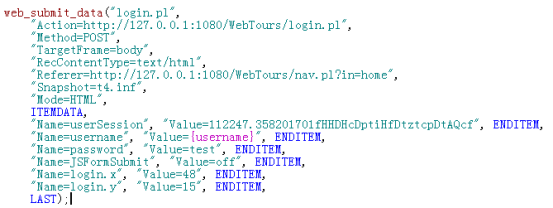
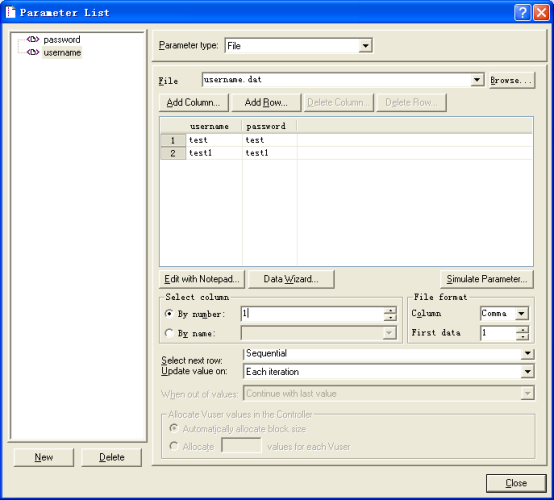
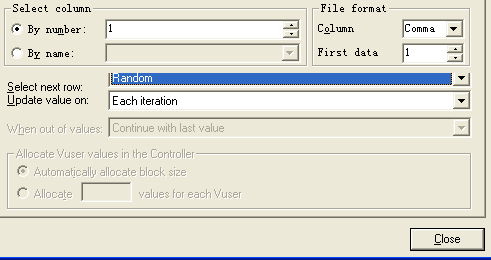
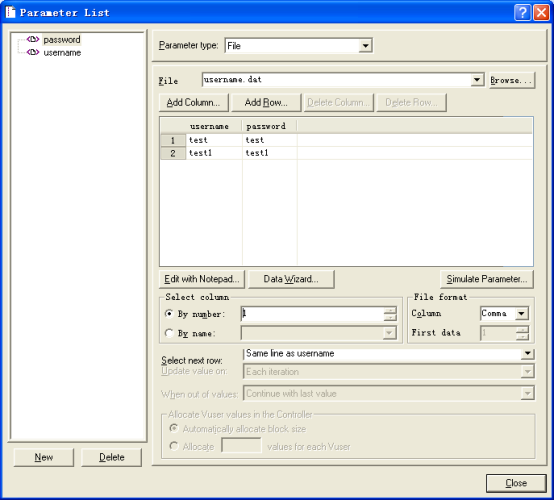
#### Run-time Setting设置

设置迭代：  
  
说明：  
Insert Action：通过插入action，可以把不同的业务放在不同的action中。只是设置迭代次数的时候，所有的action的迭代次数是相同的。  
Insert Block：如果需要把不同的业务设置成不同的迭代次数，只需要加入块,然后在块中设置迭代次数。  
  
  
http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav3ea9f5d90688b869ba6beb601c23276a.png?version=1&modificationDate=1388992794827&api=v2 ：前一次迭代一完成就进行这次迭代。  
http://10.45.241.159:8080/confluence/download/attachments/1969298/worddavb011499513384b3fb030608d0677bb92.png?version=1&modificationDate=1388992794835&api=v2 ：前一次迭代60秒后，进行下一次迭代  
http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav16b8a807535fa460651a6d6cb72a035a.png?version=1&modificationDate=1388992794845&api=v2 ：每60秒迭代一次。前一次循环不一定完成，只要达到指定时间，就进入下一次循环中去。  
设置Pacing的目的：LoadRunner 模拟客户端向服务器发出请求，必须等待服务器对这个请求做出响应，并且客户端收到这个响应之后，才会重新发出新的请求，而服务器对请求的处理是需要一个时间的。测试场景以迭代的方式进行，如果不设置步进值的话，那么对于每个虚拟用户来说，每一个发到服务器的请求得到响应之后，会马上发送下一次请求。如果此时的服务器端的排队队列已满，服务器资源正处于忙碌的状态，那么该请求会驻留在服务器的线程中，换句话说，这个新产生的请求并不会对服务器端产生真正的负载，但该请求的计时器已经启动了，因此我们很容易就可以预见到，这个请求的响应时间会变得很长，甚至可能长到使得该请求由于超时而失败。等到测试结束后，我们查看一下结果，就会发现：事务平均响应时间很长，最小响应时间与最大响应时间的差距很大，而这个时候的平均响应时间，其实也就失去了它应有的意义。也就是说，由于客户端发送的请求太快而导致影响了实际的测量结果。  
因此，为了解决这个问题，我们可以在每两个请求之间插入一个间隔时间，这将会降低单个用户启动请求的速度。间歇会减少请求在线程中驻留的时间，从而提供更符合现实的响应时间。   
  
如果做参数化的时候，可以选择扩展日志，有利于调试脚本,但是在场景中要尽量关闭扩展日志，比较耗性能。（有些用到关联的时候，找左右边界，可以使用扩展日志）。  
  
Ignore think times：忽略思考时间，回放脚本时忽略所有的lr\_think\_times函数。  
Replay think time:  
按录制参数：回放期间，使用lr\_think\_times函数中显示的参数。来等待多长时间，单位秒。  
录制思考时间乘以：回放期间，使用lr\_think\_times函数中思考时间的倍数。  
使用录制思考时间的随机百分比：通过指定思考时间的范围，可以设置思考时间值的范围。  
将思考时间限制为：限制思考时间的最大值。如果此处设置的值与脚本中lr\_think\_times的值冲突时，以此处设置的时间为准。  
  
如果未选中Define each action as a transaction 在场景中就不计算init和action中的事务的时间，只计算自己加进去事务的时间。  
  
Options设置超时。报"Error -27728: Step download timeout (120 seconds)"错的时候在此处设置。如：HTTP-request connection timeout(sec)、 HTTP-request receive timeout(sec)、Step download timeout(sec)。把这些值设置大些。

#### 其它设置

Tools-> http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav623e8d305be95bae1293b6ffbdeafa2d.png?version=1&modificationDate=1388992795431&api=v2 ，可以设置场景  
Tools-> http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav81279e505556a241a9ff0373f9def015.png?version=1&modificationDate=1388992795435&api=v2  
Parameterzation 设置参数化时，左右符号，可以在这里面定义，默认为{}  
Display 选中Show browser during replay,回放的时候，显示浏览器  
Tools== http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav406dc74e61b42c56d3c7dbe1cc03e9e2.png?version=1&modificationDate=1388992795439&api=v2 ，脚本改乱的时候，可以重新生成脚本。

#### 参数化

参数化就是在脚本中用参数取代常量值。下面以登录为例。  
1、打开脚本，找到登陆的脚本代码，如下图：  
  
2、参数化用户名，选中test，右键单击并选择"Replace with a parameter(替换为新参数)"  
3 、在弹出的对话框中，输入参数名username，并点击"确定"  
4 、参数化后，如下图：  
  
5 、参数化密码，与参数化用户名过程相同，或者点击工具栏上的Param List按钮打开参数设置页面。如下图：  
http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav76d1cb3aca38d68a9847881c1538d4f7.png?version=1&modificationDate=1388992795629&api=v2  
6、把用户名和密码放在同一个.dat文件中。  
可以通过点击"Add Row"按钮增加行，输入其他可以的值，点击"Add Column（添加列）"添加其他参数。  
  
7 、我们把login这个参数作为基准，设置它的select column（选择列）保持默认（1）；Select next row（选择下一行）选择：Random；Update value on（更新值得时间）选择：Each iteration。如下图：  
  
8、设置password参数，选择username.dat，使密码和用户名使用同一个文件。select column（选择列），在这里password选择2，Select next row（选择下一行）都选择：Same line as username，意思是和username取相同的行的数据，这样就可以保证一一对应了。下面是name参数的设置，如下图：  


#### 检查点

检查点的作用是在回放脚本期间，搜索特定的文本字符串或者图片等内容，从而验证服务器响应内容的正确性。  
插入检查点的必要性：有的时候登陆操作，测试不报错，但是查看后台就报错。是没有插入检查点导致的！LoadRunner以用户名为角色向服务器发送一个登陆请求，却不会判断请求的返回消息是什么，只要有返回，即使这是个拒绝登陆的返回，LoadRunner也认为这个用户名登陆成功了，其实这个用户名登录是失败的。所以在登录或者[其他](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1969298)有重要页面跳转的地方，很有必要做检查点。  
函数：web\_find()和web\_reg\_find()。

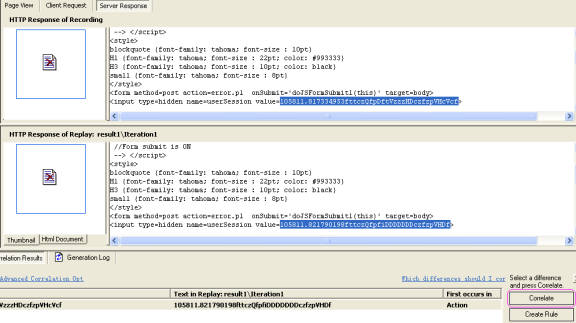
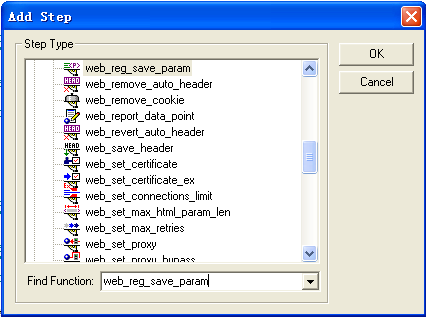
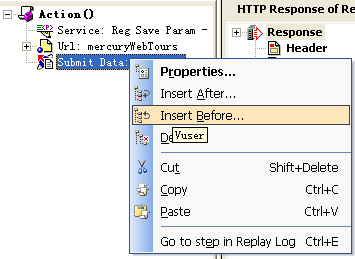
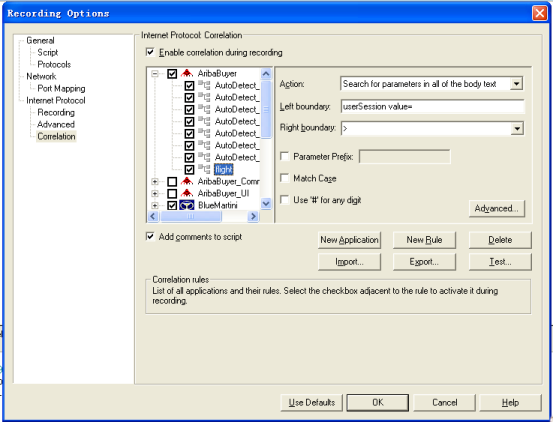
#### 集合点

集合点，测试瞬间承受多大的压力，同步虚拟用户在同一时刻执行任务。集合点放在事物之前，如果放在事物之后，可能有些虚拟用户没达到集合点就开始计时，造成事物的时间统计不正确。  
函数：lr\_rendezvous("集合点名")。

#### 事务

事物是要度量其服务器响应时间的任务或操作集。事物由开始、结束或多个业务操作/任务构成。事物必须成对出现，并且事物名称要相同。事物中不要将思考时间加进去，否则在回放思考时间设置被允许的情况下，思考时间将被算入事物的执行时间，从而影响了对事物正确执行时间分析与统计。  
如图：  

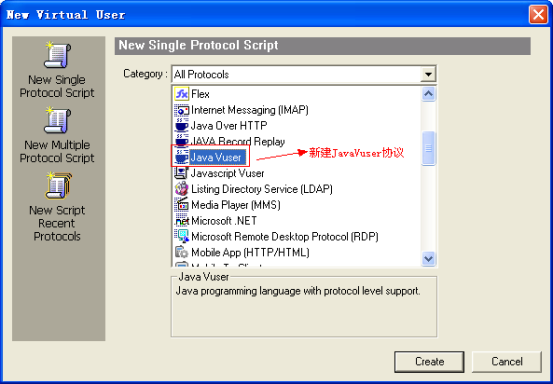
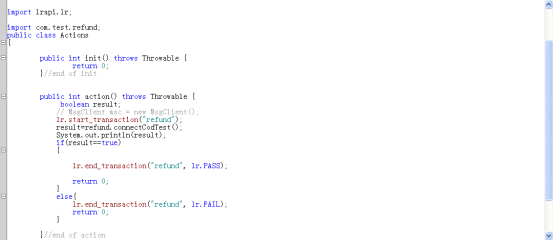
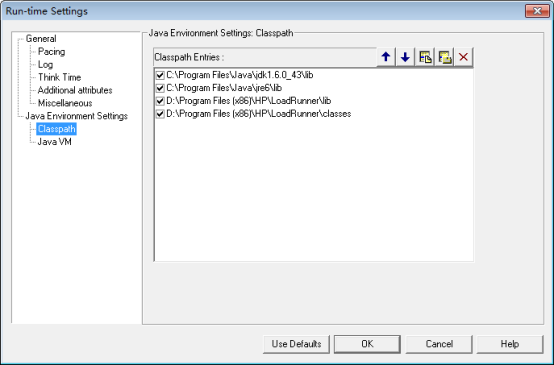

#### 关联

在脚本回放过程中，客户端发出请求，通过关联函数所定义的左右边界值（也就是关联规则），在服务器所响应的内容中查找，得到相应的值，以变量的形式替换录制时的静态值，从而向服务器发出正确的请求，这种动态获得服务器响应内容的方法被称作关联。客户端的某个请求是随着服务器端的相应而动态变化的时候，我们就需要用到关联。  
例如：登录操作、先查后修改，先查后删除。关联有3种方式：自动关联、手动关联、设置关联规则  
**自动关联**Vuser==Scan script for correlations  
1、回放该脚本至少执行一次。  
2、点Scan script for correlations关联之后，脚本中自动生成web\_reg\_save\_param方法（切换到脚本视图）  
  
**手动关联**  
录制两个相同的脚本，通过Tools==compare with Vuser  
找出2次脚本发生变化的地方，找出它的左右边界，通过关联函数，进行关联，最后再把变化的值进行参数化。  
手动关联（一）  
通过上面的操作，在脚本视图下插入函数  
Insert==new stept   
  
手动关联—（二）  
在tree视图下插入函数  
  
手动关联—（三）  
在脚本中手动写关联函数  
web\_reg\_save\_param("sid",  
"LB=userSession value=","RB=>",LAST);  
web\_reg\_save\_param 为注册函数，应该放在请求之前  
**自定义关联规则**  
录制之前定好关联规则：  
Recording Options  


#### 思考时间

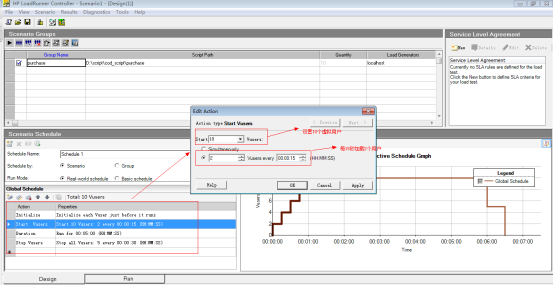
用户在执行两个连续操作期间等待的时间称为思考时间。Loadrunner在录制脚本时，VuGen将录制实际的停留等待时间，并将相应的等待时间插入到脚本。  
函数：lr\_think\_time(2）;  
思考时间可以通过运行时设置来决定是否启用思考时间，具体策略参见Run-time Setting中的thinktime

#### JavaVuser协议例子

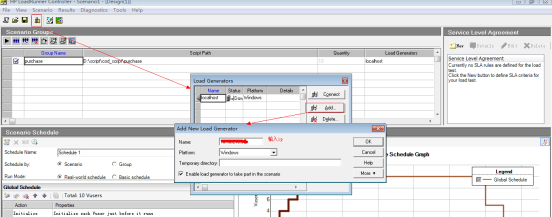
1、新建JavaVuser协议  
  
2、调试脚本  
  
3、配置环境变量  
Run-time Settings->Java Environment Settings中的Classpath，如图：  
  
4、把需要引用的包放到脚本的根目录中。

### 场景设置

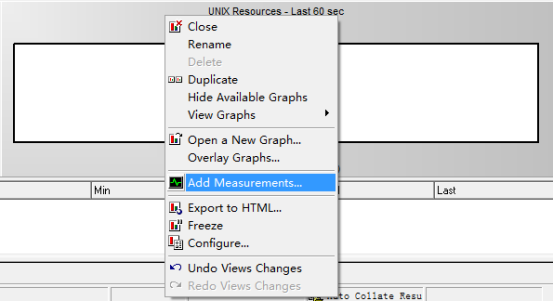
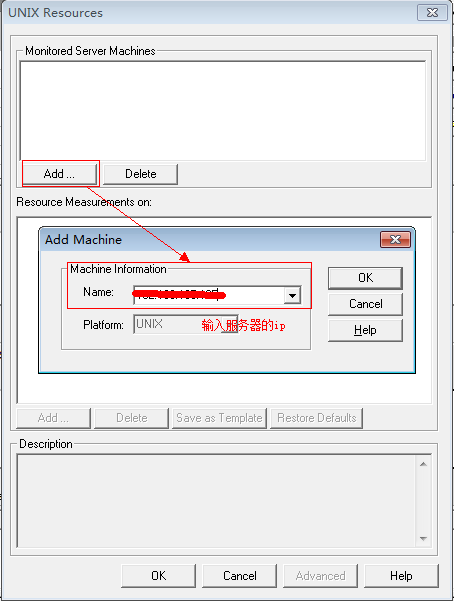
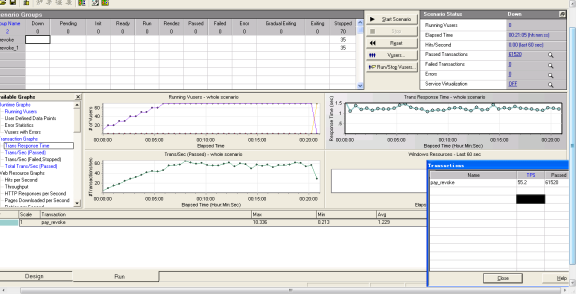
#### 如何设置并发用户数



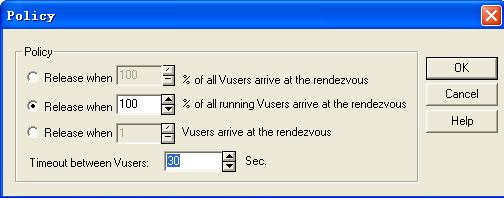
#### 设置压力机

  
注意:压力机要开启代理。

#### 监控系统资源情况

Run栏中  
  
  
点击 http://10.45.241.159:8080/confluence/download/attachments/1969298/worddav816ef4bd65cf26f652b6cac577574b4c.png?version=1&modificationDate=1388992797278&api=v2 开始运行场景  
场景运行中  


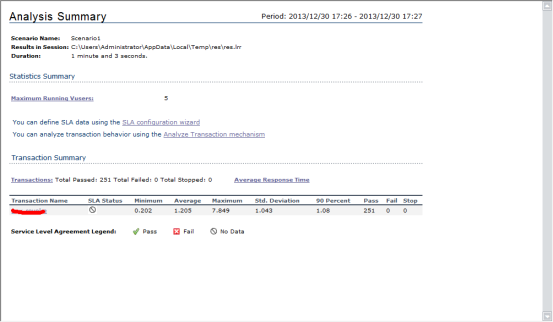
#### 集合点策略

  
1、所有用户到达集合点的时候释放  
2、百分之多少的正在运行的用户达到集合点时候释放  
3、多少个用户到达集合点的时候释放

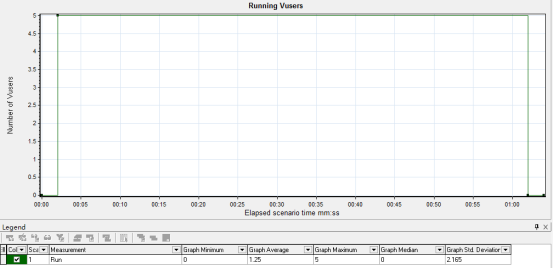
### 结果分析

场景运行完之后，点击 http://10.45.241.159:8080/confluence/download/attachments/1969298/worddava586d29d9dbe4e2f357d7f836e13e0dc.png?version=1&modificationDate=1388992797508&api=v2 产生报告

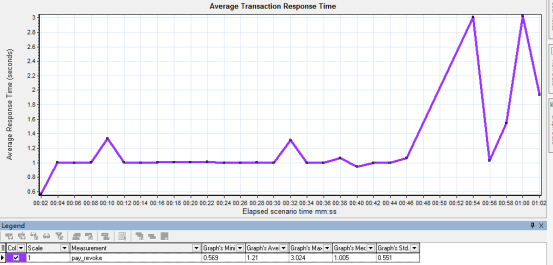
#### Analysis Summary

  
描述场景的基本信息、状态摘要信息、事物摘要信息等。通过Average和Std.deviation可以看出场景执行过程的稳定性。90Percent为90%的事物响应时间有多长。

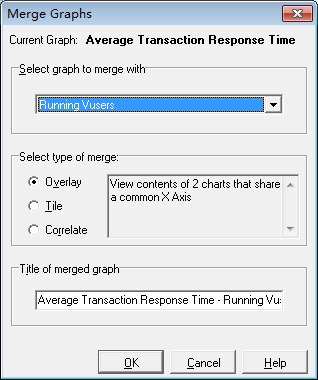
#### Running Vuser

  
说明虚拟用户加载规则，如Ramp up、Ramp down，持续多长时间。它描述场景的负载、即Vusers数量。  
Vusers数量会对吞吐量，点击率及被测服务器造成直接压力。当Vusers数量或者TPS没有达到需求时，需要增加Vusers数量。

#### Transaction Response Time(Percentile)

  
显示事物的响应时间。

#### 合并图

在图标中右键Merge Graphs  
  
通过该方法可以对各个图标进行任意合并。

# 参考文档

loadrunner 学习笔记--Web Services

<http://www.cnblogs.com/spo22/archive/2010/06/17/1759795.html>

loadrunner 参数化 Unique Number

<http://blog.csdn.net/gzh0222/article/details/6982863>

web应用性能优化经验总结

<http://www.cnblogs.com/itfriend/p/4414856.html>

深入理解Loadrunner中的Browser Emulation

<http://www.blogjava.net/tacy/archive/2007/11/06/158397.html>

webservice、socket、http 小记（一）

<http://blog.csdn.net/m_123hj_520/article/details/9370723>

负载测试、压力测试和性能测试的区别

<http://www.cnblogs.com/ITwangy/archive/2010/02/24/1672077.html>

[日志级别的选择：Debug、Info、Warn、Error还是Fatal](http://fengshen-xia.iteye.com/blog/295961)

<http://blog.csdn.net/houpengfei111/article/details/7395393>

## ms常见问题

1.测试时发现最严重或棘手的问题是什么？

2.测试时你认为哪一步最重要？

3.测试时发现的哪些问题印象最深？