**LoadRunner性能测试学习笔记**

1. **性能测试基本概念**
2. **性能测试：**模拟真实的生产环境，以各种不同的压力（模拟大量用户）去测试被测系统、去"攻击"测试系统。同时记录下被测系统中 各台 服务器的各种重要资源情况，包括cpu、内存、磁盘和网络等资源。
3. **性能测试的目的？**

识别系统中的弱点、评估系统能力、进行系统调优，提高系统的可靠性、稳定性。

1. **在具备什么条件下可以开展性能测试工作。**

答：功能测试通过；一般需要进行性能测试的系统，大多是用户量比较大、业务使用比较频繁、对响应时间要求较高、比较重要的功能模块。（**注意：**性能测试之前要做好系统备份）

1. **性能测试时首先看性能需求，如果没有需求，这时要根据与客户交流、被测系统的相关资料、以及性能测试工程师的经验，去编写测试计划，进行性能测试。**
2. **被测系统**

SUT （System Under Test）

AUT （Application Under Test）

EUT （Environment Under Test）

1. **LoadRunner工作原理：（**录制--回放的工作方式）和QTP类似
2. 录制时，LoadRunner记录下 客户端和服务器 二者之间的对话。
3. 回放时，LoadRunner模拟 真实的客户端 向服务器发起请求，并按照脚本去验证服务器的应答
4. **LoadRunner的三大组件及功能：**（三个火枪手） OALoad工具类似（触类旁通）
5. 虚拟用户脚本生成器（Virtual User Generator）VuGen VUG

功能：录制、编辑、调试测试脚本

1. 压力调度控制台 （Controller）

功能：创建场景、运行场景、监控场景、收集测试数据（场景：就是一个大型的配置文件）

1. 压力结果分析器 （Analysis）

功能：把收集到的测试数据以图表的形式展示出来，生成测试报告

1. **LoadRunner基本测试流程：**
2. 指定性能测试计划(部分) Word
3. 创建测试脚本
4. 编辑、运行测试脚本
5. 创建场景
6. 运行、监控场景，收集数据
7. 生成测试报告，分析测试结果
8. **什么是事务，为何要创建事务？**

**答：**事务分为事务的开始、结束和之间的业务操作，事务用于度量服务器性能的。（事务响应时间）

我们可以对比较关心的某个或某些业务操作，设定为一个事务，LR会记录不同事务的响应时间。

1. **请求响应时间=客户端时间+网络时间+服务器时间**
2. **负载测试和压力测试的区别：** （国内混用，国外有差别，笔试时需要注意）
3. 共同点都是在测试过程中逐步加压
4. 负载测试：强调系统正常工作情况下的性能指标； Load Testing（见好就收）

压力测试：目的是发现在什么条件下系统的性能变得不可接受，发现应用程序性能下降的拐点； Stress Testing（使劲折腾）

举例：一座大桥，桥上写最大载重量的车辆，不超过60吨

但是在桥梁内部建筑资料，最大载重量，不超过70吨

1. **吞吐量和点击率的概念、区别？**
2. 吞吐量（Throughput）:用户从服务器端获得全部数据量，单位是字节(Byte)。

吞吐量/传输时间，就是吞吐率，是服务器每秒处理的数据量。

1. 点击率（Hits per Second）:客户端每秒向服务器提交Http请求数。（鼠标的一次点击，请求数可能为n个）

说明：吞吐量是总量，是累计时间内全部数据量。

吞吐率反映服务器的处理速度和性能，也是衡量网络性能的重要指标。点击率越大，对服务器的压力也越大。

1. **并发测试和在线测试的区别？**
2. 并发和在线的区别：并发的压力是一种瞬时压力，在线的压力是一段时间的压力。
3. 20用户并发的压力相当于200用户在线的压力。（1:10比例）

在写测试计划时作为参考依据。2000用户在线，设计为200用户并发。（并发操作：查询、登录、删除、添加）

1. **QTP和LoadRunner的区别：**
2. QTP: 功能测试工具 （自动化）

LR: 性能测试工具 可以测多用户

1. QTP关心的是界面(UI)，关心的是对象（对象库的概念）；

LR只关心客户端和服务器之间的数据包（请求包、应答包），不关心对象，更不需要比对对象的属性值，只关心抓包（捕捉数据包）。

如果用户界面变了，但是业务逻辑不变：QTP脚本需要变化，LR脚本不需改变。

1. LR关心的是客户端和服务器之间的对话，前提是选择正确的网络协议（相当于网络的语言）。
2. LR不能补录。录制失败，从头再来。

注意：录制过程中出现失误，该次录制作废，从New开始重新录制；

录制时要慢，等待页面资源下载完毕后再进行下一步操作。

1. **性能测试的策略**

**重要的：基准测试、并发测试、综合场景测试 （前3个项目必备）**

**极限测试、递增测试**

**次要的：疲劳强度测试（大型系统中）、内存泄露测试、**

**数据容量测试。**

**共同点：向被测系统发起攻击**

1. 基准测试：就是单用户测试（重点）

**注意：**还是需要使用控制台，运行场景，自动搜集数据，通过Analysis进行结果分析。

1. **递增测试：**每隔一定的时间(1s,5s,10s)逐步加载虚拟用户，逐步加压。

**用途：**登录测试时，可以递增测试

1. **并发测试：**多用户并发执行某一操作（同一时刻，LR精确到毫秒级别）。

**注意：**并发测试是一种严格的测试，主要考察系统对瞬时较大压力的承受能力。

1. **综合场景测试：**

**概念：**号称“能够最真实的模拟 实际生产环境”。

**综合场景的几个要素：**多用户、多个脚本（至少3个）、在线执行一段时间（1个小时、50分钟等）

**注意：**一般不需要设置并发点。

多用户一起运行，一定会有并发。

**比如：100用户在线综合场景：**

100用户 共同对被测系统执行操作，其中30用户执行浏览首页操作，50用户执行查询订单操作，20用户执行提交订单操作。（要真实模拟人数比例）

**问题：为什么不模拟大量的登录操作？**

因为用户不可能一直在登录，模拟真实情况。

以上操作，用户在循环执行。

1. **响应时间：**业内一般有“358原则”，系统响应时间在3秒以内，则用户能够接受；响应时间在5秒以内，用户能够忍受；响应时间超过8秒，用户不能忍受。

比如：一般需求指标，不超过3秒

1. **疲劳强度测试：**在一定的强度（压力）下，对系统进行长时间的性能测试，一般为7\*24小时、或24小时、12小时等。

比如：银行系统，7\*24\*365 全天候不间断运行

**考察疲劳强度测试时，要考察其平均响应时间，以及各台服务器的各项资源情况。**

比如：集群 负载均衡、降低成本

1. **内存泄露检查：**通过正常的性能测试，如果被测系统的内存曲线走势不正常，则关注其相应的各项重要的内存指标，通过对应走势来确定是否发生内存泄露。
2. **数据容量测试：**使用大容量的数据添加到数据库中，观察被测系统是否能够正常运行。

比如：向数据库中添加200G数据量，再进行测试，甚至几个T大数据，一般是T级、P级的数据量

1024Byte = 1KB

1024K = 1M

1024M = 1G

1024G = 1T

1024T = 1P

1. **极限测试：**使用并发测试、在线测试等方法，测试出系统能够承受的极限压力（如最大用户数），或系统能够达到的最大处理能力（如最大吞吐量）。测试方法可以采用递增测试，比如对系统进行100用户、500用户、1000用户等测试。（也称为：摸高测试）
2. **三大基本测试（基准测试，并发测试和综合场景测试）的具体方法及配置**
3. **归纳基准测试：**

**方法1：单用户循环5次**

1. 调试好脚本（加检查点，在VuGen中运行成功）
2. 打开控制台，设置Run-time Settings
3. 迭代次数：5
4. Pacing值：随机2~3 （每次迭代之间的时间间隔）
5. Think time: 忽略 （请求之间的时间间隔）

忽略的原因：单用户对系统压力较小，忽略与否对结果影响不大。

**方法2：单用户持续运行1分钟**

1. 调试好脚本（加检查点，在VuGen中运行成功）
2. 打开控制台，设置Run-time Settings
3. Pacing值：随机2~3
4. Think time: 忽略
5. Duration: 1分钟

提示：配置好后，观察图表状态，有所变动，才修改成功。

**注意：**当Run-time Settings中迭代和VU部署设置(Duration)有冲突时，Duration的优先级较高。

**比如：**Duration选择第二项，就以此为准

Run for \_\_ days and \_\_ (HH:MM:SS)

如果选择第一项：Run until completion 还是听Duration，只是它放权了。

Duration是一把手，让二把手看着办，此时Run-time Settings说的算。

**测试报告中的结果，应该测试三次，取中间值。**

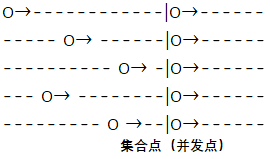
**比如：**0.1秒 0.3秒 0.4秒 结果取0.3秒

1. **并发测试**

**a、并发测试两个条件**

1. 脚本中要有集合点（并发点）
2. 控制台中要设置并发策略（选择第一项，所有虚拟用户到达集合点后释放）

集合点： 5个线程，代表5个VU 并发执行一次购票



等所有线程到达集合点时，才一起释放，此时的压力最大（瞬时压力）。

**注意：要在事务开始之前，设置并发点**

**b、并发点只有在并发测试中使用。**

案例：在脚本中添加并发点，执行并发测试

需求：并发购票

注意：在事务脚本之前添加

lr\_start\_transction("buy");

在事务开始之前 --> 点击Insert --> Rendezvous

--> 输入集合点名称Rendezvous Name: buy 一般与事务名相同

就会生成脚本：lr\_rendezvous("buy");

--> 编译 Compile（同时会立即保存）

**注意：脚本中发生变动（加了检查点、集合点、代码等）**

1. 一定要点击编译 Compile按钮，同时也会自动保存
2. 在控制台中要刷新脚本

**c、并发策略的设置：**

并发策略是在控制台配置：

控制台界面选择Scenario菜单 --> Randezvours...（并发点）

--> 打开窗口，设置策略 --> 点击Policy（策略）按钮

**第1项：**Release when 100% of all Vusers arrive at the rendezvous.（一般都选择此项）

当100%虚拟用户到了集合点时释放虚拟用户VU

（所有VU的n%） 10个VU 都算 10 \* n%

**第2项：**Release when 100% of all running Vusers arrive at the rendezvous.

当100%正在运行的VU到达集合点时释放VU

（所有正在运行的VU的n%）

如果10个VU只有5个正在运行，5 \* n%

**第3项：**Release when 1 Vusers arrive at the rendezvous.

指定n个虚拟用户达到集合点，再释放

**d、并发测试案例：完成5个VU的并发**

控制台 -> Basic schedule -> Quantity 改为 5

Start Vusers: 用户数少，登录时间快，不用改

Duration: 选Run until completion 表示瞬时压力

继续设置Run-time Settings:

Run Logic: 迭代次数 1

Pacing: 改为As soon as the previous iteration ends.

Log: 默认

Think time: 默认 忽略 Ignore think time（好比：不停地发请求，不给喘息时间）

**e、并发测试要点回顾：**

1. 事务前设置并发点（lr\_rendezvous("buy");）
2. 控制台中设置并发策略
3. 要忽略Think time
4. **综合场景测试**

**综合场景测试号称能够最真实的模拟实际生产环境**

**综合场景的几个要素：多用户、多个脚本（至少3个）、在线执行（多种操作）一段时间（1小时、50分钟等），一般是不加并发点。**

**注意：只要是多用户，就存在并发**

**综合场景测试过程中，所有用户循环执行相应的操作**

1. **录制好3个脚本**：购买机票buy、查询路线search、浏览航班scan

添加好事务点、检查点；（无需集合点）

转移事务中的Think time脚本至事务之外：lr\_think\_time(23);

将脚本载入到控制台中，并设置人数比例：

Group Name Quantity

buy 2

search 4

scan 4

1. **设置场景**

**Schedule by:**

Scenario: 按场景 场景中，多个VU统一配置、行动 （选择）

Group: 按组 每个组，组内VU统一行动 （按组行动）

**重点设置左下角Global Schedule:**

以上三个脚本都选中，一次配置三个（出现黑框） -->

**a、Start Vusers双击**-> 设置一个小的递增 单选第2项

--> 1 00:00:01 [HH:MM:SS] --> OK

**该设置表示：**每隔1秒钟加载一个VU--> 及时观察右边效果图：锯齿状

**b、Duration 双击**

--> 单选第2项：Run for 0 days and 00:30:00 (HH:MM:SS)--> OK

**该设置表示：**确定指定测试运行的时间为30分钟 （项目中一般50分钟、1小时）

如果第1项：Run until completion 直到结束，适合于循环，确定次数；

如果第3项：Run indefinitely 一直跑，直到手动停止

1. **Run-time Settings设置**

**说明：迭代次数默认1 具体次数由持续时间决定**

**a、pacing: 随机4-6秒 或 5-9秒**

正常：2-3秒，教学机较慢，设置偏大些，保证不出错

**b、日志log: 保留原有选项（出错时发送）**

原因：大量日志也会占用磁盘空间。

**c、Think time: 随机百分比，适当调大 200% - 300%**

**d、Continue on error: 错误时继续**

原因：长时间执行大量事务，个别出错继续运行，不影响全局。

**e、Vuser选择 线程 方式。节省系统资源**

**f、网络：模拟用户的网络，使用最大带宽**

**g、模拟缓存：选择不模拟**

**h、Option 选项：3个120改为600**

一般疲劳测试设置600足够了。

1. **配置Windows resources**

选择Run视图-->右击窗口-->Add Measurements...

-->Monitered Server Machines: 选机器 点击Add.按钮 -->

Machine Information:

Name: localhost 指定监控服务器的IP地址，主机名，目前就是本地主机

Platform: WINXP

--> OK

Resource Measurements on: localhost 清空里面所有选项

**自己完成选项的添加（固定13项）--> 点击Add按钮 --> 选择以下内容：**

**Memory中有4项：（内存）**

Available MBytes --> Add

%Committed Bytes in Use --> Add

Page Faults/sec --> Add

Pages/sec --> Add

**Network Interface中有2项： （网络）**

Bytes Total/sec --> MS TCP Loopback inter...回环 --> Add

本地主机才选回环

Packets/sec --> MS TCP Loopback inter...回环 --> Add

本地主机自己和自己通信，用回环

**PhysicalDisk中有4项（2个队列）：（磁盘）见到Total就选**

Avg.Disk Queue Length --> Total --> Add

Current Disk Queue Length --> Total --> Add

Disk Read Bytes/sec --> Total --> Add

Disk Write Bytes/sec --> Total --> Add

**Processor中有2项：（进程）**

%Processor Time --> Total --> Add Total表示总和

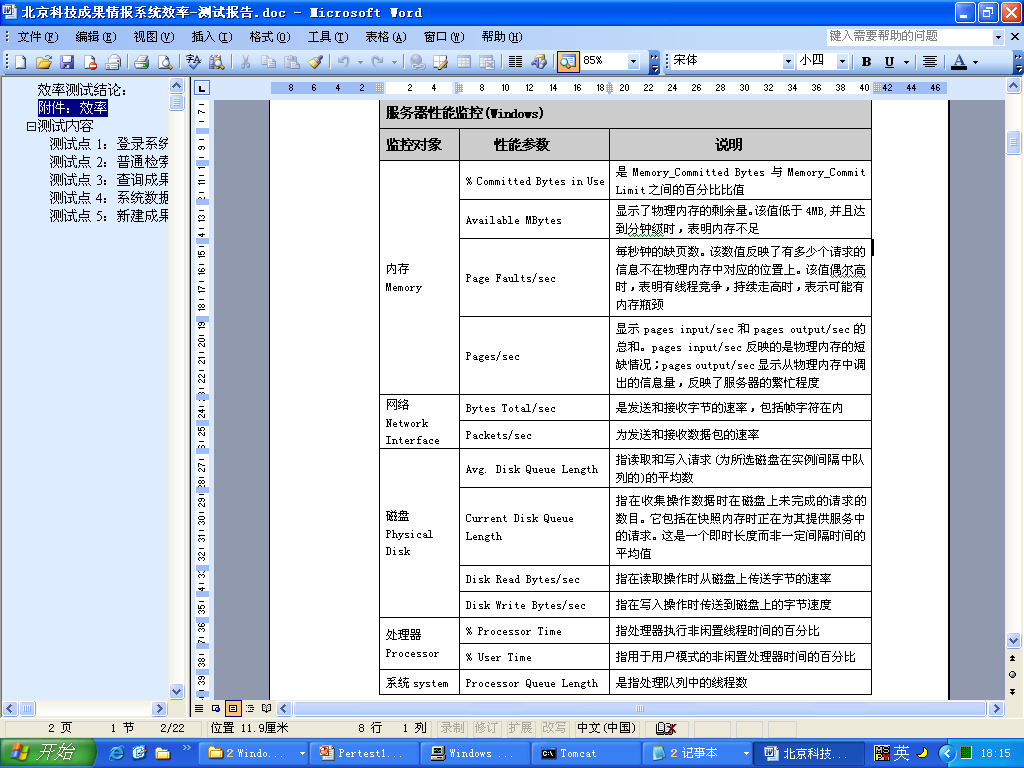
%User Time --> Total --> Add

**System中有1项： （系统）**

Processor Queue Length --> Add

--> OK

**以上一共13项windows监控资源，具体说明如下：**



**注意：运行过程中如果有错误，观察Scenario Status中的Errors 部分 0 （点开链接，寻找出错原因）**

**如果是场景设置问题，需要重新设置场景，重新运行**

**如果是脚本的问题，需要停止场景并调试脚本，重新运行**

1. **参数化**
2. **含义：同样的业务，出现不同的数据，考虑采用脚本参数化技术**
3. **步骤：**

确定参数化的数据-->准备数据-->提供策略-->参数化

1. **准备数据 （目前：File方式）**
2. 每个参数使用独立文件 （参数池）
3. 多个参数共享同一个文件 （常用）

Excel工具、Column（序号、列名）、First Data（从哪开始）

1. **参数池的策略（难点）**
2. Select next row 选择下一行 （How? 如何取?）

a、顺序 Sequential: 从第一行开始顺序向下选取

b、唯一 Unique: 从第一行开始，唯一向下选取

c、随机 Random: 随机取值

d、Same line as xxx: 取值策略与参数xxx相同 （同一行）

1. Update value on 更新方式 （When? 何时取?）

a、每次迭代 Each iteration: 每次迭代时取值。（Action一次）

b、每次遇到 Each occurrence: 每次遇到该参数时。

c、取值一次 Once: 脚本运行过程中只取值一次。

1. When out of values: 选择Unique才有，数据不足时处理情况：

a、放弃虚拟用户 Abort Vuser

b、以循环的方式继续 Continue in a cyclic manner

c、持续最后一个值 Continue with last value

经典需求：注册脚本，多用户，则参数池策略组合

Unique + Each iteration + Abort Vuser

另外，常用的组合：SE组合 （顺序 + 每次迭代）

1. **综合题（帮助理解参数池的策略）**

某参数现有备用数据a1, a2, a3, ... a30; 脚本迭代4次；3个VU;

完成一项结果： （必须掌握！）

1. 顺序 + 每次迭代 <重要>

VU1: a1 a2 a3 a4 顺序 每个VU取值序列相同

VU2: a1 a2 a3 a4 每个VU都从第一个开始往下取

VU3: a1 a2 a3 a4 每个VU共迭代4次，取4次值

1. 唯一 + 每次迭代（无特殊说明，块大小为自动分配） <重要>

VU1: a1 a2 a3 a4 唯一 从第一行依次唯一向下选取

VU2: a5 a6 a7 a8 每个VU都不同

VU3: a9 a10 a11 a12 每个VU共迭代4次，取4次值

1. 随机 + 每次迭代 <重要>

VU1: a18 a3 a5 a17

VU2: a20 a18 a13 a22

VU3: a30 a2 a28 a14

1. 顺序 + 每次遇到

VU1: a1 a2 a3 a4

VU2: a1 a2 a3 a4

VU3: a1 a2 a3 a4

顺序：每个VU都从第一个顺序向下取值 Action中参数只出现1次

每次遇到：每个VU遇到4次

（注意：使用参数模拟器，会出问题，不能被模拟，因为不确定数据量是否足够）

1. 唯一 + 每次遇到（块大小为6） <重要>

VU1: a1 a2 a3 a4

VU2: a7 a8 a9 a10

VU3: a13 a14 a15 a16

唯一：块大小6 第一块(a1 a2 a3 a4 a5 a6)

第二块(a7 a8 a9 a10 a11 a12)

第三块(a13 a14 a15 a16 a17 a18)

每次遇到：每个VU遇到4次，取每块中的前4个

1. 随机 + 每次遇到

VU1: a5 a8 a11 a20

VU2: a7 a5 a9 a15

VU3: a12 a17 a24 a26

每次遇到：由于参数只出现一次，效果等同于每次迭代

1. 顺序 + 一次 <重要>

VU1: a1 a1 a1 a1

VU2: a1 a1 a1 a1

VU3: a1 a1 a1 a1

顺序：每个VU取值一样；一次：只取第一个值

1. 唯一 + 一次 <重要>

VU1: a1 a1 a1 a1

VU2: a2 a2 a2 a2

VU3: a3 a3 a3 a3

唯一：每个VU从第一开始，唯一向下取值

一次：每个VU取值后，不再换值

1. 随机 + 一次 <重要>

VU1: a5 a5 a5 a5

VU2: a16 a16 a16 a16

VU3: a18 a18 a18 a18

随机：每个VU任意取值； 一次：取值一次，不再换值

1. 唯一 + 每次迭代 （块大小为6）注意与 2）的区别

VU1: a1 a2 a3 a4

VU2: a7 a8 a9 a10

VU3: a13 a14 a15 a16

唯一：块大小6 第一块(a1 a2 a3 a4 a5 a6)

第二块(a7 a8 a9 a10 a11 a12)

第三块(a13 a14 a15 a16 a17 a18)

1. **常用函数**
2. **web\_find函数**
3. 写在相应请求之后
4. Run-time Settings需要设置：可用
5. 左边界：RightOf
6. 右边界：LeftOf
7. web\_find执行效率低于web\_reg\_find.
8. **lr\_output\_message是LR的输出函数，经常用于脚本的调试。**
9. **脚本中快速定位**
10. 从日志定位到脚本行：双击日志、根据行号 Ctrl + g
11. 从脚本行定位到日志中：右击脚本 -> Go to Step in Replay log
12. **检查点函数区别**
13. **web\_find()**  （LR普通函数）

特点：从HTML页面中查找指定的文本字符串（比较占资源、效率低、较少使用）

1. **web\_image\_check()**

a、alt参数：当鼠标悬浮在图片上时显示的名字，给用户看的

b、src参数：图片的路径名及图片名称，给程序员看的

1. **web\_reg\_find()**  （LR注册性函数）

特点：在缓存中（Html源代码级别）查找相应的内容

效率高，最常用

web\_reg\_find要写在相应请求之前，因为是注册性函数

规律：LR所有注册性函数，含有reg字样，都要写在相应请求之前。

1. **脚本代码阅读说明（重要）：**

web\_reg\_find("Text=ABC", "SaveCount=abc\_count", LAST);

web\_url("Step", "URL=...", LAST);

if (strcmp(lr\_eval\_string("{abc\_count}"), "0") == 0) {

lr\_output\_message("not found");

} else{

lr\_output\_message("%s times", lr\_eval\_string("{abc\_count}"));

}

1. LR脚本中有两种变量，一种是C语言的变量，需要在脚本的开始处定义(如：int a; )；另一种是LR的变量，出现在LR的函数里，不需要定义，但是如果使用LR变量的实际值，则必须加{} (如：{name})
2. lr\_eval\_string函数 起到一个桥梁的作用，将LR的变量介绍给C语言的函数使用。 LR变量 --> C语言能够理解的字符串（高级脚本调试时常用）

该函数的返回结果，就可以给C语言函数当做参数使用。

lr\_eval\_string("{abc\_count}");

lr\_eval\_string("{name}");

1. strcmp(a, b) 表示比较两个字符串变量的值是否相等，如果返回0，则表示两个字符串相等。（C语言函数）
2. lr\_eval\_string("{abc\_count}")

lr\_eval\_string之后默认接(); ()中是字符串，所以加 " "

" " 中不是普通字符串，而是LR字符串变量，固定需要用{ }

1. C语言中%s表示字符串格式符，经常用于输出语句；%d 表示整数的格式符。
2. **关联**
3. **关联产生的原因**

**产生问题的原因：**Client端 WebServer端

**真实情况：**

Client 请求1 -------> WebServer

响应1 <------- 提供一个uid 123

请求2 携带uid 123 ---->

响应2 <---------

**LR模拟情况（第3项）：**

Client 请求3 携带uid 123 ---> WebServer

响应3 <------- 提供一个uid 456

请求4 携带uid 123 ---->

响应4 <--------- 错误原因：uid不匹配

某些系统的服务器在第一次和客户端打交道时，会附带一个id（该id随不同用户的变化而变化）；当客户端之后发请求时需要该id才能继续；而LR在录制时形成的id是一个静态的id（如123），这就导致客户端在发请求时无法获取服务器每次发送的动态的id，导致脚本失败。（脚本调试方向：将静态的id换成动态id）

关联的概念：Correlation

把脚本中某些写死的数据，转变为服务器所发送的、动态的、每次都不一样的数据。

何时做关联：正常录制，但是回放不成功，考虑是否关联。

1. **一般关联的操作的三大步骤：**
2. 从服务器返回的数据中选取需要进行关联的数据。（找数据）
3. 将该数据存入脚本的一个参数中。

变动的数据在脚本中是参数，由于是变化的，所以叫动态ID

1. 将脚本中需要使用数据的地方用该参数代替。

动态ID：需要关联的数据

比如：每一次执行脚本都会变动的值，就需要关联

执行一次就变动一次，或录制一次也会变动一次

1. **如何找到动态ID**
2. 原理：每执行（或者录制）一次，都会变动的值。
3. 录制两个一模一样业务流程的脚本，进行比对，找出动态ID。
4. 动态ID一般是一串无规律的字符串（少数情况下有规律）。
5. 动态ID一般不是坐标值、think time时间、检查点函数,也不是整段的请求。

关联类型：

1. 手动关联 （常用）
2. 自动关联 （经常导致脚本失败，LR回退，很少使用 了解）

手动关联的具体步骤：

1. 录制脚本，回放有错误。确定由于动态数据造成的。
2. 再录制一份相同流程的脚本。
3. 使用WDiff工具进行比较。（操作熟练，可以省略）
4. 使用web\_reg\_save\_param()函数，在源文件中找到需要关联的字符串（动态数据），存入一个参数中。
5. 把录制时数据替换为该参数（动态数据）

web\_ reg\_save\_param注册性函数，需要写在相应请求之前：寻找相应请求

1. **如何查找到相应请求？**（为了在相应请求之前写关联函数）
2. 拷贝动态ID适当长度（太长有可能在显示时换行、太短可能重复较多），从Generation Log的第一行开始查找！！！

如果查找到的位置在服务器的应答包中，说明查找正确。

（第一次，是从服务器发出的ID）

在脚本中拷贝ID的一小段：114276.5 ctrl + c

点击Generation Log --> 从第一行开始查找 ctrl + f

--> ctrl + v 查找该文本

找到位置在应答包中 Response标题 With Id 14

1. 查找到之后，选取适当长度的左右边界，拷贝下来，准备写函数。

name=userSession value=114276.568975294fiHiVAVpAVzzzzzHDfczipzAzV

1. 先向下慢慢翻找，找到与当前应答id号相同的id号的请求，该请求即为相应请求。（90%的情况都是在下方不远处）
2. 如果找不到，则回到原位置，向上慢慢找到最近的一条请求。
3. Request 和 Event 都是请求。

web\_url("... "Snapshot=t1.inf", ); 快照名是唯一的，作为线索

寻找脚本中ti.inf对应的请求位置。将拷贝的串信息粘贴到请求之前。

1. 找到相应请求之后，在请求之前写关联函数，并将后续脚本中用到动态ID的位置用函数中的参数替代。

web\_reg\_save\_param("uid", //参数名称

"LB=左边界的字符串", //左边界

"RB=右边界的字符串", //右边界

LAST);

相应请求;

之后，将脚本中的ID值，用参数uid代替：

"Name=userSession", "Value=114276.568975294fiHiVAVpAVzzzzzHDfczipzAzV", ENDITEM,

改为：

"Name=userSession", "Value={uid}", ENDITEM,

即可。{uid} 取得LR的变量值，LR语法范围内有效的。

编译脚本 --> 回放

1. **从业务逻辑去分析请求，相应请求所在的位置，当前脚本是基于HTML方式录制，结构较简单，所以当客户端 第一次对服务器发请求时，服务器就已经分配了动态ID，来识别不同的用户。所以第一次发请求就是相应请求。（基于URL的录制方式，则比较复杂，需要更加仔细分析）**

**所谓相应请求，就是 产生动态ID的应答 的相应请求。**

1. **性能测试分析**

**性能测试结束后，要对测试结果进行分析。分为三种情况：**

1. **测试结果完全符合需求，不需要分析。**
2. **测试结果存在问题（不符合需求，时间超长），直接通过测试报告（Analysis）查找到原因。直接写出报告。（前提：对应Analysis中结果描述非常清楚。）**
3. **如果通过测试报告没有得出结果（性能瓶颈），这种情况比较复杂。**
4. **比如发现某些事务响应时间超长（最普遍），该如何处理？**
5. 通过Analysis报告中几个比较重要的图表进行查看，初步定位问题，再通过网页细分图（网页诊断图）去确定响应时间长在系统的哪个部分。大多数情况下，时间长在服务器端。

（响应时间 = 客户端时间 + 网络时间 + 服务器时间）

1. 要进一步确定是哪台服务器（复杂的系统服务器n，甚至几十台--集群）

答案：通过 监控服务器的系统资源图，能够很容易地定位出哪台服务器不正常。

1. 如果是应用服务器（在企业中也常称为中间件），如Tomcat、JBoss、Weblogic、WebSphere，发生问题，调整服务器配置参数即可。

共性：都是软件，都安装在服务器主机上提供企业级应用的支持。

区别：免费、收费（上百万）

Tomcat: Apache开源组织、免费

Weblogic: BEA公司出品，后来被Oracle收购了

WebSphere: IBM公司应用服务器

1. 大部分情况都是数据库服务器出现问题，可以通过专门的 数据库监控工具 去监控，甚至可以提取到相应有问题的sql语句。可以对sql语句进行分析、调优，进而提升被测系统的性能。
2. **以上的调试流程不适合于每个被测系统，绝大部分的系统可以在前面的某些步骤中即可停止，完成性能分析过程。**
3. **页面细分图：**
4. **操作：**右击Graphs --> Add New Item --> Add New Graphs...

--> 打开 Open a New Graph窗口，在Web Page Diagnostics中进一步查看细分图。

**a、Web Page Breakdown**

页面中的组件，也叫做页面中的元素。包括组成网页的内容：文字、图片、音频、视频、动画...

**b、Page Component Breakdown (Over Time)**

页面组件细分图（随时间变化） 更细致

**c、Page Download Time Breakdown**

页面下载时间细分图

响应时间：包括请求后，响应的各个阶段

**八项中主要关注前4项：**

DNS（Domain Name System） 域名解析时间（好比：根据公司名找到主机号码时间）

Connection 连接时间（好比：公司找一个客服解析接待的时间）

First Buffer 第一次缓冲时间（好比：获取第一个数据包的时间，很重要）

Receive 接收时间（好比：获取到所有数据包的接收时间，从第一个字节开始记录）

SSL握手 只发生在基于Https协议通信的网页上

Client Time 由于客户端引起的延迟

Error Time 系统页面报错时才发生，不是总有

FTP Time FTP验证时间：当系统中存在FTP(File Transfer Protocol) 文件传输协议，服务器下载操作时才产生该时间

**特别说明：**

First Buffer Time第一次缓冲的时间 （第一个数据包）

Client ----> Web Server ----> DB Server

<---- <-----

可以细分为：网络及服务器处理时间 + 数据库时间

客户端（Client）从发送请求 到 收到第一个缓冲（大小 8K）之间的时间

**问题：First Buffer时间 和 Receive时间 有没有交集?**

发送 4K内容，如果First Buffer是8K，则有交集。无需过于区分，因为侧重点不同。

**d、First Buffer的时间：细分为网络时间 和 服务器时间，如果网络（带宽）情况不好，则网络的时间可能会对服务器时间造成影响；如果是内网测试，则基本没有这种问题。**

**操作：**打开**Time to First Buffer Breakdown** 第一次缓冲时间细分图

看到只对First Buffer进行细分，分为：

Network Time 网络时间

Server Time 服务器时间 （占绝大部分）

打开**Time to First Buffer Breakdown(Over Time)**第一次缓冲时间细分图（随时间变化）

发现细分图条数：元素个数 \* 2

**f、页面中所有组件（元素）大小的和 = 该页面的大小**

**操作：**打开**Download Compoent Size(KB)** 已经下载的组件大小图

结合Breakdown Tree 细分树 查看

1. **看结果分析图时，一定注意：不光看图的走势，还要看图中坐标的单位。要二者结合出结果。**

**比如：一张图中走势在某处（某个时间范围）很陡峭，但是其下载时间的坐标轴中最高值为0.006，所以该时间没有问题。**

**现象：First Buffer占的比例较大（与网络 和 服务器有关），可以进一步细分：网络时间 和 服务器时间**

**网络时间：**

Client 第一次Http请求开始计时 --> Web Server --> DB Server

<--直到返回服务器的第一个字节为止（收到确认）

**服务器时间：**

--收到请求确认，并处理请求开始计时 -->

<--直到返回第一次缓冲的时间

1. **页面诊断图（综合图）**

**操作：**

1. 右击Graph --> Add New Item --> Add New Graph--> Web Page Diagnostics (第一项)
2. 从事务响应时间图中选中某条需要分析的时间，点击右键（页面诊断），直接打开（经常使用，目标明确）

找到Graph --> Average Transaction Response Time

--> 找到某条折现 --> 右击 --> show Diagnostics for "xxx" 打开，xxx就是事务的名称，比如"login"。

**观察综合图的分布规律：**

1. 左边是细分树 Breakdown Tree

比如login的URL地址，对应右边中间：

Select Page to Break Down 中的地址。

1. 最上边 Average Download Time 下载细分时间
2. 右边中间的组件细分部分 （最关心的部分）

具备4个单选钮：

Download Time

Component(Over Time) 组件细分

Download Time(Over Time) 下载时间细分

Time to First Buffer(Over Time) 第一次缓冲时间

关注：Download Time 分为8种颜色对应相应时间

右击条目：

a) 拷贝全路径名到剪切板

b) 在浏览器显示 View page in browser

各种颜色中，蓝色的First Buffer比例最大，想进一步分析First Buffer，单选：Time to First Buffer(Over Time)

又细分为：Network Time 和 Server Time (大部分)

基本线索：找到某组件，单选钮切换是针对该组件看不同相关的图

**归纳：**

**右侧：上中下都有关系**

**上：显示下载时间图**

**中：显示URL地址、具体细分图**

**下：显示该项打钩**

1. **页面诊断图（综合图）的分析方式：**

一般从事务平均响应时间中定位到某个事务产生诊断图。

1. 一般情况下，先通过默认页面（Download Time）--> First Buffer，即发现当前页面中某个组件的第一次缓冲时间较长，再到第一次缓冲时间细分图中，确认网络时间还是服务器时间，多半是服务器时间较长，则瓶颈定位到服务器端。
2. 选中页面的某一元素，则其后的Sheet页逐一切换，能够自动定位到统一元素。（加亮显示）

分析：如果一个图片0.2M，假如1000用户并发执行某事务，该页面中包含这个图片，则0.2M \* 1000 = 200M，如果带宽10MB，则光下载该元素（图片）需要200/10 = 20秒。使用造成性能影响，带来较大压力。（一般页面是3-5-8原则）

1. **windows资源监控**
2. %Processor time: CPU忙的时间的百分比，当该值的平均值超过80%或者85%时（(yu)阈值），一般怀疑CPU有瓶颈。

---如果CPU的确有瓶颈（处理能力不足），可以添加CPU或者置换性能较高的CPU.

（硬件问题较易解决，软件问题较难解决）

1. Processor Queue Length

a、阈值：单CPU不应大于2；如果是n块CPU，则该值不应大于n+1;

b、双核，则n=2 （虽然双核性能不及2块CPU，但也当做2块CPU处理）

1. 注意：在服务器资源中，如果没有特殊说明，则都是值 平均值。（最大值和最小值只作为一般参考）
2. 如果一个性能测试点（比如login）,其中依次完成单用户登录、50用并发登录、100用户并发登录；则这个测试中%Processor Time的平均值应该呈现递增的趋势。
3. 内存重要指标 Memroy

a、对于Windows系统而言，可用物理内存不要少于1%。

比如：2G内存，如果可用物理内存20M或者更少，则要引起注意，怀疑内存处理能力不足。

b、观察可用物理内存曲线时，在测试过程中提交的物理内存会在测试结束后慢慢释放回来，则这种系统为正常的内存表现。

c、系统的内存存在大量软错误（在内存的其它位置找到该资源）的情况下能正常工作，但是如果硬错误（必须重新回磁盘找数据）较多，则会对系统的性能带来影响。

d、Page Reads/sec (页面读取率)：如果该值持续大于内存的1%（如果2G内存，该值持续大于20M），则怀疑内存不足。

说明：在Windows资源监控中，Memory部分：Page Faults/sec和Pages/sec也要监控。

1. 磁盘的重要指标 Physical Disk

a、Disk Reads(Writes)/sec 磁盘读或写的阈值：

一般不超过几十M

如果发现磁盘的读或写超过过了几十M或者上百M，会严重影响系统性能。怀疑是磁盘的瓶

b、性能调优的一个重要原则就是：尽量减少磁盘IO

说明：Disk I/O 就是磁盘的in或out 读/写

c、磁盘的I/O绝对不可避免（磁盘和内存的交互必不可少），但是可以尽量减少。

1. 网络重要指标：Bytes Total/sec (重要) 和 Packets/sec
2. Bytes Total/sec: 将该值乘以8，与网络带宽的一半进行比较，如果小于带宽的一般，则一般没有问题。
3. 1Byte=8bits

说明：网络商带宽单位一般都用bit，显得数值大。比如100Mbps

1. 性能分析--监控服务器资源
2. 客户端在测试时，不需要监控，因为主要对AUT监控，主要是其服务器，为何只监控服务器：测试AUT的性能，只需关注AUT的服务器即可，客户端只需偶尔观察：使用安装LR的测试机（模拟客户端）测试时，偶尔打开本机的任务管理器，点击“性能窗口”，查看，保证正常使用即可。
3. 如何监控服务器的内存泄露？

首先正常测试（监控正常指标），当发现被测系统的内存曲线不正常时，再去测试三项指标

正常的情况：物理内存在测试过程中不断被吃进，在测试结束后释放回来。

三项指标：（两升一降）

process\private bytes 和 process\working set 计数器 升高

available bytes 的值降低

1. 如何判断应用程序的问题：CPU平均值高，吞吐率整体水平不高甚至有所下降，但是上下文切换很高（正常系统中偶尔的、适当的上下文切换是有必要的，若频繁，则有问题），说明系统应用程序存在设计或代码方面的问题。