目录

[性能测试工具Loadrunner 2](#_Toc324754258)

[创建脚本 2](#_Toc324754259)

[事务（transaction） 2](#_Toc324754260)

[集合点（rendezvous） 3](#_Toc324754261)

[参数化 4](#_Toc324754262)

[关联： 4](#_Toc324754263)

[检查点 8](#_Toc324754264)

[Loadrunner结果分析 10](#_Toc324754265)

[图表含义： 10](#_Toc324754266)

[性能测试流程： 15](#_Toc324754267)

[第一：建立压力模型。 15](#_Toc324754268)

[第二：准备基础数据。 15](#_Toc324754269)

[第三：开发性能测试脚本。 15](#_Toc324754270)

[第四：设置运行场景。 15](#_Toc324754271)

[第五：性能测试结果分析。 16](#_Toc324754272)

[常见问题： 16](#_Toc324754273)

[综述： 19](#_Toc324754274)

# 性能测试工具Loadrunner

Loadrunner是HP公司研发的性能测试工具，原理是通过刻录传输协议生成脚本，增强脚本以后模拟大量用户并发进行性能测试。

对于负载生成器配置要：按HP过去官方说法1.5-2MB内存可以生成一个虚拟用户，由于系统复杂程度增加，关联字符串越来越长，现在已经不局限于2MB可以生成一个虚拟用户，很可能会远远超过这个数字。内存的特点是使用率超过75%会转移数据到硬盘存储，所以生成虚拟用户加上机器操作系统本身用户不宜超过总内存的75%；对于CPU要求CPU使用率不超过80%，一般情况对CPU要求不高，不过当TPS达到每秒几百级别的时候普通PC机的CPU可能会产生瓶颈。

## 创建脚本

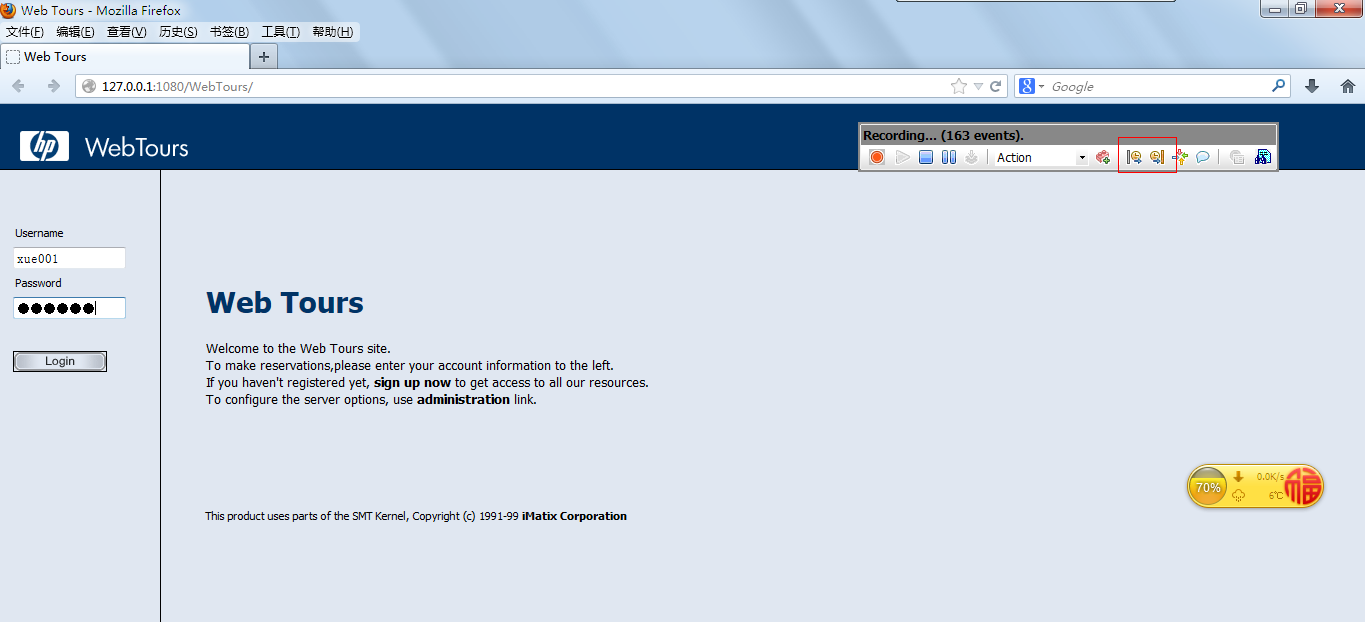
选择loadrunner刻录协议和启动浏览器（客户端程序）方法不再详述，在刻录过程中，要等页面打开完全（有时候是LR接收到的事务不再增加，如果系统某些功能不停刷新那么接收到的事务就会一直增加）。我们在loadrunner的Script中看到的代码均为发送到服务器的数据，在增强脚本过程中，重点是事务（可以在刻录的同事完成添加）、集合点（可以在刻录的同事完成添加）、参数化、关联（在参数化后调试后增加）和设置检查点（每一个单独刷新部位最好都有检查点）。

### 事务（transaction）

**作用：**记录lr\_start\_transaction 和 lr\_end\_transaction之间所用的时间。

**使用方法：**

方法一：在刻录过程中，通过点击按钮插入要计时操作的开始位置和结束位置。



方法二：生成脚本后在需要插入事务的步骤开始位置和结束位置Insert → start transaction 和Insert → end transaction

**注意：**不同脚本的事务如果名称相同也会被认为是同一个事务。事务中间不能有思考时间。

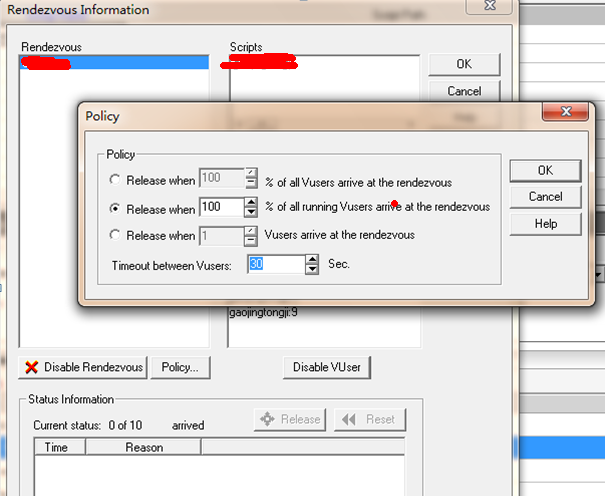
### 集合点（rendezvous）

**作用：**虚拟用户在集合点处等待达到用户并发数量后释放。

**设置方法：**与事务设置方法基本相同。

**注意：**不同脚本的集合点如果名称相同也会被认为是同一个集合点。在场景设计中，用户可以通过集合点策略控制并发用户数量。集合点不能放在事务内，建议集合点和事务之间不要有思考时间。

集合点策略设置方法：场景设置→Scenario →Rendezvous.. →Policy..



### 参数化

**作用：**参数化是模拟不同用户向服务器提交不同数据，让脚本更接近真实用户使用情况。

**参数化类型：**各种参数化类型可以在设置参数类型时直接选择，不同参数有不用特地，可以参照帮助。

注意：参数化后注意参数的分配模式。

组合类型：（来自网络，不过通用理论）

Sequential

* 与Each iteration组合：将为每次迭代从数据表中提取下一个值。
* 与Each occurrence组合：将为每一次参数的出现从数据表格中提取下一个值，即使它在同一次迭代中。
* 与Once组合：第一次迭代中分配的值就会在每个Vuse接下来所有的迭代中使用。

Random

* 与Each iteration组合：将会为每一次迭代从数据表中提取一个新的随机值。
* 与Each occurrence组合：将会为每一次参数的出现从数据表中提取一个新的随机值，即使它在同一次迭代中。
* 与Once组合：第一次迭代中分配的随机值就会在改Vuser的所有迭代中使用。

Unique

* 与Each iteration组合：将会为每一次迭代从数据表格中提取下一个唯一值。
* 与Each occurrence组合：将会为每一次参数的出现从数据表格中提取一个新的一直，即使它在同一次迭代中。
* 与Once组合：第一次迭代中分配的唯一值就会在每个Vuser的所有接下来的迭代中使用。

**如果前面已经有了参数化的值，我们可以和前面参数分配行数保持一致。**

**在参数化时，对于一次压力 测试 中均只能用一次的资源应该怎么参数化呢？就是说这些资源用了一次就不能在用了的。**

**－－参数化时,在select next row选择unique,update value on选择 each occurence**

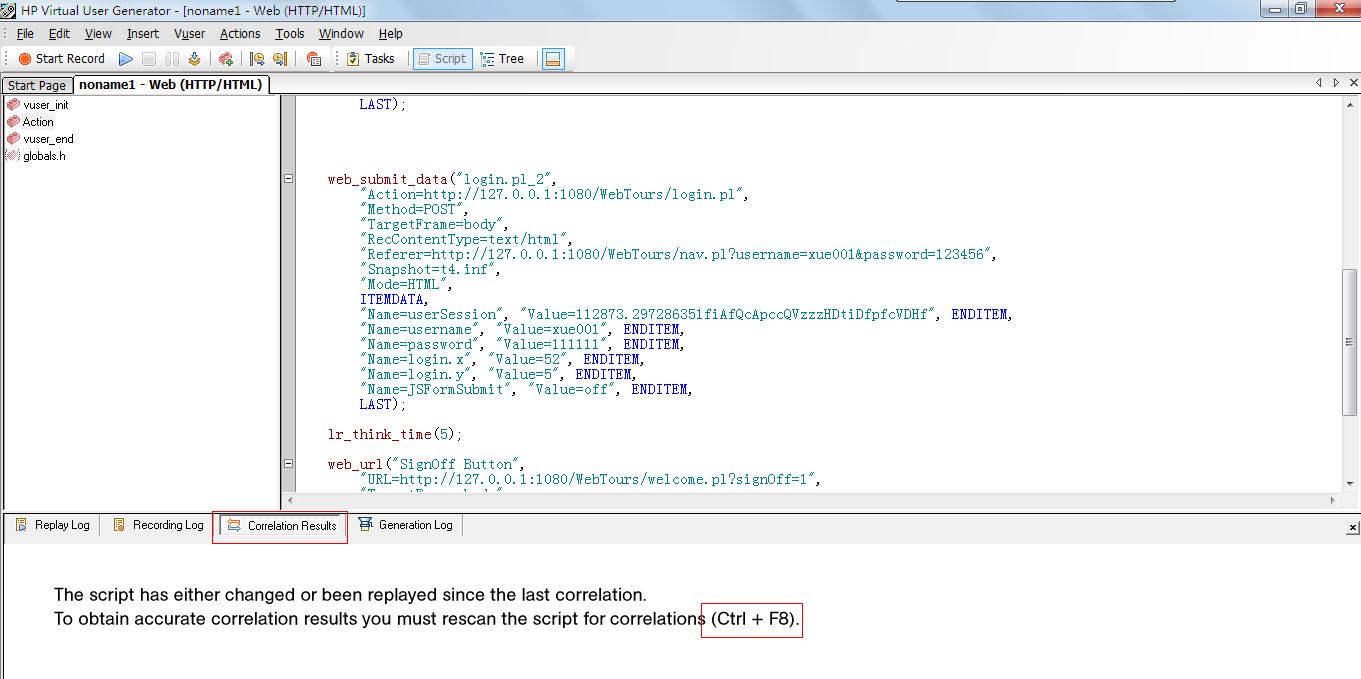
### 关联：

**作用：**当脚本像服务器发送请求的数据不同，服务器返回数据也不相同，而后面要提交的数据恰好是服务器返回变化的数据时，我们通过关联获取服务器返回的数据并将数据赋与提交函数。（可以通过网络详细了解关联的描述）

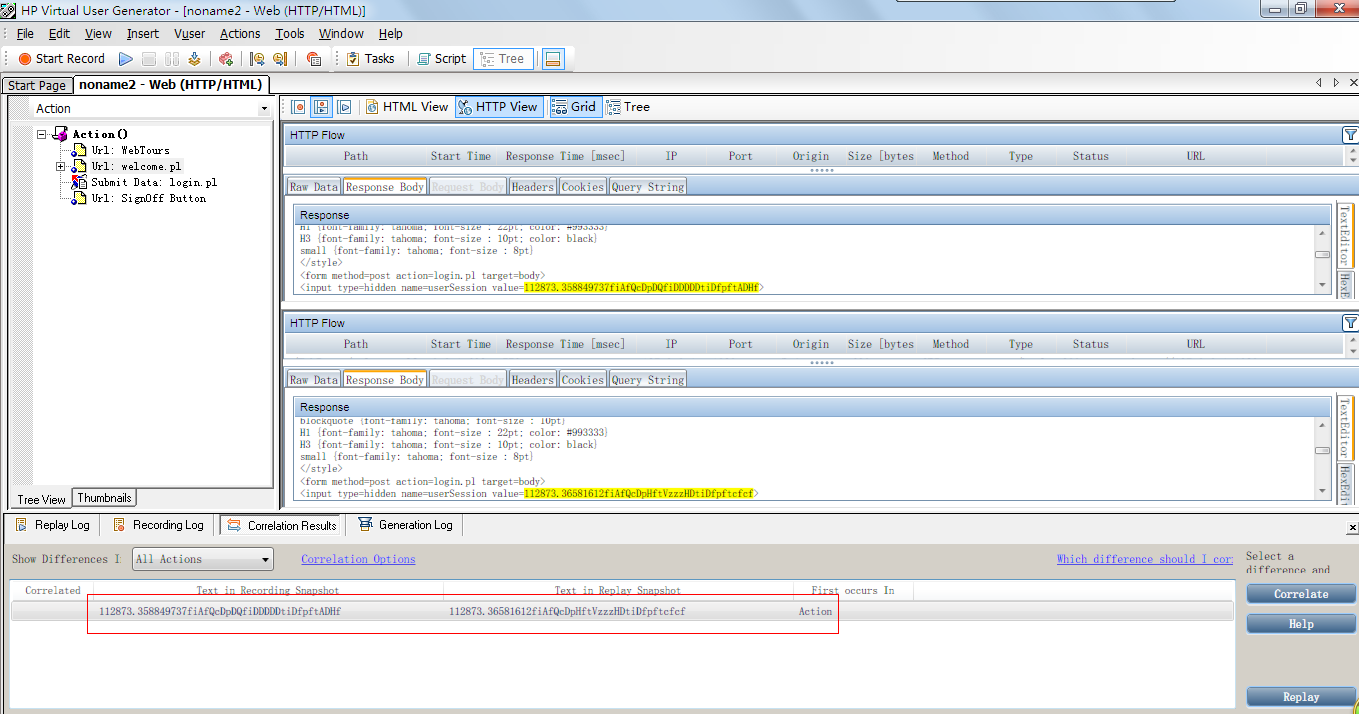
**关联方法：**

自动扫描：

* 1. 选择Correlation Results→从键盘输入Ctrl + F8（如果经过参数化，还没有回放过脚本会提示回放脚本，确认回放即可回放即可，扫描关联之前需要回放脚本。）

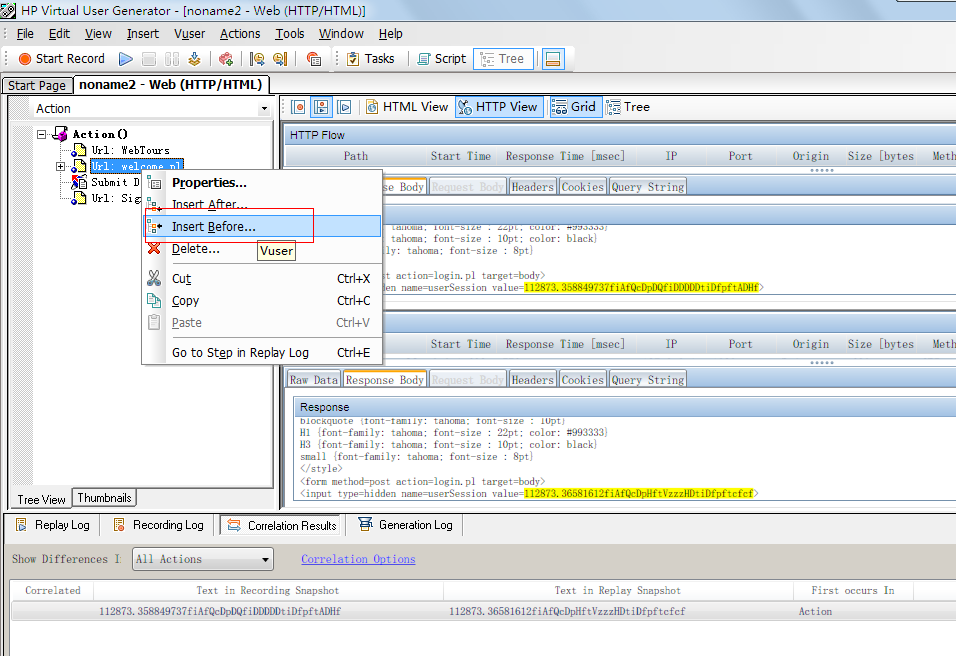


* 1. 选中应该关联的值点击Correlate。

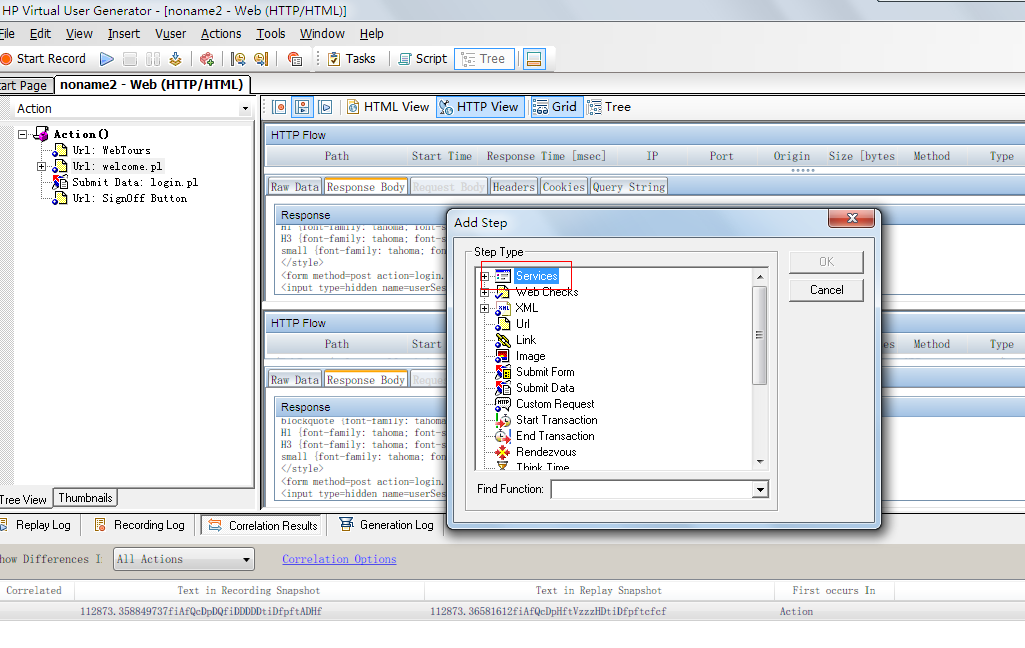


手动关联：

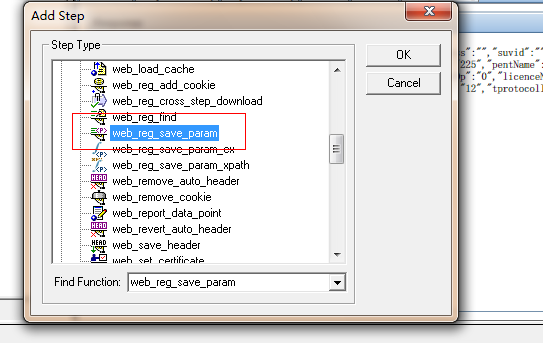
* 1. 用不同的参数按相同的操作步骤刻录脚本，对比两个脚本不同的提交数据。（该方法在操作流程短的脚本中比较奏效，如果是很长的业务流程，只能凭借经验观察）
  2. 通过脚本Tree模式提交函数中的Grid显示下的Response Body找到出现不同数据。
  3. 在该提交函数之前写入添加关联函数。
     1. 在该提交函数右击选择insert before



* + 1. 选择Services



3. web\_reg\_save\_param →点击OK



* 1. 输入关联名称、左边界和右边界。（通过左右边界确定唯一值）

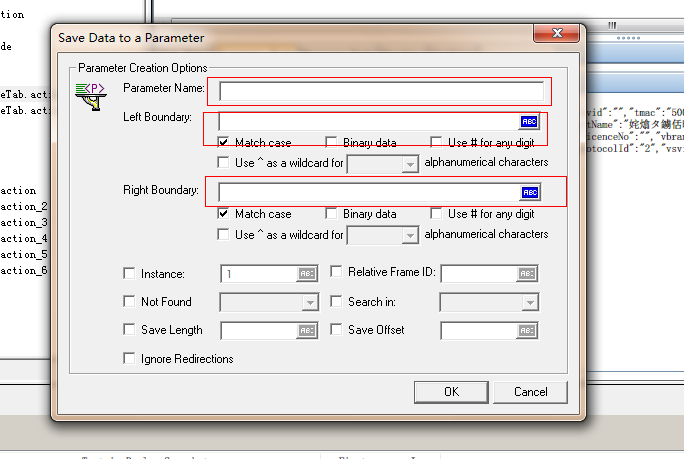
注意：当边界值中遇到“””时，需要用“\””进行转义，例如：

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

"LB=<input type=hidden name=userSession value=",

"RB=>",

LAST);



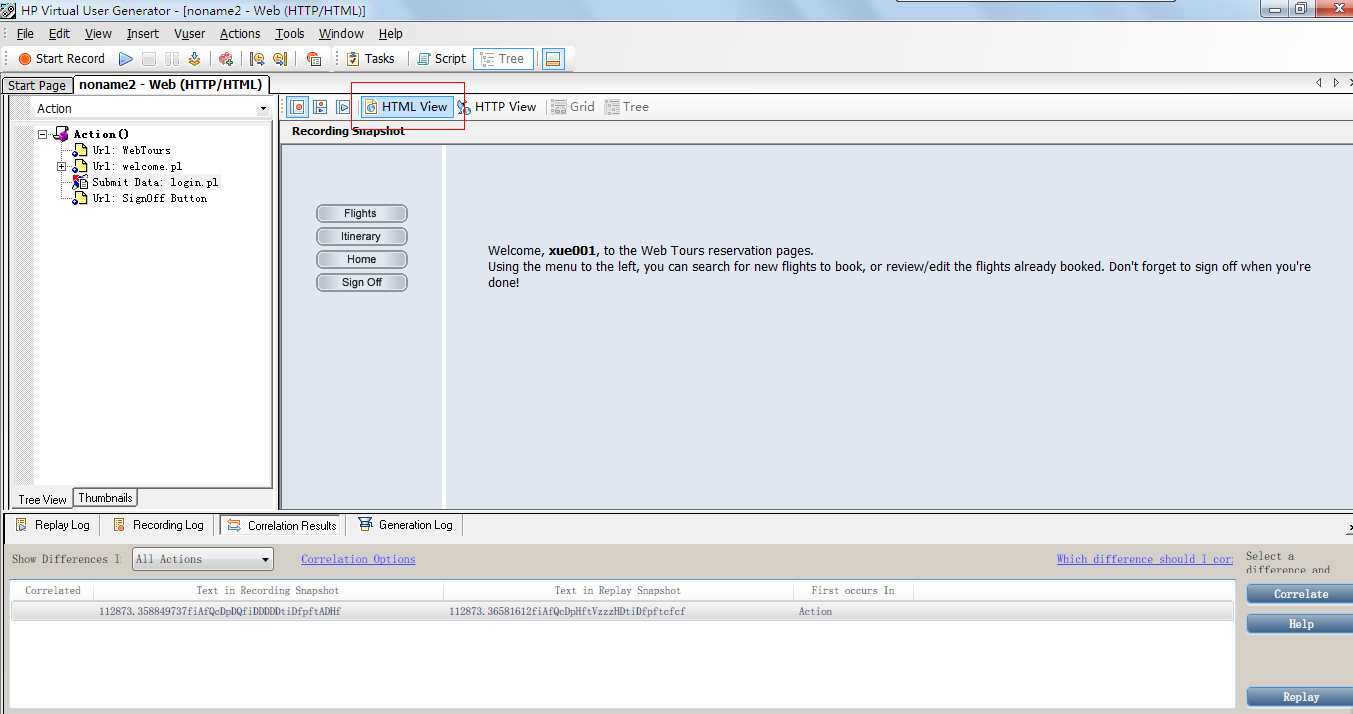
5.用“{关联名称}”替换掉提交的数据。

### 检查点

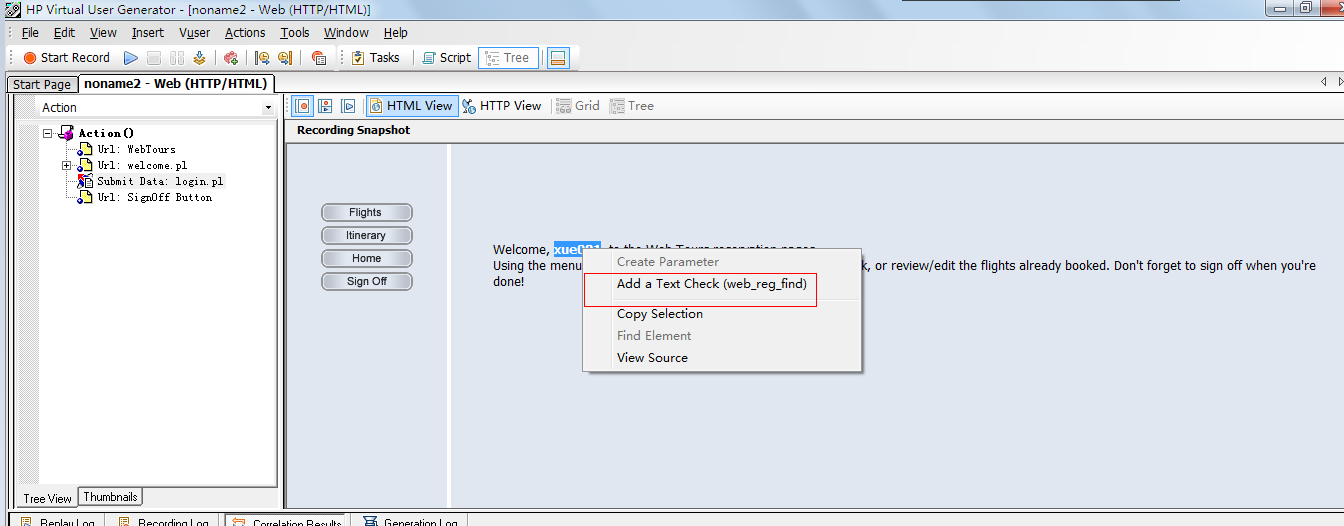
**作用：**由于脚本运行所接受到的都是协议，不会展示给测试人员，所以通过检查点来确定返回的页面是不是正确。

文本检查点添加步骤：

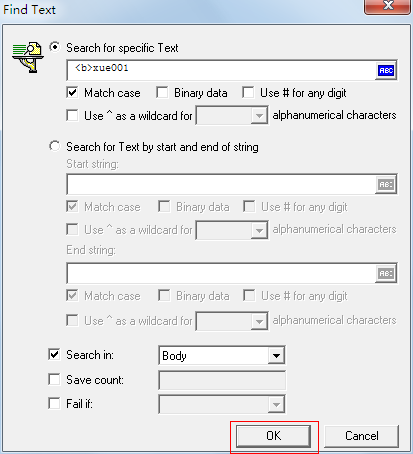
1. 通过脚本Tree模式中的HTML View找到要添加检查点的文字。



1. 选择要检查的文字，右击选择 Add a Text Check (web\_reg\_find)



1. 点击OK



注意：手工写web\_reg\_find请查询帮助，这个函数实际上是注册函数，必须在获取到检查文本以前就注册，方位类似与关联函数；文本检查函数还有一个web \_find请查询帮助，使用方法类似与位图检查点，所放的位置必须在获取到检查文本以后。

位图检查点添加步骤：

1. 选择要检查的图片，查询它的完整路径。
2. 在页面源代码中确定是Src还是Alt。
3. 在得到图片的函数后面添加检查函数

web\_image\_check("web\_image\_check",

"Src=http://www.51testing.com/imagesnew/435-92-2011.jpg",

LAST);

**注意：设置检查点后无论文本检查点还是图片检查点，勾选 Vuser → Run-time Settings → Preferences → Enable Image and text check.检查点才能发挥作用。**

### 迭代

1. 如果场景scheduler设置run until completed, 那么所有虚拟用户都会完成设定的迭代次数

2. 假如一个脚本，设置最大并发量为10，每5秒中增加2个并发用户，而Action设置的迭代为10次：当开始至2秒时，加载了2个用户，这2个用户分别开始运行，并都运行10次，不管这个2个用户运行10次是否结束，当下一个2两秒到来时，即开始至第4秒时又加载了2个用户，这2个又运行10次；就这样一直加载到10个并发用户，然后当每个用户都运行完10次时就结束。这样中间最大并发是10个，但不一定能达到10个，因为在加载最后几个时，前面的有可能已经运行结束，所以如果要真正达到最大并发10就必须设置集合点来完成，不过也不一定非要设置集合点才能实现同时处在running的状态有10个用户

## Loadrunner结果分析

### 图表含义：（来源网络，浅色字是我添加）

Transactions（用户事务分析）  
用户事务分析是站在用户角度进行的基础性能分析。

1、Transation Sunmmary（事务综述）  
对事务进行综合分析是性能分析的第一步，通过分析测试时间内用户事务的成功与失败情况，可以直接判断出系统是否运行正常。通过响应时间的最大值、最小值、平均值和90%响应时间值判断系统是否稳定运行，如果响应时间过大需要观察其他图表分析瓶颈。

2、Average Transaciton Response Time（事务平均响应时间）  
“事务平均响应时间”显示的是测试场景运行期间的每一秒内事务执行所用的平均时间，通过它可以分析测试场景运行期间应用系统的性能走向。图表的绘制方法是采点绘图，图片中的数据与Transation Sunmmary中未必完全相符，这不奇怪。  
例：如果随着测试时间的变化，系统处理事务的速度开始逐渐变慢，这说明应用系统随着投产时间的变化，整体性能将会有下降的趋势，造成这种情况的一般是由于系统负载增加造成系统处理能力下降。如果随着测试时间的变化，系统处理事务的速度出现剧烈变化，则需要合并用户运行状态图确定用户量对响应时间的影响，并且对比系统资源使用情况确定对系统产生的压力，这时不能通过单一图表做出简单判断。

3、Transactions per Second（每秒通过事务数/TPS）  
“每秒通过事务数/TPS”显示在场景运行的每一秒钟，每个事务通过、失败以及停止的数量，使考查系统性能的一个重要参数。通过它可以确定系统在任何给定时刻的时间事务负载。分析TPS主要是看曲线的性能走向。  
将它与平均事务响应时间进行对比，可以分析事务数目对执行时间的影响。  
例：当压力加大时，点击率/TPS曲线如果变化缓慢或者有平坦的趋势，服务器某项资源居高不下，很有可能是服务器或者程序开始出现瓶颈。

4、Total Transactions per Second（每秒通过事务总数）  
“每秒通过事务总数”显示在场景运行时，在每一秒内通过的事务总数、失败的事务总署以及停止的事务总数。

5、Transaction Performance Sunmmary（事务性能摘要）  
“事务性能摘要”显示方案中所有事务的最小、最大和平均执行时间，可以直接判断响应时间是否符合用户的要求。  
重点关注事务的平均和最大执行时间，如果其范围不在用户可以接受的时间范围内，需要进行原因分析。

6、Transaction Response Time Under Load（事务响应时间与负载）  
“事务响应时间与负载”是“正在运行的虚拟用户”图和“平均响应事务时间”图的组合，通过它可以看出在任一时间点事务响应时间与用户数目的关系，从而掌握系统在用户并发方面的性能数据，为扩展用户系统提供参考。此图可以查看虚拟用户负载对执行时间的总体影响，对分析具有渐变负载的测试场景比较有用。

7、Transaction Response Time(Percentile)（事务响应时间(百分比)）  
“事务响应时间(百分比)”是根据测试结果进行分析而得到的综合分析图，也就是工具通过一些统计分析方法间接得到的图表。通过它可以分析在给定事务响应时间范围内能执行的事务百分比。

8、Transaction Response Time(Distribution)（事务响应时间(分布)）  
“事务响应时间(分布)”显示在场景运行过程中，事务执行所用时间的分布，通过它可以了解测试过程中不同响应时间的事务数量。如果系统预先定义了相关事务可以接受的最小和最大事务响应时间，则可以使用此图确定服务器性能是否在可以接受的范围内。

Web Resources（Web资源分析）  
Web资源分析是从服务器入手对Web服务器的性能分析。

1. Hits per Second（每秒点击次数）  
   “每秒点击次数”，即使运行场景过程中虚拟用户每秒向Web服务器提交的HTTP请求数。  
   通过它可以评估虚拟用户产生的负载量，如将其和“平均事务响应时间”图比较，可以查看点击次数对事务性能产生的影响。

通过对查看“每秒点击次数”，可以判断系统是否稳定。系统点击率下降通常表明服务器的响应速度在变慢，需进一步分析，发现系统瓶颈所在。

2、Throughput（吞吐量）  
“吞吐率”显示的是场景运行过程中服务器的每秒的吞吐量。其度量单位是字节，表示虚拟用在任何给定的每一秒从服务器获得的数据量。  
可以依据服务器的吞吐量来评估虚拟用户产生的负载量，以及综合服务器硬件资源使用图看出服务器在流量方面的处理能力以及是否存在瓶颈。  
“吞吐率”图和“点击率”图的区别：  
“吞吐率”图，是每秒服务器处理的HTTP申请数。  
“点击率”图，是客户端每秒从服务器获得的总数据量。

3、HTTP Status Code Summary（HTTP状态代码概要）  
“HTTP状态代码概要”显示场景或会话步骤过程中从Web服务器返回的HTTP状态代码数，该图按照代码分组。HTTP状态代码表示HTTP请求的状态。

4、HTTP Responses per Second（每秒HTTP响应数）  
“每秒HTTP响应数”是显示运行场景过程中每秒从Web服务器返回的不同HTTP状态代码的数量，还能返回其他各类状态码的信息，通过分析状态码，可以判断服务器在压力下的运行情况，也可以通过对图中显示的结果进行分组，进而定位生成错误的代码脚本。

5、Pages Downloader per Second（每秒下载页面数）  
“每秒下载页面数”显示场景或会话步骤运行的每一秒内从服务器下载的网页数。使用此图可依据下载的页数来计算Vuser生成的负载量。  
和吞吐量图一样，每秒下载页面数图标是Vuser在给定的任一秒内从服务器接收到的数据量。但是吞吐量考虑的各个资源极其大小（例，每个GIF文件的大小、每个网页的大小）。而每秒下载页面数只考虑页面数。  
注：要查看每秒下载页数图，必须在R-T-S那里设置“每秒页面数(仅HTML模式)”。

6、Retries per Second（每秒重试次数）  
“每秒重试次数”显示场景或会话步骤运行的每一秒内服务器尝试的连接次数。  
在下列情况将重试服务器连接：  
A、初始连接未经授权  
B、要求代理服务器身份验证  
C、服务器关闭了初始连接  
D、初始连接无法连接到服务器  
E、服务器最初无法解析负载生成器的IP地址

7、Retries Summary（重试次数概要）  
“重试次数概要”显示场景或会话步骤运行过程中服务器尝试的连接次数，它按照重试原因分组。将此图与每秒重试次数图一起使用可以确定场景或会话步骤运行过程中服务器在哪个时间点进行了重试。

8、Connections（连接数）  
“连接数”显示场景或会话步骤运行过程中每个时间点打开的TCP/IP连接数。  
借助此图，可以知道何时需要添加其他连接。  
例：当连接数到达稳定状态而事务响应时间迅速增大时，添加连接可以使性能得到极大提高（事务响应时间将降低）。

9、Connections Per Second（每秒连接数）  
“每秒连接数”显示方案在运行过程中每秒建立的TCP/IP连接数。  
理想情况下，很多HTTP请求都应该使用同一连接，而不是每个请求都新打开一个连接。通过每秒连接数图可以看出服务器的处理情况，就表明服务器的性能在逐渐下降。

10、SSLs Per Second（每秒SSL连接数）  
“每秒SSL连接数”显示场景或会话步骤运行的每一秒内打开的新的以及重新使用的SSL连接数。当对安全服务器打开TCP/IP连接后，浏览器将打开SSL连接。

Web Page Breakdown（网页元素细分）  
“网页元素细分”主要用来评估页面内容是否影响事务的响应时间，通过它可以深入地分析网站上那些下载很慢的图形或中断的连接等有问题的元素。

1、Web Page Breakdown（页面分解总图）  
“页面分解”显示某一具体事务在测试过程的响应情况，进而分析相关的事务运行是否正常。  
“页面分解”图可以按下面四种方式进行进一步细分：  
1)、Download Time Breaddown（下载时间细分）  
“下载时间细分”图显示网页中不同元素的下载时间，同时还可按照下载过程把时间进行分解，用不同的颜色来显示DNS解析时间、建立连接时间、第一次缓冲时间等各自所占比例。  
2)、Component Breakdown(Over Time)（组件细分(随时间变化)）  
“组件细分”图显示选定网页的页面组件随时间变化的细分图。通过该图可以很容易的看出哪些元素在测试过程中下载时间不稳定。该图特别适用于需要在客户端下载控件较多的页面，通过分析控件的响应时间，很容易就能发现那些控件不稳定或者比较耗时。  
3)、Download Time Breakdown(Over Time)（下载时间细分(随时间变化)）  
“下载时间细分(随时间变化)” 图显示选定网页的页面元素下载时间细分（随时间变化）情况，它非常清晰地显示了页面各个元素在压力测试过程中的下载情况。  
“下载时间细分”图显示的是整个测试过程页面元素响应的时间统计分析结果，“下载时间细分(随时间变化)”显示的事场景运行过程中每一秒内页面元素响应时间的统计结果，两者分别从宏观和微观角度来分析页面元素的下载时间。  
4)、Time to First Buffer Breakdown(Over Time)（第一次缓冲时间细分(随时间变化)）  
“第一次缓冲时间细分(随时间变化)”图显示成功收到从Web服务器返回的第一次缓冲之前的这段时间，场景或会话步骤运行的每一秒中每个网页组件的服务器时间和网络时间（以秒为单位）。可以使用该图确定场景或会话步骤运行期间服务器或网络出现问题的时间。  
First Buffer Time：是指客户端与服务器端建立连接后，从服务器发送第一个数据包开始计时，数据经过网络传送到客户端，到浏览器接收到第一个缓冲所用的时间。

2、Page Component Breakdown（页面组件细分）  
“页面组件细分”图显示每个网页及其组件的平均下载时间（以秒为单位）。可以根据下载组件所用的平均秒数对图列进行排序，通过它有助于隔离有问题的组件。

3、Page Component Breakdown(Over Time)（页面组件分解(随时间变化)）  
“页面组件分解(随时间变化)”图显示在方案运行期间的每一秒内每个网页及其组件的平均响应时间 （以秒为单位）。

4、Page Download Time Breakdown（页面下载时间细分）  
“页面下载时间细分”图显示每个页面组件下载时间的细分，可以根据它确定在网页下载期间事务响应时间缓慢是由网络错误引起还是由服务器错误引起。  
“页面下载时间细分”图根据DNS解析时间、连接时间、第一次缓冲时间、SSL握手时间、接收时间、FTP验证时间、客户端时间和错误时间来对每个组件的下载过程进行细分。

DNS分析时间：LR发出协议到获取到目标IP。

连接时间：一般认为是TCP/IP“三次握手”时间。

第一次缓冲时间：一般认为是LR发出协议请求到获取第一个HTTP协议响应包。

接收时间：从获得第一个HTTP协议响应包到获得最后一个响应包。

客户端时间：客户端延时。（我还不理解是怎么计算的）

错误间：（我还不理解是怎么计算的）

注意：以上时间计算方法未必准确，需要大家在实际工作中验证。

5、Page Download Time Breakdown(Over Time)（页面下载时间细分(随时间变化)）  
“页面下载时间细分(随时间变化)”图显示方案运行期间，每一秒内每个页面组件下载时间的细分。使用此图可以确定网络或服务器在方案执行期间哪一时间点发生了问题。  
“页面组件细分(随时间变化)”图和“页面下载时间细分(随时间变化)”图通常结合起来进行分析：首先确定有问题的组件，然后分析它们的下载过程，进而定位原因在哪里。

6、Time to First Buffer Breakdown（第一次缓冲时间细分）  
“第一次缓冲时间细分”图显示成功收到从Web服务器返回的第一次缓冲之前的这一段时间内的每个页面组件的相关服务器/网路时间。如果组件的下载时间很长，则可以使用此图确定产生的问题与服务器有关还是与网络有关。  
网络时间：从接收到第一个HTTP响应开始，到接收到最后一个HTTP数据包所用的平均时间。（千万不要认为是数据在网络上的传输时间）  
服务器时间：从建立初始连接，分配第一个HTTP进程开始，到返回第一个HTTP数据包所用平均时间。（千万不要认为是服务器处理请求的时间，如果是分步式处理就不是完整服务器处理时间。）

7、Time to First Buffer Breakdown(Over Time)（第一次缓冲时间细分(随时间变化)）  
“第一次缓冲时间细分(随时间变化)”图显示成功收到从Web服务器返回的第一个缓冲之前的这段时间内，场景运行的每一秒中每个网页组件的服务器时间和网络时间。可以使用此图确定场景运行期间服务器或网络出现问题的时间点。

8、Downloader Component Size(KB)（已下载组件大小）  
“已下载组件大小”图显示每个已经下载的网页组建的大小。通过它可以直接看出哪些组件比较大并需要进一步进行优化以提高性能。

System Resource（系统资源）

Loadrunner系统资源监控能力被认为是最弱的功能，监控windows系统需要输入用户名和密码，资源监控相对全面；监控linux首先要在被监测系统安装rstatd监护进程，并且无法直接监控到内存使用率。建议使用其他工具对系统资源进行监控。重点监控CPU、内存使用率和吞吐量走势图等数据。

这些图表的含义来自网络，并不完全准确，可以自己试验。弄清楚各个图表的含义后结合图表和系统资源监控情况来确定瓶颈产生在在网络传输、CPU、内存、I/O设备、程序还是数据库。相关资料可以在网络上查找到。

**结果分析方法：**

理解上面图表含义，根据实际情况确定问题所在，一般性能测试的瓶颈不会单独在系统资源还是程序，CPU或者内存出现瓶颈，可能是程序资源使用不合理；程序出现瓶颈也可能是争夺数据资源造成。需要在时间工作中慢慢总结，我可分享的经验也不多。

## 性能测试流程：

### 第一：建立压力模型。

确定用户访问量，用户并发数量，模块分部情况，用户操作习惯和基础数据量等等。

平均并发用户数量=用户总量\*在线操作时间/在线时间（这个公式并不是很实用）

平均并发用户数量通常取值在用户总量/10 到用户总量/5 之间。

并发持续时间：根据具体情况进行确定，我们知道用户最大在线数量和期望最大并发数量，我们可以采用集合点控制并发数量而不考虑持续并发时间；如果脚本业务流程比较短，系统响应比较快，持续并发时间可以根据时间使用情况设置；如果业务很长，响应时间比较大，我们用平均用户并发数和持续并发时间做参考意义不大，这时可以考虑服务器单位时间内产生的交易量和用户数量设置场景。

### 第二：准备基础数据。

通过LR脚本或者其它方式生成数据。

基础数据量直接决定了我们测试结果与生产环境表现的相似程度，所以基础数据的准备很大程度上也决定了性能测试结果的可信度。

准备基础数据时尽可能是使用工具从前台注入数据，不建议直接通过数据库写入数据。

### 第三：开发性能测试脚本。

（创建脚本中已经介绍，重点是设置事务、集合点、参数化、关联和检查点，其他功能在需要可以通过网络和LR帮助获取）

### 第四：设置运行场景。

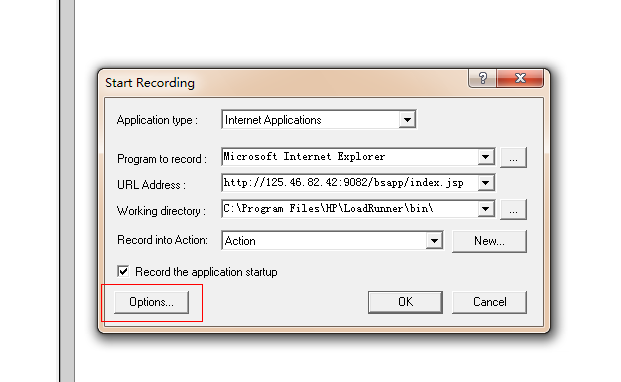
设置用户增加方式，持续时间，减少方式，日志输出方式，思考时间，监控服务器，并且可以通过集合点策略控制并发用户的数量或者百分比等等。

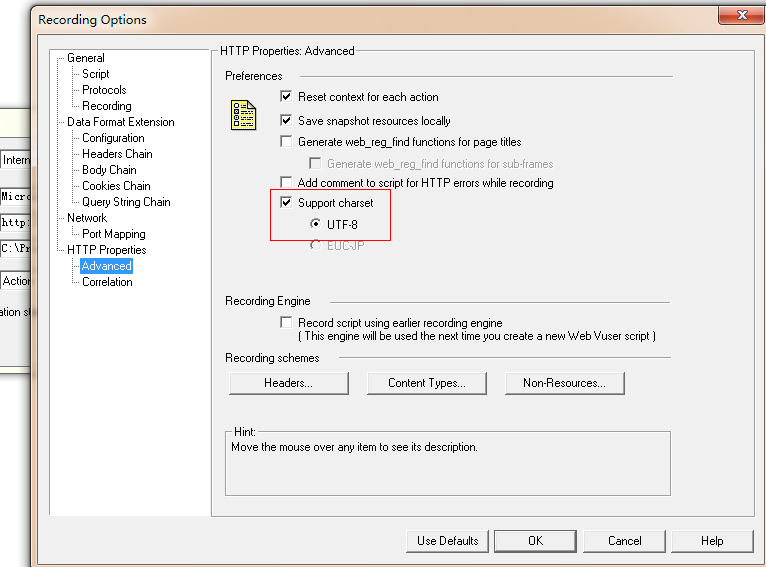
### 第五：性能测试结果分析。

（结果分析中已经介绍，重点是对图表含义的理解结合系统情况，用现有知识做出测试结果分析）

# 常见问题：

**乱码问题：**Start Recording 设置UTF-8





乱码问题是多种因素造成，如果该方法不能解决建议使用LR11，支持中文比较好。如果还是无法解决，如果需要进行参数化使用字符强制转换，不过不能设置检查点。如果不需要有特殊操作可以不用理会乱码。

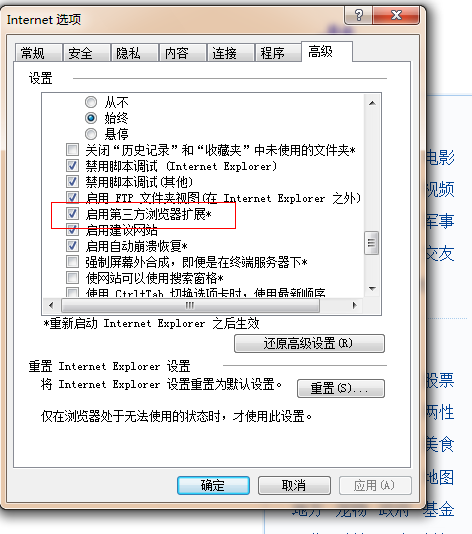
转换方式：

lr\_convert\_string\_encoding( lr\_eval\_string("{custaddress}")

LR\_ENC\_SYSTEM\_LOCALE LR\_ENC\_UTF8 "str" );

函数用法请查询帮助。

**启动刻录无法打开IE：**勾选“启用第三方浏览器扩展\*”，也可能是无法兼容，更换更高版本loadrunner。



**自动扫描关联完成后运行脚本失败：**首先检查关联是否都正确，然后从新扫描关联。因为关联一次很难完整扫描出所有关联，需要反复多次。

**关联完成回放报错：**Error -26377: No match found for the requested parameter "Siebel\_Analytic\_ViewState2". Check whether the requested boundaries exist in the response data. Also, if the data you want to save exceeds 1024 bytes, use web\_set\_max\_html\_param\_len to increase the parameter size

解决办法：，1.关联错误，输出关联值看看是否正确。2.关联的字符串超过1024B，在关联函数前添加函数“web\_set\_max\_html\_param\_len("10240");”,扩充关联函数字符串长度，“10240”代表字符串长度。3.页面返回错误，在返回页面设置检查点，确保返回页面正确与否。

**刻录过程中录不到事务：**可能是协议选择不合适，更换刻录协议。

# 综述：

性能测试工作是一项繁琐而且不断积累改进的工作，它要求测试人员拥有多方面知识，比较重要的知识包括对于网络环境、数据库、程序的工作原理和特性的了解加上操作系统特性与对硬件操作原理的了解，这样才能比较好的分析测试结果，我们需要学习的东西还很多，学习性能测试是通过学习性能测试理论、再学习测试工具现实理论、通过对测试工具的理解和相关知识的理解来加深对性能测试理论的理解这样一个循环的过程，在以后的日子里大家共同努力奋斗吧！