使用 WAS 对 Web 应用程序进行压力测试.

刘艳会

WAS（Microsoft Web Application Stress Tool，Web 应用负载测试工具）提供了一种简单的方法模拟大量用户来访问你的网站。这个工具能告诉我们你的 Web 应用程序工作时对硬件和软件的使用情况。在本文中我将告诉大家如何使用 WAS，以及如何理解 WAS 测试的数据。

1 压力测试的必要性

随着服务器端处理任务的日益复杂以及网站访问量的迅速增长，服务器性能的优化也成了非常迫切的任务。在优化之前，最好能够测试一下不同条件下服务器的性能表现。找出性能瓶颈所在是设计性能改善方案之前的一个至关紧要的步骤。

负载测试是任何 Web 应用的开发周期中一个重要的步骤。如果你在构造一个为大量用户服务的应用，搞清楚你的产品配置能够承受多大的负载非常重要。如果你在构造一个小型的 Intranet 网站，测试能够暴露出最终会导致服务器崩溃的内存漏洞以及竞争情况。

但是在实际的开发过程中，要按照实际投入运行的情况，组织成千上万的用户来进行压力测试，无论从那个方面看，都是不现实的。而且这样一旦发现了问题，不仅需要重复的进行这种耗费巨大的测试，而且问题不容易重现，不能方便的找出性能的瓶颈所在。而使用软件进行压力测试就不会存在这种情况。

无论是哪种情形，花些时间对应用进行负载测试可以获得重要的基准性能数据，为未来的代码优化、硬件配置以及系统软件升级带来方便。即使经费有限的开发组织也可以对它们的网站进行负载测试，因为 Microsoft 的压力测试工具 WAS 是可以免费下载的。

2 WAS 概要介绍

为了有效的对 Web 应用程序进行压力测试，Microsoft 发布了这个简单易用，功能强大的工具 WAS。WAS 要求 Windows NT 4.0 SP4 或者更高，或者 Windows 2000。为了对网站进行负载测试，WAS 可以通过一台或者多台客户机模拟大量用户的活动。WAS 支持身份验证、加密和 Cookies，也能够模拟各种浏览器类型和 Modem 速度，它的功能和性能可以与数万美元的产品相媲美。

使用 WAS 时，为了更加接近真实的进行压力测试，我们推荐运行 WAS 的测试机和 Web Server 分开。

3 开始使用 WAS

要对网站进行负载测试首先必须创建 WAS 脚本模拟用户活动。我们可以用下面四种方法之一创建脚本：

- 通过记录浏览器的活动
- 通过导入 IIS 日志
- 通过把 WAS 指向 Web 网站的内容
- 手工制作

在这里我们拿最常用的方法——通过记录浏览器的活动 来讲解。其他三种方法在后面将会提到。

3.1 录制测试脚本

在录制测试脚本前，需要首先关闭 IE 的缓冲区。

1. 在工具菜单，点 Internet 选项
2. 点常规标签，然后点删除文件。。。按钮。

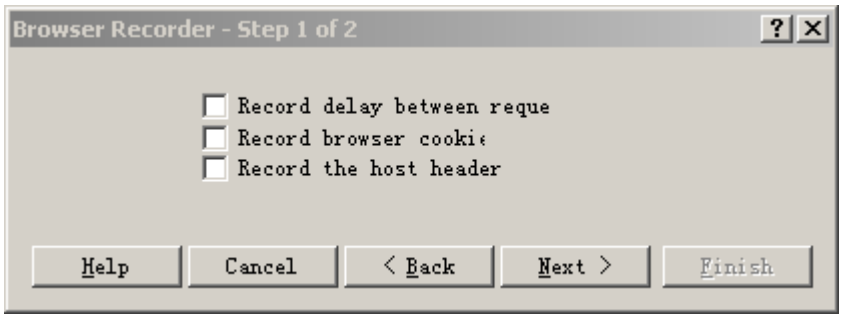
如果使用 IE5。0 或以上版本则不需要修改代理设置，因为 5。0 以上版本的 IE 允许 WAS 改变这些设置。然而，对于 IE4。0 或早期版本，WAS 使用一个内置的代理服务器来记录浏览器活动。

按 WAS 的需要指定代理设置

1. 在工具菜单，点 Internet 选项
2. 在连接标签里，修改代理设置以使代理服务器指向 Localhost 并且使用端口 8000
3. 不选对于本地地址不使用代理服务器

打开菜单，选择 Scripts|Create|Record 创建一个测试脚本

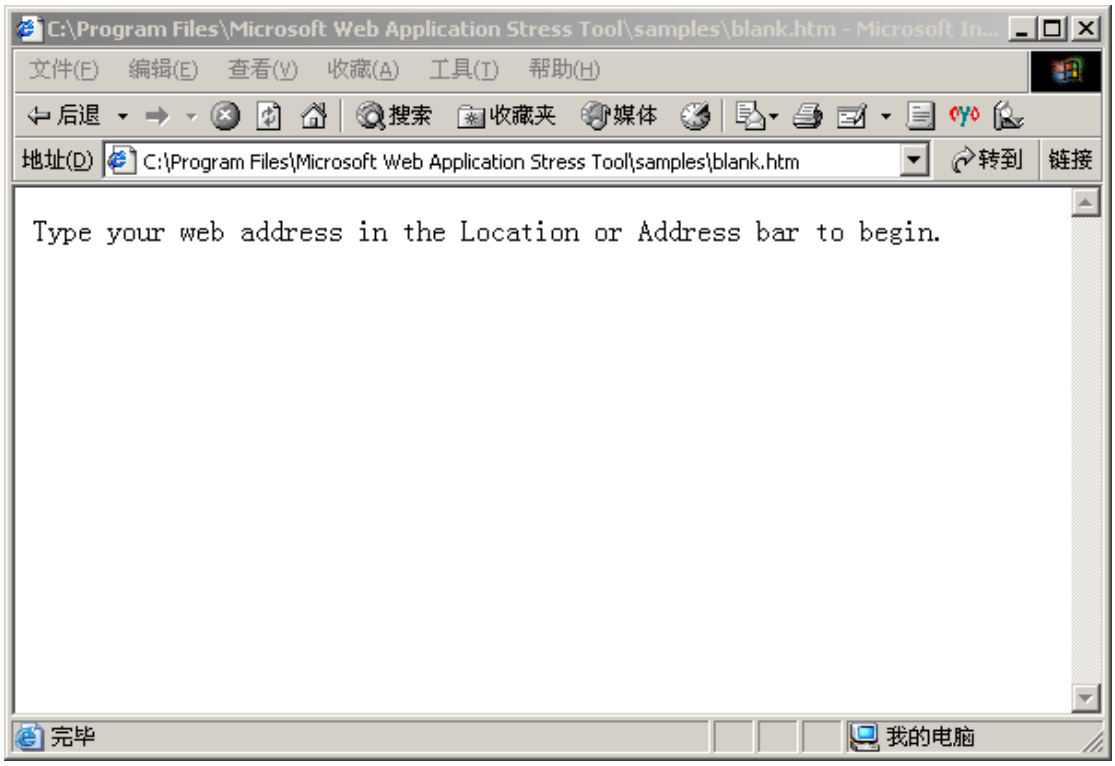
选取要记录的内容，有下面 3 种



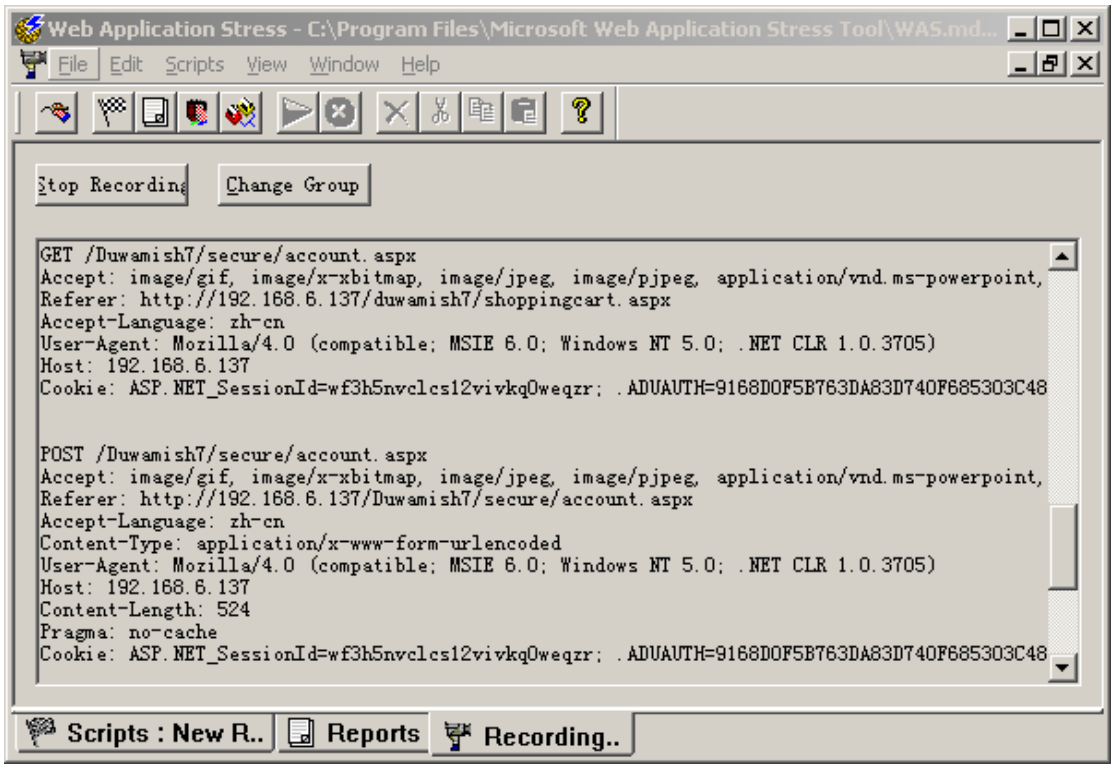
- Record delay between request：记录了请求之间的延迟。由于用户实际上在浏览网站时，请求之间存在几秒甚至几分钟的延迟，这种录制方法在执行时会模仿用户之间的延迟发送请求，所以会是一个更加实际的测试。如果我们的目的是要发现 Web 应用程序的承受极限，就不要选择该项；如果只是想模拟一个特定数量的用户场景，那么选择该项进行测试捕捉请求延迟。
- Record browser cookies & Record the host header：只记录用户的会话，不记录延迟时间。一般情况下，我们不需要选择这两项，可以让 WAS 创建 cookies 和 host header，就好像用户登陆你的网站一样。然而，如果你有网站的回归信息时（比如一个用户的主要特征信息或者与一个永久性 cookies 相连的其他信息），在模拟一个新的用户登陆网站和进行必要的用户配置测试前，必须保证清除 cookies，如果 Web 应用程序需要用户接受 cookies，那么需要选中该选项。

目前这个版本的 WAS 软件对基于浏览器 IE 录制脚本的方式还不支持 HTTP/SSL 请求。一般情况下，只选择后二种会增加压力的强度。

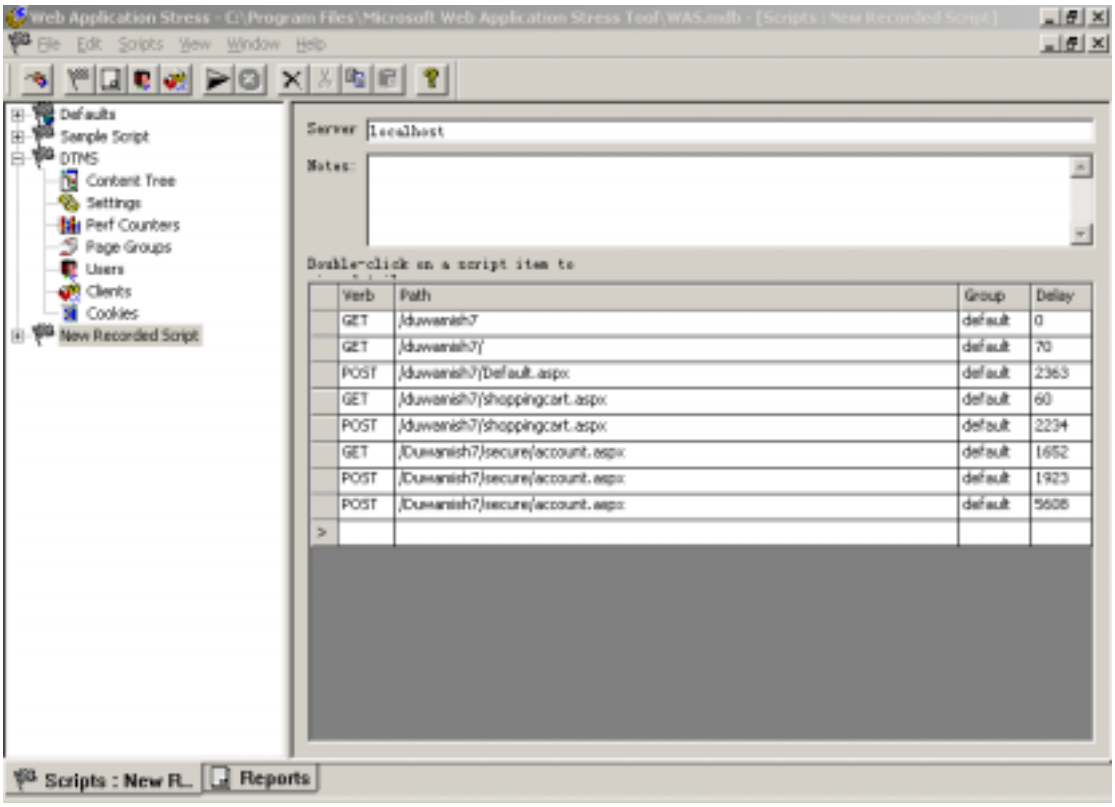
根据压力测试实际的情况，选择合适的选项，然后点“ Next | Finish ”，WAS 会打开一个 IE 窗口，在 IE 中输入要测试的站点地址，然后我们就可以按照实际的情况开始浏览站点了，浏览的同时也就是执行测试用例的过程。



等测试用例执行完成后，切换到 WAS 窗口，点“ Stop Recording: ”按钮，停止录制脚本



WAS 回到了视图页面，在该页面中你可以看到在录制过程中 WAS 收集的每一个链接，而且还可以编辑 GET、POST 以及 HEAD 信息。



制作 WAS 脚本是相当简单的，不过要制作出模拟真实用户活动的脚本有点儿复杂。如果你已经有一个运行的 Web 网站，可以使用 Web 服务器的日志来确定 Web 网站上的用户点击分布。如果你的应用还没有开始运行，那么只好根据经验作一些猜测了。

3.2 负载参数设置

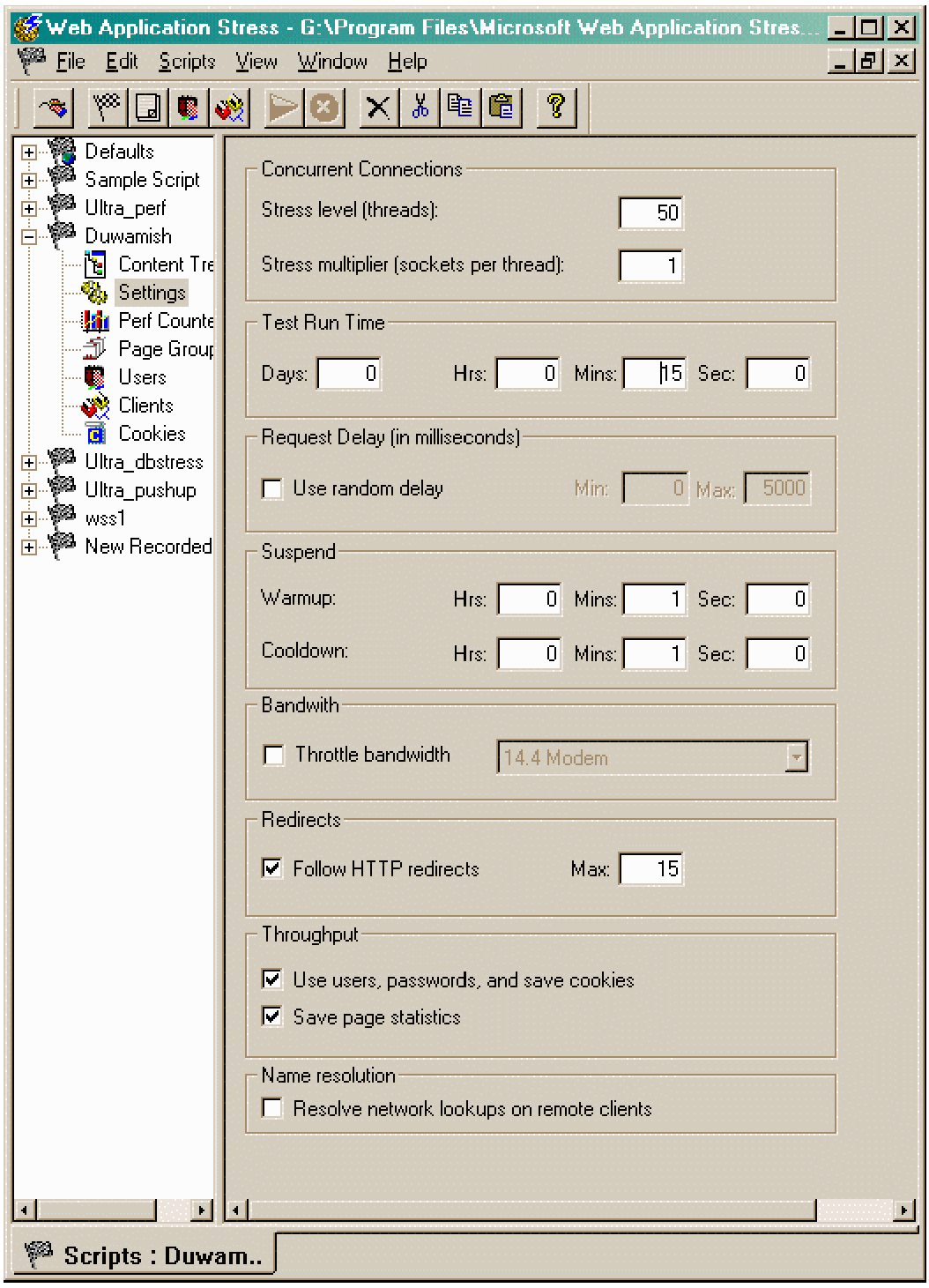
测试脚本录制完成后，下一步我们要作的就是配置运行脚本的负载选项，我们可以调整测试配置以便观察不同条件下的应用性能。

3.2.1 Content Tree

由于我们的 WAS 和 Web Server 是分开的，所以这里我们不需要设置

3.2.2 Setting（设置）

我们只要点击“ Setting ”就开始负载选项设置。



1. ConCurrent Connections：

   Stress Level（threads）的数值决定了所有客户机创建的 Windows 的线程的数值。每一个线程创建多个 Socket 连接（具体多少 Socket 连接数取决于 Stress multiplier (sockets per thread)），每个 Socket 连接就是一个并发的请求(request)。下面这个公式表示了它们之间的关系：

   总的并发请求数 = Stress Level * Stress multiplier = 总的 Socket 连接数

   Stress Level 和 Stress multiplier 这二个项决定了访问服务器的并发连接的数量。

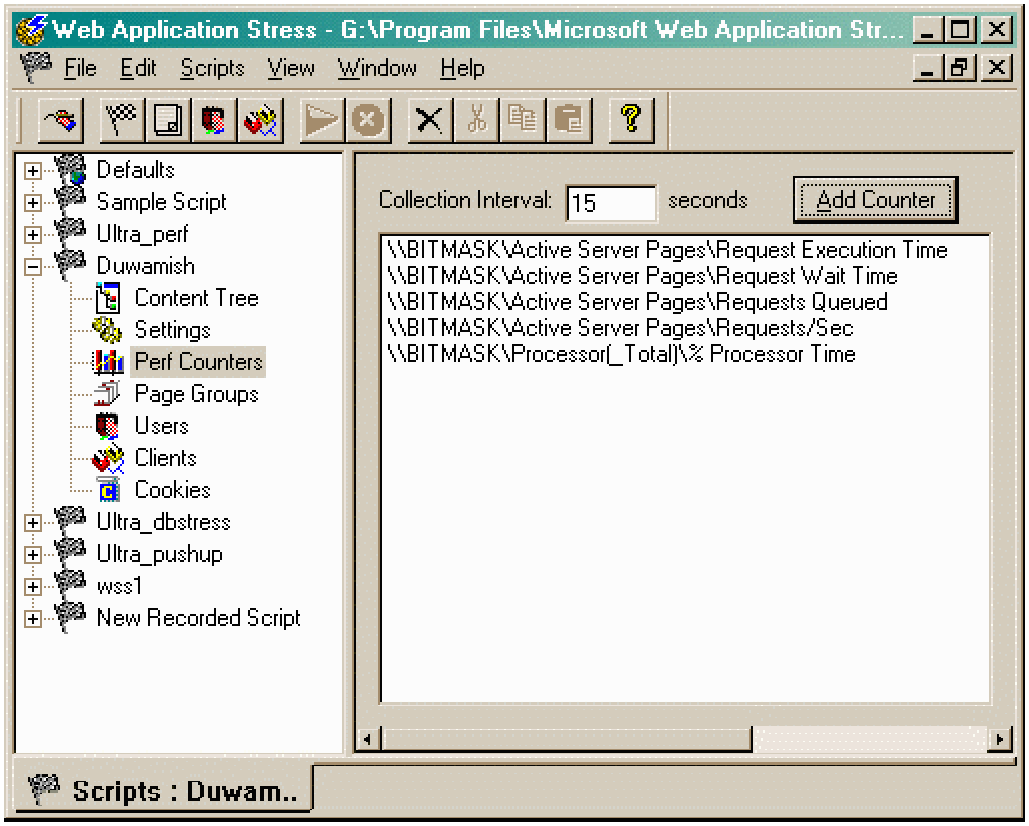
Microsoft 建议不要选择超过 100 的 Stress Level 值。如果要模拟的并发连接数量超过 100 个，可以调整 Stress multiplier 或使用多个客户机。在负载测试期间 WAS 将通过 DCOM 与其他客户机协调。

2. Test Run Time：设定持续运行多长时间的测试。我们可以在这里设定让 WAS 持续运行多少天、多少个小时、多少分钟、多少秒
3. Request Delay（in milliseconds）：设定请求延迟时间的最大、最小值，当然我们也可以选择“ Use random delay ”使用随机的延迟时间。一般情况下，我们常常会浏览一页，发现一个链接后，我们点击它。即便对该网站熟悉的人，
4. Suspend：WAS 允许设置 warmup（热身）时间，一般可以设置为 1 分钟。在 warmup 期间 WAS 开始执行脚本，但不收集统计数据。warmup 时间给 MTS、数据库以及磁盘缓冲等一个机会来做准备工作。如果在 warmup 时间内收集统计数据，这些操作的开销将影响性能测试结果。WAS 也允许设置 CoolDown 时间。在 WAS 执行的时间达到设定的 Test Run Time 时，进入 CoolDown Time，这时 WAS 并没有停止执行脚本，同样也不会收集统计数据。下图表示了它们的先后关系。

5. Bandwith：设置页面提供的另外一个有用的功能是限制带宽（throttle bandwidth）。带宽限制功能能够为测试模拟出 Modem（14.k K，28.8 K，56 K）、ISDN（64 K，128 K）以及 T1（1.54 M）的速度。使用带宽限制功能可以精确地预测出客户通过拨号网络或其他外部连接访问 Web 服务器所感受的性能。
6. Redirects、Throughput、Name resolution：
   这几个选项一般情况下采用默认情况即可。
   选中 Follow HTTP redirects 选项将会支持重定向。
   选中 Throughput 中的两项，WAS 将会收集活动用户的 cookies，以及收集网站的统计数字。默认情况下都会选中这两项，如果不选择，将会增加压力测试的强度。
   Name resolution 默认情况下没有选中。选中该选项，会让每一个客户测试机执行查询，只有在使用多个子网时才需要选中该项。（帮助原文：have each individual test client perform a lookup，this is useful when using multiple subnets）

3.2.3 Perf Counters（性能计数器）

使用 WAS，从远程 Windows NT 和 Windows 2000 机器获取和分析性能计数器（Performance Counter）是很方便的。加入计数器要用到下图所示的 Perf Counters 分枝。



一般情况下，这里需要添加的性能计数器有：

1 Web Server：

- 处理器：CPU 使用百分比（% CPU Utilization）
- 内存：内存使用百分比（% Memory Utilization）
- 线程：每秒的上下文切换次数（Context Switches Per Second (Total)）
- ASP：每秒请求数量（Requests Per Second）
- ASP：请求执行时间（Request Execution Time）
- ASP：请求等待时间（Request Wait Time）
- ASP：置入队列的请求数量（Requests Queued）

2 各个 WAS 测试机

- 处理器：CPU 使用百分比（% CPU Utilization）
- 内存：内存使用百分比（% Memory Utilization）

在测试中选择哪些计数器显然跟测试目的有关。虽然下面这个清单不可能精确地隔离出性能瓶颈所在，但对一般的 Web 服务器性能测试来说却是一个好的开始。

- 处理器：CPU 使用百分比（% CPU Utilization）
- 线程：每秒的上下文切换次数（Context Switches Per Second (Total)）
- ASP：每秒请求数量（Requests Per Second）
- ASP：请求执行时间（Request Execution Time）
- ASP：请求等待时间（Request Wait Time）
- ASP：置入队列的请求数量（Requests Queued）

CPU 使用百分比反映了处理器开销。CPU 使用百分比持续地超过 75%是性能瓶颈在于处理器的一个明显的迹象。每秒上下文切换次数指示了处理器的工作效率。如果处理器陷于每秒数千次的上下文切换，说明它忙于切换线程而不是处理 ASP 脚本。

每秒的 ASP 请求数量、执行时间以及等待时间在各种测试情形下都是非常重要的监测项目。每秒的请求数量告诉我们每秒内服务器成功处理的 ASP 请求数量。执行时间和等待时间之和显示了反应时间，这是服务器用处理好的页面作应答所需要的时间。

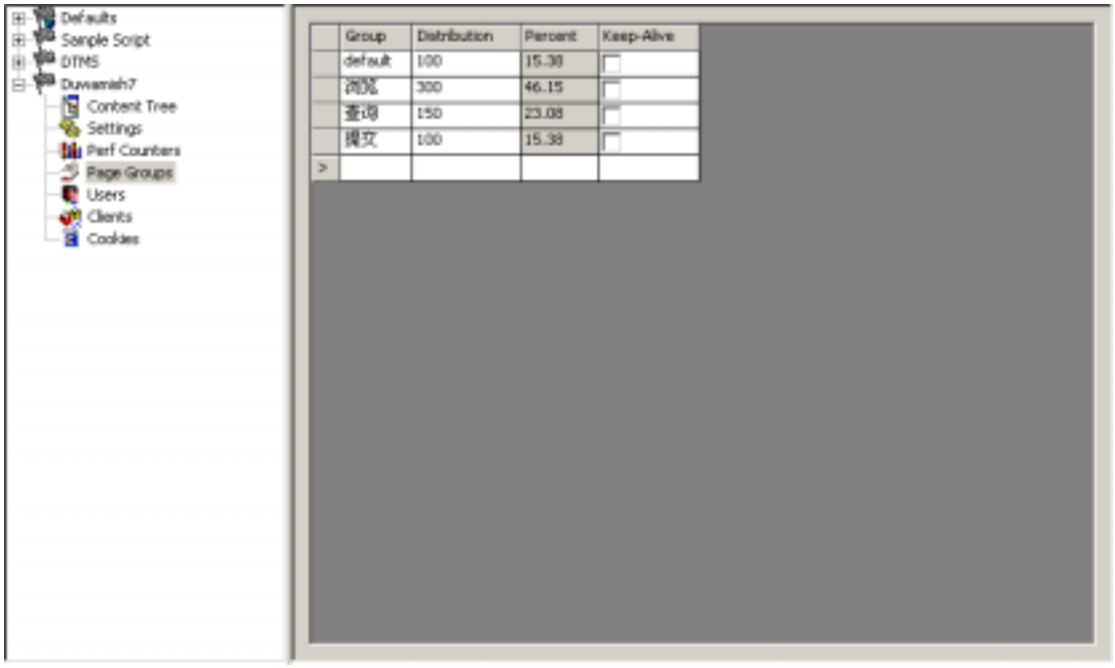
我们可以绘出随着测试中并发用户数量的增加每秒请求数量和反应时间的变化图。增加并发用户数量时每秒请求数量也会增加。然而，我们最终会达到这样一个点，此时并发用户数量开始“压倒”服务器。如果继续增加并发用户数量，每秒请求数量开始下降，而反应时间则会增加。要搞清楚硬件和软件的能力，找出这个并发用户数量开始“压倒”服务器的临界点非常重要。

置入队列的 ASP 请求数量也是一个重要的指标。如果在测试中这个数量有波动，某个 COM 对象所接收到的请求数量超过了它的处理能力。这可能是因为在应用的中间层使用了一个低效率的组件，或者在 ASP 会话对象中存储了一个单线程的单元组件。

运行 WAS 的客户机 CPU 使用率也有必要监视。如果这些机器上的 CPU 使用率持续地超过 75%，说明客户机没有足够的资源来正确地运行测试，此时应该认为测试结果不可信。在这种情况下，测试客户机的数量必须增加，或者减小测试的 Stress Level。

3.2.4 Page Groups

对于一个 Web 应用而言，同一时刻用户点击分布是不一样的。WAS 允许设置用户点击流量的分布比例。

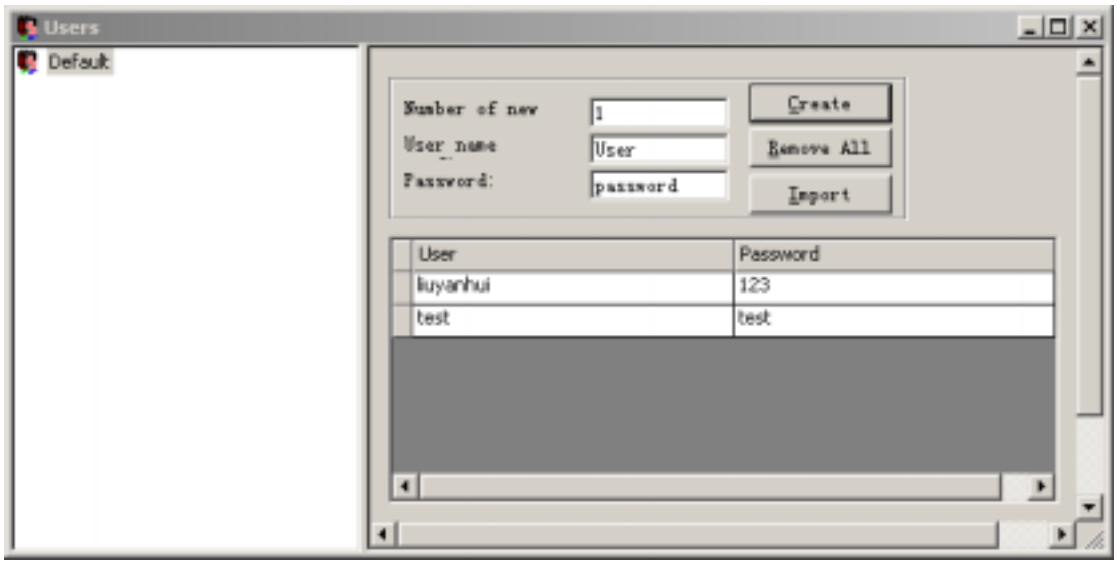


这里我们假设在一个 Web 应用程序中，有 650 个人同时在线，其中 100 人正在添加提交数据，占 15.38%；有 150 人正在查询，占 23.08%。

按照不同的 Web 应用，我们可以根据实际的情况在定制这个比例关系，来更加符合实际的情况。

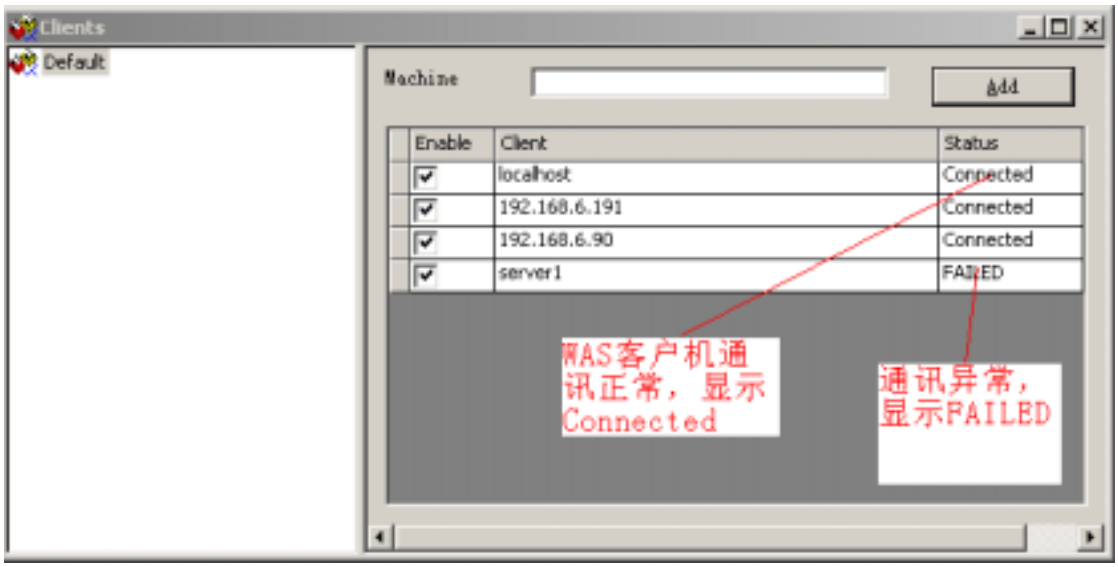
3.2.5 Users

现在很多 Web 应用程序为了提供个性化的服务，都设计了登陆过程。每个用户都有自己的登陆名和密码。WAS 也考虑到了这种情况，我们只要在 Users 分支中添加用户名和对应的密码即可。



3.2.6 Clients

添加多个 WAS 客户机。在运行期间，各个 WAS 客户机是通过 DCOM 来协调的。各个 WAS 客户机只要正确安装了 WAS 软件，启动了 WebTool 服务，它们就可以自己协调操作。我们只要在 Clients 分支内添加 WAS 客户机即可。



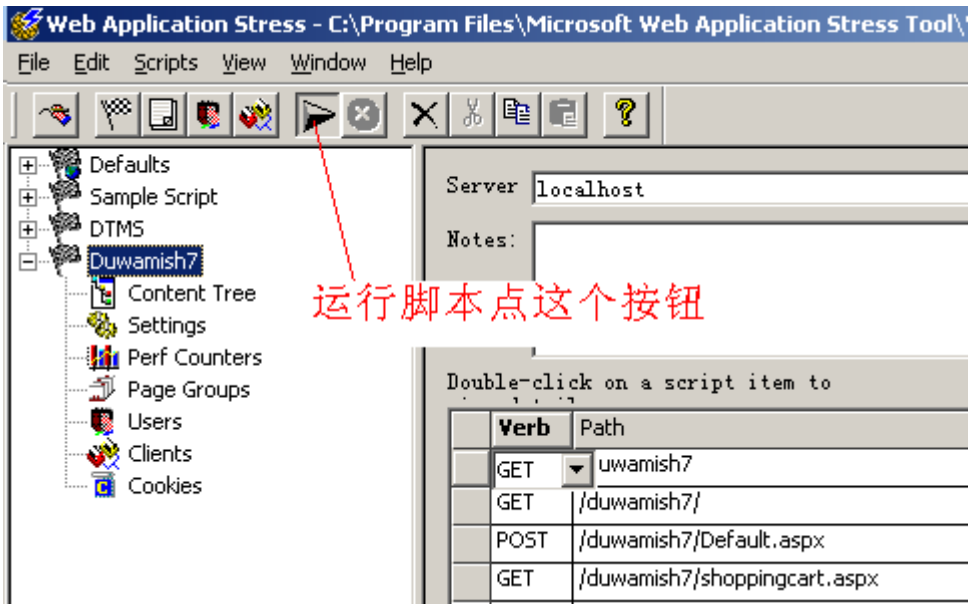
3.2.7 Cookies

这里显示的是用户名以及对应的 cookies。这里不需要设置。

4 运行测试脚本

所有的设置完成以后，我们就可以运行 WAS 来进行压力测试了。要运行测试脚本很简单，只要选中测试脚本的名称，然后点工具栏上的“ 运行 ”按钮，即可。

建议：第一次运行测试脚本时，Test Run Time 不要太长，Stress Level 以及 Stress multiplier 不要太大。第一次运行的目的只是为了检验测试脚本正确的运行。



5 测试结果

每次测试运行结束后 WAS 会生成详细的报表，即使测试被提前停止也一样。WAS 报表可以从 View 菜单选择 Reports 查看。下面介绍一下报表中几个重要的部分。

5.1 摘要

页面摘要部分提供了页面的名字，接收到第一个字节的平均时间（TTFB），接收到最后一个字节的平均时间（TTLB），以及测试脚本中各个页面的命中次数。TTFB 和 TTLB 这两个值对于计算客户端所看到的服务器性能具有重要意义。TTFB 反映了从发出页面请求到接收到应答数据第一个字节的时间总和（以毫秒计），TTLB 包含了 TTFB，它是客户机接收到页面最后一个字节所需要的累计时间。

只要选中页面的名字，即可显示页面概要。

Overview

Report name:	2003-4-16 15:57:15 ——页面的名字
Run on:	2003-4-16 15:57:15
Run length:	00:01:00

Web Application Stress Tool Version:1.1.288.1

Notes

Sample Microsoft Web Application Stress Script

Number of test clients:	4
Number of hits:	8526
Requests per Second:	141.86

Socket Statistics

Socket Connects:	8575
Total Bytes Sent (in KB):	1736.48
Bytes Sent Rate (in KB/s):	28.89
Total Bytes Recv (in KB):	25880.91
Bytes Recv Rate (in KB/s):	430.63

Socket Errors

Connect:	0
Send:	0
Recv:	0
Timeouts:	0

RDS Results

Successful Queries:	0

Script Settings

Server:	localhost
Number of threads:	50
Test length:	00:01:00
Warmup:	00:00:00
Cooldown:	00:00:00
Use Random Delay:	Yes
Min Delay Time:	20
Max Delay Time:	40
Follow Redirects:	Yes
Max Redirect Depth:	15

Clients used in test

192.168.6.191
192.168.6.90
localhost

Clients not used in test

server1

Result Codes

Code	Description	Count
404	Not Found	8526

Page Summary

Page	Hits	TTFB Avg	TTLB Avg	Auth
GET /samples/cookie.asp	1244	290.90	291.65	No
POST /samples/post.asp	1231	304.86	348.67	No
GET /samples/browser.asp	1229	299.05	342.42	No
GET /samples/fileacc.asp	1221	296.70	340.28	No
GET /samples/htmltest.htm	1214	289.59	289.76	No
GET /samples/ad_test.asp	1197	292.89	293.63	No
GET /samples/logo.jpg	1190	295.84	296.05	No

各个页面的点击次数　TTFB　TTLB

5.2 Result Codes

如果这是一个新创建的测试脚本，你应该检查一下报表的 Result Codes 部分。这部分内容包含了请求结果代码、说明以及服务器返回的结果代码的数量。如果这里出现了 404 代码（页面没有找到），说明在脚本中有错误的页面请求。具体的错误代码表示的意义，可以参考 IIS 的说明文档。

Result Codes

Code	Description	Count
404	Not Found	4280
NA	HTTP result code not given	15

5.3 Perf Counters

报表中还包含了所有性能计数器的信息。这些数据显示了运行时各个项目的测量值，同时还提供了最大值、最小值、平均值等。报表实际提供的信息远远超过了我们这里能够介绍的内容。

Number of measurements:	6
Computer:	\\YANHUIL
Object:	Processor
Instance:	_Total
Counter:	% Processor Time
Average:	69.53
Min:	50.00
25th Percentile:	61.48
50th Percentile:	62.42
75th Percentile:	67.29
Max:	100.00
Computer:	\\YANHUIL
Object:	ASP.NET
Counter:	Request Execution Time
Average:	0.00
Min:	0.00
25th Percentile:	0.00
50th Percentile:	0.00
75th Percentile:	0.00
Max:	0.00
Computer:	\\YANHUIL
Object:	ASP.NET
Counter:	Request Wait Time
Average:	0.00
Min:	0.00
25th Percentile:	0.00
50th Percentile:	0.00

5.4 Script Settings

这里显示的是运行本次测试时的设置，也就是前面讲到的 Setting 部分的内容。

5.5 Test Clients

这里显示的是各个 WAS 客户机的情况。先总体说明在测试中使用了那些 WAS 客户机，在使用的 WAS 客户机中显示

- 执行了多少线程
- 模拟了多少用户
- 点击的次数
- 连接失败的次数

```
Client machine:  192.168.6.191
=================================================================
Number of threads:              1
Number of users:                19
Hit Count:                      1274
Connect Failures:               5352
```

5.6 Page Summary

显示了在测试中各个请求内容的 TTFB 和 TTLB，以及点击的次数等信息。具体的说明已经包含在 5.1 摘要中

5.7 Page Groups

显示不同的用户组在测试中的执行情况。这里提供的信息包括

- 用户组的分布情况，以及在所有用户组中所占的比例
- 点击的次数，以及在所有点击次数中所占的比例
- Result Codes 情况
- Socket 连接的信息

```
Group Results
-----------------------------------------
Distribution:                   100
% Total Distribution:           %50.00

Hit Count:                      7000
% Total Hits:                   %28.88

Result Codes
Code      Description                   Count
-----------------------------------------
404       Not Found                     7000

Socket Statistics
-----------------------------------------
Socket Connects:                7007
Total Bytes Sent (in KB):       2258.98
Bytes Sent Rate (in KB/s):      12.53
Total Bytes Recv (in KB):       21246.09
Bytes Recv Rate (in KB/s):      117.83
```

5.8 Page Data

显示了各个请求内容的更加详细的信息。技术需求中的运行效率信息可以在这里验证。

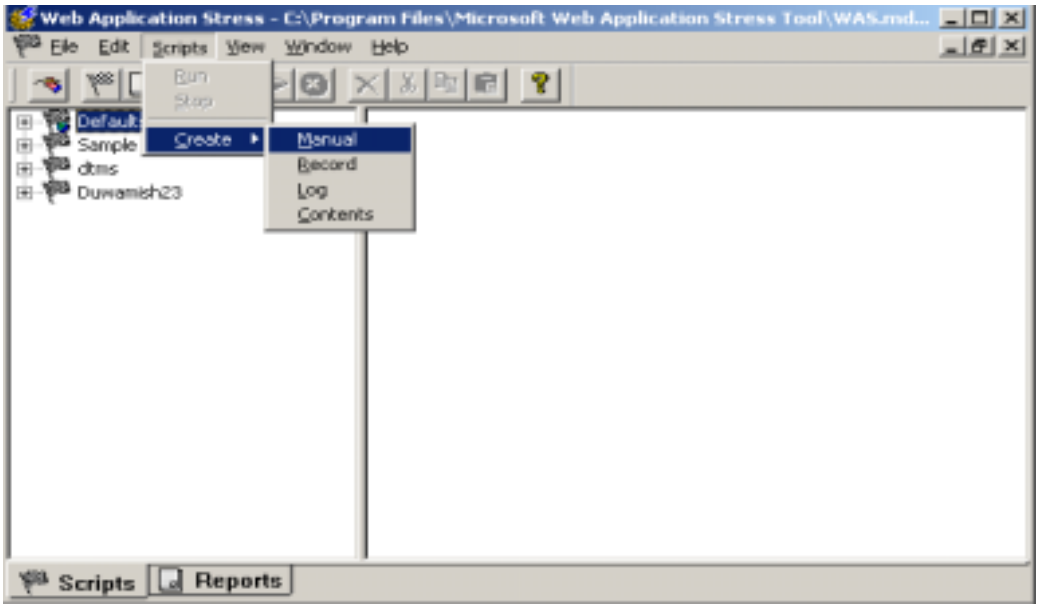
25%、50%、75%的数字还没有弄明白什么意思？？？？？平均值还不知道怎样计算出来的

6 其他方式编写测试脚本

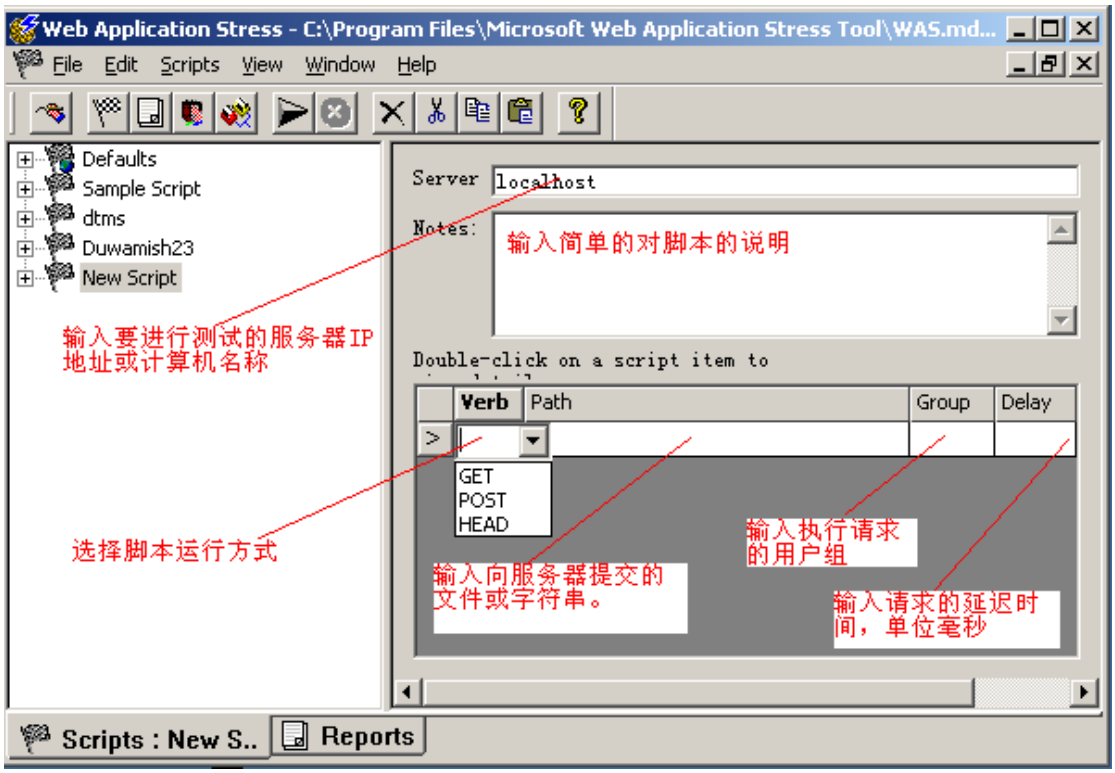
在前边提到，编写测试脚本有 4 种方法，现在对其他三种方法进行简单的介绍

6.1 手动编写测试脚本

打开菜单，选择 Scripts|Create|Manual 手动创建一个测试脚本



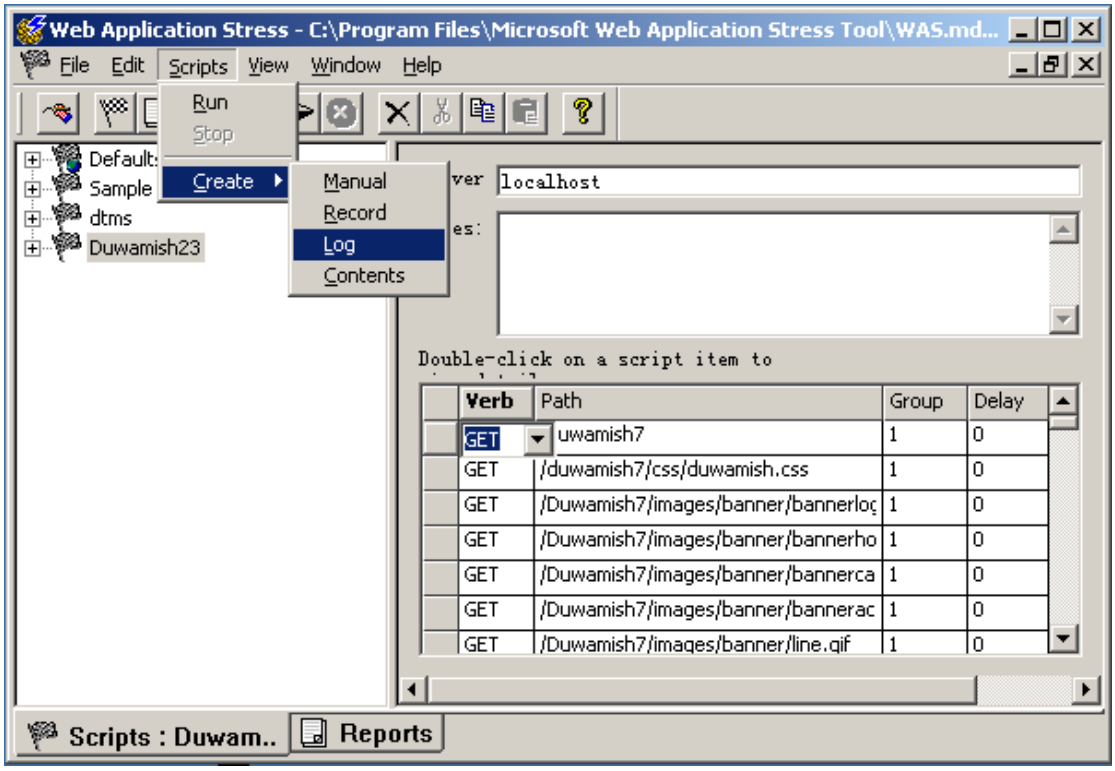
然后出现了 NewScript，[server]中输入要进行测试的服务器 IP 地址或计算机名称；在脚本的内容表格中[verb]项选择脚本运行方式 get、post、head；[path]中输入向服务器提交的文件或字符串。



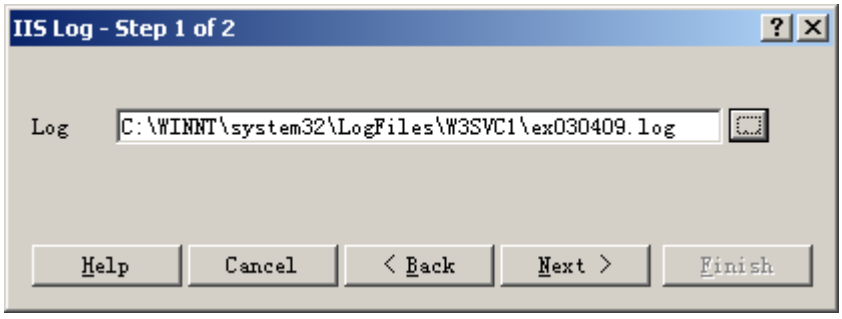
6.2 导入 IIS 日志

这种方法适合于开始投入运行的 Web 应用程序。IIS 日志记录了用户访问系统的所有信息。通过导入 IIS 日志的方法建立的测试脚本，是最符合实际运行情况的方法。如果有 IIS 日志，我们推荐使用这种方法。

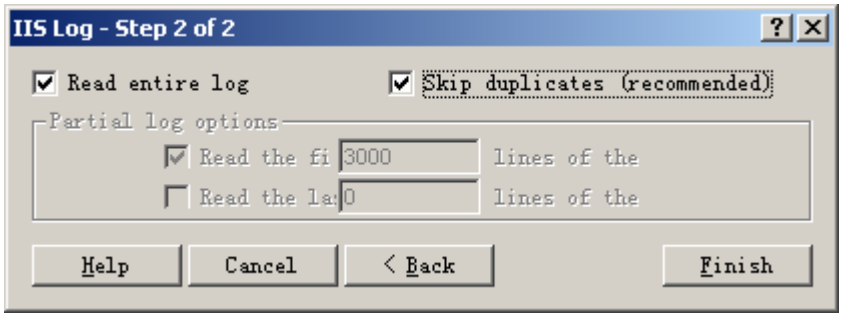
这种方法也比较简单。打开菜单，选择 Scripts|Create|Log 导入 IISs 日志创建一个测试脚本。



然后出现导入 IIS 日志的第一步，选择 IIS 日志的路径，默认情况下的路径如图所示



Next 进入第二步，一般情况下不用做改动。取默认即可

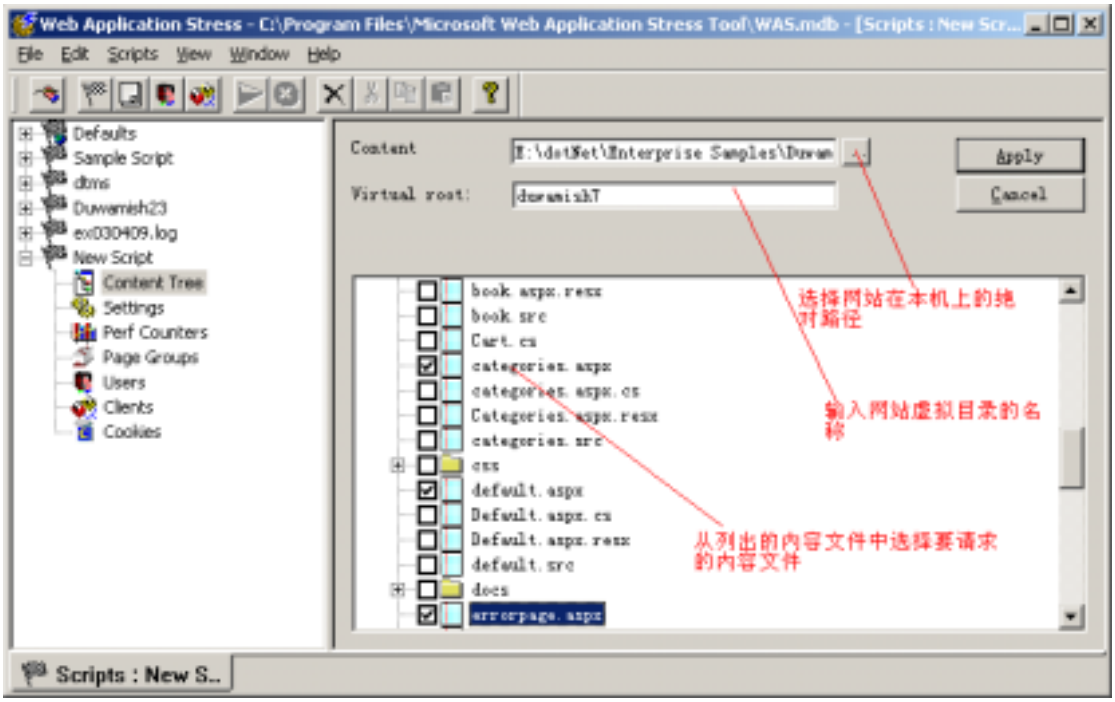


Finish 后，WAS 自动生成脚本。

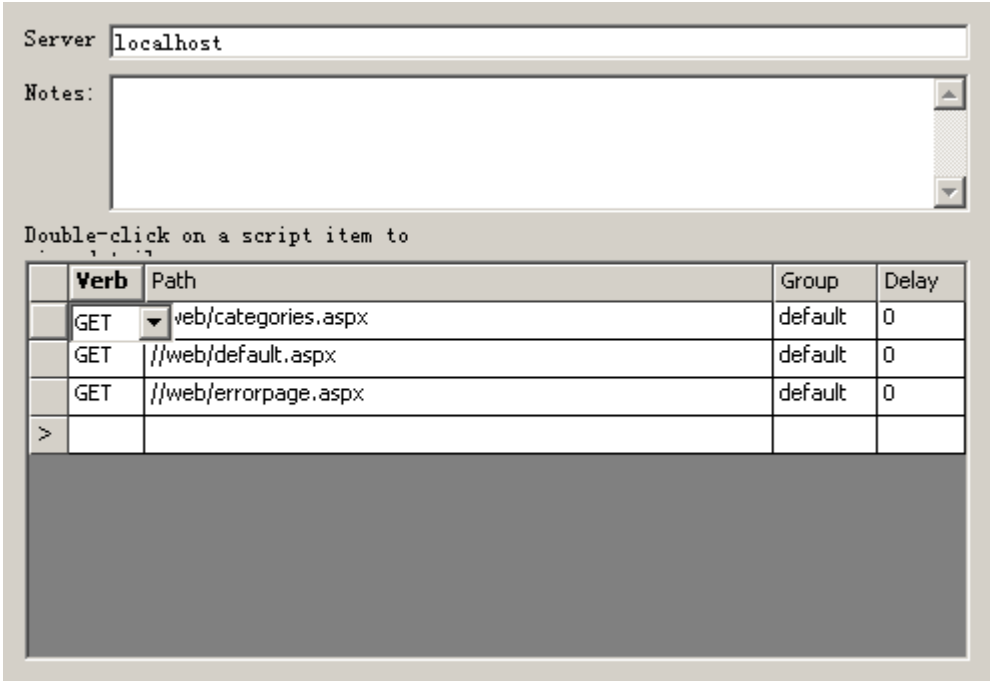
6.3 导入网站内容文件

这种方法通过导入网站上具体的文件来生成测试脚本。一般情况下，不推荐使用这种方法。下面简单说明这种方法的使用。

打开菜单，选择 Scripts|Create|Contents ，WAS 自动新建一个测试脚本，并且切换到 Contents Tree 节点。



然后回到 New Script 的主页面，会看到选择的内容文件自动添加到表格中



使用方法就是这样。

7 初步的想法

1 在大规模的测试时，需要很多 WAS 客户端。为了充分利用资源，可以在项目组每个

人的机器上都安装 WAS 软件（只要正确安装，启动 WebTool 服务即可）。这样测试时，WAS 会自动协调，自动分配线程。

2 我们的系统中有很多的角色。虽然 WAS 可以将不同角色执行请求的顺序进行混合，但是这样我们不知道 WAS 怎样混合的，不能随时控制角色的状态。建议将不同的角色分组，每个角色放到一个 WAS 测试机（充当控制器）上，这样可以分角色而又集中的对 WebServer 进行压力测试，同时又能随意的控制各个 WAS 客户机的状态。这种思想是模仿 LoadRunner 软件，具体实施还需要不断的实验和学习。

3 在集成测试时，先注重性能方面的测试，逐步的加压，寻找 WebServer 的最大负载量。进而对照技术需求，不断改进。

4 WAS 首先是性能测试工具，然后才是压力测试工具。具体测试时，可以先在正常的条件下（满足技术需求）进行性能测试，然后才是异常情况（增加压力到技术需求规定的 1.5-2 倍）的测试。

5 不断的研究学习，利用 WAS 更深入地了解你的应用的性能、稳定性、瓶颈和局限性。

8 存在的问题

目前存在的问题：

1 在测试结果 Report 中的 Page Data 部分，25%、50%、75%的值到底是什么意思？Average 又是怎样计算出来的

2 在关于身份验证的页面中，如何模拟多用户使用不同的账号登陆？按照帮助说明和例子的说明，在 ASP.NET 开发的登陆程序中没有试验成功。（帮助中这么说明：取用户名是在 POST 数据项中把用户名用“ %Username% ”代替，密码用“ %Password% ”代替）

3 添加性能计数器时，添加本机的当然没有问题。但是如何添加服务器的性能计数器。这需要在服务器端进行设置，但怎样设置？还没有弄清楚。**（这个问题已经清楚了，服务器端不需要任何设置，只要客户机和服务器在一个工作组即可）**

9 经常提问的问题

1. Q：无法录制外部网站（www.yhsoft.com）的脚本，IE 的代理自动变为：localhost，我们是通过宽带局域网上网的;

   A：WAS 支持透过防火墙/代理服务器来录制脚本。

   可以做如下设置：

   - 在控制面板—管理工具—服务中，双击 Webtool 服务；
   - 选择登陆标签页，选择登陆帐户，输入用户名和密码，其中用户名输入的格式为：DOMAIN\username
   - 重新启动 WebTool 服务即可。

   按照这样的方法在每一个 WAS 客户机设置一下即可。

2. Q：对于需要登录的网站，并发数和用户数之间有没有什么关系；

   A：当并发数大于给定的 users 帐号时，WAS 会给出一个警告，要求增加 user 帐号数。最好是并发数<用户数，让每一个虚拟用户使用不同的账号登陆。当然并发数>=用户数也可以运行，只不过某些用户可能不会认证或者正确的接受 cookies。

3. Q：可不可以指定要测试网站的端口。

   A：不用过多的考虑端口问题，跟其他的（采用默认端口）录制方法一样，在 Server 文

本框中输入 Server 的名字或者 IP 地址。WAS 会自动把端口信息记录下来。

10 使用 WAS 的好处

首先，我们来讨论一下使用 WAS 测试你的应用程序的好处。

它简单

WAS 允许你以不同的方式创建测试脚本：你可以通过使用浏览器走一遍站点来录制脚本，可以从服务器的日志文件导入 URL，或者从一个网络内容文件夹选择一个文件。当然，你也可以手工地输入 URL 来创建一个新的测试脚本。

不像其它的工具，你可以使用任何数量的客户端运行测试脚本，全部都有一个中央主客户端来控制。在每一个测试开始前，主客户机透明地执行以下任务：

- 与其他所有的客户机通讯
- 把测试数据分发给所有的客户端
- 在所有客户端同时初始化测试
- 从所有的客户端收集测试结果和报告

这个特性非常重要，尤其对于要测试一个需要使用很多客户端的服务器群的最大吞吐量时非常有用。

它的高可用性

WAS 是被设计用于模拟 Web 浏览器发送请求到任何采用了 HTTP1.0 或 1.1 标准的服务器，而不考虑服务器运行的平台。

除了它的易用性外，WAS 还有很多其它的有用的特性，包括：

- 对于需要署名登录的网站，它允许创建用户帐号。
- 允许为每个用户存储 cookies 和 Active Server Pages (ASP) 的 session 信息
- 支持随机的或顺序的数据集，以用在特定的名字-值对
- 支持带宽调节和随机延迟（“ 思考的时间 ”）以更真实地模拟显示情形。
- 支持 Secure Sockets Layer (SSL)协议
- 允许 URL 分组和对每组的点击率的说明
- 提供一个对象模型，可以通过 Microsoft Visual Basic® Scripting Edition (VBScript)处理或者通过定制编程来达到开启，结束和配置测试脚本的效果。

WAS 的缺陷

除了优势外，WAS 的确有一些缺陷存在。当前知道的 bug 和有关事项都列在 WAS 的网站上了。以下是当前 WAS 不支持的特性：

- 以前面所发请求返回的结果为基础，修改 URL 参数的能力。
- 运行或模仿客户端逻辑的能力
- 为所分配的测试指定一个确定数量的测试周期的能力。
- 对拥有不同 IP 地址或域名的多个服务器的同时测试能力

注意　你可以使用多个主客户端来同时测试多个服务器。然而，如果你想把所有测试结果联系起来成为一个整体，则需要整理从各个 WAS 数据库得到的数据

- 支持页面在不同 IP 地址或域名间的重定向的能力
- 从 Web 浏览器直接记录 SSL 页面的能力

注意　WSA 已经支持 SSL 页面的测试，但是没有记录它们。你需要在脚本录制完后，手工地为每个设计好的 URL 打开 SSL 支持

虽然对这些限制有一些相应的解决办法，但是如果你的应用依赖一个或多个这样的功能的话，你也许不能完全享受 WAS 带来的好处。

11 性能优化

以下文字摘自网络上的文章。仅供参考。

随着 Internet 应用的日益广泛，用户的要求和期望也在不断地发展。今天的客户期待个性化的可定制的方案，期待这些方案不仅简单，而且快速、可靠、成本低廉。对于能够适应用户需求不断变动的可定制页面来说，静态 HTML 已经退出了舞台，比如内容根据客户请求变化的页面就是其中一例。这一切都要求系统保存相关的数据，例如有关用户本身以及用户可能请求哪些信息的数据。

紧跟这些趋势的 Web 开发者已经开始提供可定制的 Web 网站。象搜索数据之类的任务现在可以由服务器执行而无需客户干预。然而，这些变革也导致了一个结果，这就是许多网站都在使用大量的未经优化的数据库调用，从而使得应用性能大打折扣。

我们可以使用以下几种方法来解决这些问题：

1. 优化 ASP 代码。
2. 优化数据库调用。
3. 使用存储过程。
4. 调整服务器性能。

优秀的网站设计都会关注这些问题。然而，与静态页面的速度相比，任何数据库调用都会显著地影响 Web 网站的响应速度，这主要是因为在发送页面之前必须单独地为每个访问网站的用户进行数据库调用。

这里提出的性能优化方案正是基于以下事实：访问静态 HTML 页面要比访问那些内容依赖于数据库调用的页面要快。它的基本思想是：在用户访问页面之前，预先从数据库提取信息写入存储在服务器上的静态 HTML 页面。为了保证这些静态页面能够及时地反映不断变化的数据库数据，必须有一个调度程序管理静态页面的生成。

当然，这种方案并不能够适应所有的情形。例如，如果是从持续变化的大容量数据库提取少量信息，这种方案是不合适的。不过可以适用该方案的场合还是很多。

为了保证能够在合适的时间更新静态 HTML 页面，把下面的代码加入到相应的 ASP 页面前面：

```
< %
lastUpdated=Application("LastUpdated")
presentTime=now

if DATEDIFF("h",lastUpdated,presentTime)  >= 1 then
   Application ("LastUpdated") =presentTime
   response.redirect
   "Update.asp?physicalpath="&Request.ServerVariables("PATH_TRANSLATED")
end if
% >
< html >
Static content goes here
< /html >
```

每当该页面被调用，脚本就会提取最后的更新时间并将它与当前时间比较。如果两个时间之间的差值大于预定的数值，Update.asp 脚本就会运行；否则，该 ASP 页面把余下的 HTML 代码发送给浏览器。

最后更新时间从 Application 变量得到，它的第一次初始化由 global.asa 完成。具体的更新时间间隔应根据页面内容的更新要求调整。

如果每次访问 ASP 页面的时候都要提供最新的信息，或者输出与用户输入密切相关，这种方法并不实用，但这种方法可以适应以固定的时间间隔更新信息的场合。

如果数据库内容由客户通过适当的 ASP 页面更新，要确保静态页面也能够自动反映数据的变化，我们可以在 ASP 页面中调用 Update 脚本。这样，每当数据库内容改变时服务器上也有了最新的静态 HTML 页面。

另一种处理频繁变动数据的办法是借助 Microsft SQL Server 7.0 的 Web 助手向导（Web Assistant Wizard），这个向导能够利用 Transact-SQL、存储过程等从 SQL Server 数据生成标准的 HTML 文件。

利用 SQL Server 任务，Web 助手向导能够用来定期地生成 HTML 页面。正如前面概要介绍的方案， Web 助手可以通过触发子更新 HTML 页面，比如在指定的时间执行更新或者在数据库数据变化时执行更新。

SQL Server 使用名为 sp_makewebtask 的存储过程创建 HTML 页面，它的参数是目标 HTML 文件的名字和待执行存储过程的名字，查询的输出发送到 HTML 页面。另外，也可以选择使用可供结果数据插入的模板文件。

从前面的代码可以看出，当 ASP 页面 HtmlMain.asp 需要更新时，控制以 ASP 文件的物理路径为参数转到了 Update 页面。Update 脚本的任务是用新的 HTML 数据刷新发出调用的 ASP 文件，并把调度 ASP 代码加入到文件的开头。为此，Update 脚本打开调度模板文件，拷贝调度 ASP 代码，然后控制转到了另一部分脚本，这部分脚本主要任务是执行数据库操作。Update 用路径参数以写模式打开 HtmlMain.asp 文件，数据库操作的输出以 HTML 格式写入这个文件。

万一用户访问页面的时候正好在执行更新，我们可以利用锁或者其他类似的机制把页面延迟几秒钟。

HtmlMain.asp（纯 HTML 加调度 ASP 代码）和 main.asp（普通的 ASP 文件）在 WAS 下进行了性能测试。main.asp 文件要查找 5 个不同的表为页面提取数据。为了和这两个文件相比较，一个只访问单个表的 ASP 页面（SingleTableTest.asp）和一个纯 HTML 文件（PlainHtml.html）也进行了测试。　　测试结果如下表所示：

文件名字	命中数	平均 TTFB（ms）	平均 TTLB（ms）
PlainHtml.html	8	47	474
SingleTableTest.asp	8	68.88	789.38
Main.asp	9	125.89	3759.56
HtmlMain.asp	9	149.89	1739.89

其中 TTFB 是指 Total Time to First Byte，TTLB 是指 Total Time to Last Byte。

这些测试在一台 Windows NT Workstation 4.0 SP6 运行 Personal Web Server 的机器上实施。为了使性能指标更明显，带宽限制到了 14.4 K。在实际环境中数值变化可能很大，但这个结果精确地反映了各个页面在性能上的差异。

测试结果显示访问单个表的 ASP 页面的处理时间是 720.5ms，而纯 HTML 文件则为 427ms。Main.asp 和 HtmlMain.asp 的输出时间相同，但它们的处理时间分别为 3633.67ms 和 1590ms。也就是说，在这个测试环境下我们可以把处理速度提高 43%。

如果我们要让页面每隔一定的访问次数更新，比如 100 次，那么这第 100 个用户就必须等待新的 HTML 页面生成。不过，这个代价或许不算太高，其他 99 个用户获得了好处。

静态页面方法并不能够适合所有类型的页面。例如，某些页面在进行任何处理之前必须要有用户输入。但是，这种方法可以成功地应用到那些不依赖用户输入却进行大量数据库调用的页面，而且这种情况下它将发挥出更大的效率。

在大多数情况下，动态页面的生成将在相当大的程度上提高网站的性能而且无需在功能上有所折衷。虽然有许多大的网站采用了这个策略来改善性能，也有许多网站完全由于进行大量没有必要的数据库调用而表现出很差的性能。

12 参考资料

Load Testing Web Applications using Microsoft's Web Application Stress Tool

WAS 服务器负载测试软件导读

测试时代 www.testage.net