**一、总结前一天的学习**

在前两天的学习中我们知道、了解并掌握了Web Server结合App Server实现单向Https的这样的一个架构。这个架构是一个非常基础的J2ee工程上线布署时的一种架构。在前两天的教程中，还讲述了Http服务 器、App Server的最基本安全配置（包括单向https的实现）， 它只是避免了用户可以通过浏览器侵入我们的Web访问器或者能够通过Web浏览器来查询我们的Web目录结构及其目录内的文件与相关内容，这种入侵我们把 它称为：

Directory traversal，当然我们只是实现了最基本的防范Directory traversal的手段，在日后的Security课程中将会详细地去擅述完整的Web Security的相关理论。

从今天起我们将继续在原有的这种Apache+Tomat的架构上，去论述如何在性能及Performance上优化这个架构，因此这两天的课程在有些人看来，可能会有些“枯燥”，所以我在此给大家打个招呼：

这两天的课程论述的是如何在不改动代码与SQL语句的前提下，如何去改善和提高web server与app server的性能，千万不要小觑这一内容，它可以让你在不改动代码的情况下得到10-20倍以上的性能提高，网上有其它的大牛们写过一篇文章叫 “Tomcat如何支持到1000个用户”，经本人经过几个重大工程的实践，Opensource的Tomcat如果调优的好不只可以支持者1000个用 户，尤其当你的布署环境是64位操作系统的情况下，可能能够支持更大更高的并发性能，最后本节内容将会以Tomcat集群来做收场，在将来的课程中还会进 一步详细讲述Weblogic的集群配置与IBM WASND的集群配置。

**二、从性能测试谈起**

**2.1 性能测试简介**

即压力测试，就是根据一定数量的VU（Virtual Users）我称为并发用户操作核心交易后，系统所能达到的最大瓶劲，以便于发现系统的极限、有没有Outof memory这样的问题存在以及相关的系统设置、配置是否搭挡的合理的一种测试。

一般商业的比较好的用LoaderRunner，如果没钱的就用Opensource的Jmeter来模拟这个VU的操作。

压力测试，存在几个误区，需要小心。

1）  无限大的拼命增加VU的数量

系统再完美，硬件配置再高，也经不住没有经过合理运算的VU的压力呀。

2）  偏执的用一定的数据量的VU，跑7\*24小时

不是说这个没必要，很有必要，小日本的电视为什么寿命敢说比中国人生产的电视机寿命长？因为它用一个机械臂就对着电视机的按钮不断的点点点。

我们说的压力测试要测试多长时间，关键是要看经过科学计算的VU的数量以及核心交易数有多少，不是说我拿250个VU跑24\*7如果没有问题我这个系统就没有问题了，这样的说法是不对的，错误的。随便举个例子就能把你推倒。

假设我有250个VU，同时跑上万笔交易，每个VU都有上万笔交易，250个VU一次跑下来可能就要数个小时，你又怎么能断定250个VU对于这样 的系统我跑24\*7小时就能真的达到上万笔交易在250个VU的并发操作下能够真的跑完7天的全部交易？可能需要一周半或者两周呢？对吧？

我还看到过有人拿500个VU对着一条交易反复跑24\*7小时。。。这样的测试有意义吗？你系统就仅仅只有一条交易？你怎么能够判断这条交易涉及到 的数据量最大？更不用说交易是彼此间有依赖的，可能a+b+c+d的交易的一个混合组织就能够超出你单笔交易所涉及到的数据量了呢！

**2.2 合理的制定系统最大用户、并发用户**

提供下面这个公式，以供大家在平时或者日常需要进行的性能测试中作为一个参考。

（1）        计算平均的并发用户数：C = nL/T

公式（1）中，C是平均的并发用户数；n是login session的数量；L是login session的平均长度；T指考察的时间段长度。

（2）        并发用户数峰值：C’ ≈ C+3根号C

公式（2）则给出了并发用户数峰值的计算方式中，其中，C’指并发用户数的峰值，C就是公式（1）中得到的平均的并发用户数。该公式的得出是假设用户的loginsession产生符合泊松分布而估算得到的。

**实例：**

假设有一个OA系统，该系统有3000个用户，平均每天大约有400个用户要访问该系统，对一个典型用户来说，一天之内用户从登录到退出该系统的平均时间为4小时，在一天的时间内，用户只在8小时内使用该系统。

则根据公式（1）和公式（2），可以得到：

C = 400\*4/8 = 200

C’≈200+3\*根号200 = 242

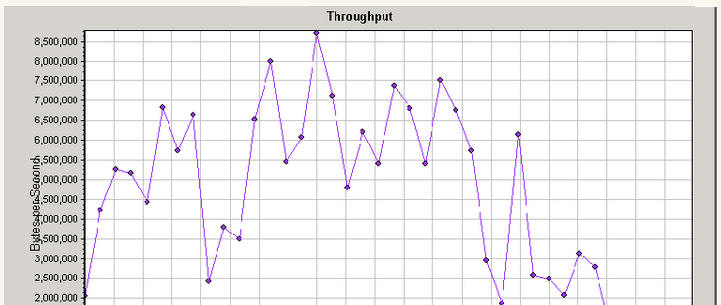
F=VU \* R / T

其中F为吞吐量，VU表示虚拟用户个数，R表示每个虚拟用户发出的请求数，T表示性能测试所用的时间

R = T / TS。

**2.3 影响和评估性能的几个关键指标**

从上面的公式一节中我们还得到了一个名词“吐吞量”。和吞吐量相关的有下面这些概念，记录下来以供参考。



²  **吞吐量**

指在一次性能测试过程中网络上传输的数据量的总和。

对于交互式应用来说，吞吐量指标反映的是服务器承受的压力，在容量规划的测试中，吞吐量是一个重点关注的指标，因为它能够说明系统级别的负载能力，另外，在性能调优过程中，吞吐量指标也有重要的价值。

²  **吞吐率**

单位时间内网络上传输的数据量，也可以指单位时间内处理客户请求数量。它是衡量网络性能的重要指标，通常情况下，吞吐率用“字节数/秒”来衡量，当 然，你可以用“请求数/秒”和“页面数/秒”来衡量。其实，不管是一个请求还是一个页面，它的本质都是在网络上传输的数据，那么来表示数据的单位就是字节 数。

²  **事务**

就是用户某一步或几步操作的集合。不过，我们要保证它有一个完整意义。比如用户对某一个页面的一次请求，用户对某系统的一次登录，淘宝用户对商品的一次确认支付过程。这些我们都可以看作一个事务。那么如何衡量服务器对事务的处理能力。又引出一个概念----TPS

²  **TPS (Transaction Per second)**

每秒钟系统能够处理事务或交易的数量，它是衡量系统处理能力的重要指标。

²  **点击率（Hit Per Second）**

点击率可以看做是TPS的一种特定情况。点击率更能体现用户端对服务器的压力。TPS更能体现服务器对客户请求的处理能力。

每秒钟用户向web服务器提交的HTTP请求数。这个指标是web 应用特有的一个指标；web应用是“请求-响应”模式，用户发一个申请，服务器就要处理一次，所以点击是web应用能够处理的交易的最小单位。如果把每次 点击定义为一个交易，点击率和TPS就是一个概念。容易看出，点击率越大。对服务器的压力也越大，点击率只是一个性能参考指标，重要的是分析点击时产生的 影响。

需要注意的是，这里的点击不是指鼠标的一次“单击”操作，因为一次“单击”操作中，客户端可能向服务器发现多个HTTP请求。

²  **吞吐量指标的作用：**

ü   用户协助设计性能测试场景，以及衡量性能测试场景是否达到了预期的设计目标：在设计性能测试场景时，吞吐量可被用户协助设计性能测试场景，根据估算的吞吐 量数据，可以对应到测试场景的事务发生频率，事务发生次数等；另外，在测试完成后，根据实际的吞吐量可以衡量测试是否达到了预期的目标。

ü   用于协助分析性能瓶颈：吞吐量的限制是性能瓶颈的一种重要表现形式，因此，有针对性地对吞吐量设计测试，可以协助尽快定位到性能冰晶所在位置。

²  **平均相应时间**

也称为系统响应时间，它一般指在指定数量的VU情况下，每笔交易从mouse 的click到IE的数据刷新与展示之间的间隔，比如说：250个VU下每笔交易的响应时间不超过2秒。

当然，响应时间也不能一概而论，对于实时交易如果银行柜台操作、超市收银员（邪恶的笑。。。）的操作、证交所交易员的操作来说这些操作的响应时间当然是越快越好，而对于一些企业级的如：

与银行T+1交易间的数据跑批、延时交易、T+1报表等，你要求它在2秒内响应，它也做不到啊。就好比你有个1MB的带宽，你传的东西是超过 4MB，你要它在2秒内跑完理论速度也做不到啊，对吧，所以有些报表或者数据，光前面传输时间就不止两秒了。。。一口咬死说我所有的交易平均相应时间要2 秒，真的是不科学的！

**2.4 合理的性能测试**

²  **VU数量的增加**

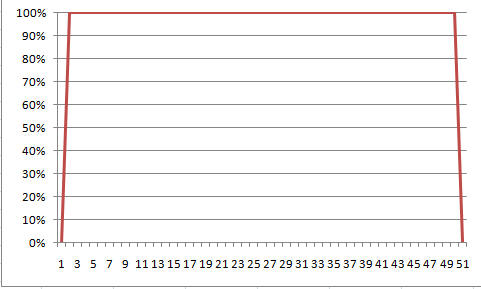
一个合理的性能测试除了需要合理的计算VU的数量、合理的设置系统平均响应时间外还需要合理的在测试时去规划发起VU的时间，比如说，我看到有人喜欢这样做压力测试。

第一秒时间，500个并发用户全部发起了。。。结果导致系统没多久就崩了，然后说系统没有满足设计要求。

为什么说上述这样的做法是不对的？我们说不是完全不对，只能说这样的测试已经超过了500个VU的并发的设计指标了。

合理的并发应该是如下这样的：

有25-50个VU开始起交易了，然后过一段时间又有25-50个用户，过一段时间又增加一些VU，当所有的设计VU都发起交易了，此时，再让压力 测试跑一段时间比如说：24\*7是比较合理的。所以VU数量不是一上手就500个在一秒内发起的，VU数量的增加应该如下面这张趋势图：



这是一个阶梯状的梯型图，可以看到VU的发起是逐渐逐渐增多的，以下两种情况如果发生需要检查你的系统是否在原有设置上存在问题：

ü   VU数量上升阶段时崩溃

有时仅仅在VU数量上升阶段，系统就会了现各种各样的错误，甚至有崩溃者，这时就有重新考虑你的系统是否有设置不合理的地方了。

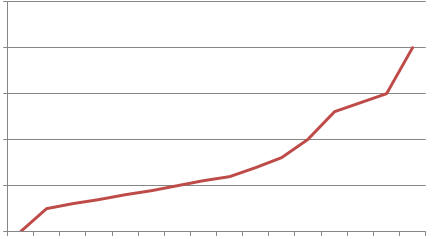
ü   VU全部发起后没多久系统崩溃

VU在达到最高值时即所有的VU都已经发起了，此时它是以一条直的水平线随着系统运行而向前延伸着的，但过不了多久，比如说：运行24\*7小时，运行了没一、两天，系统崩溃了，也需要做检查。

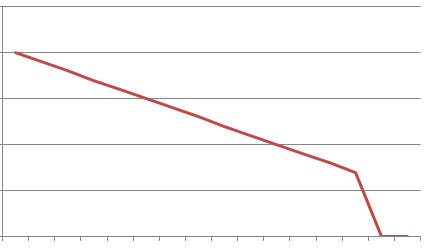
所以，理想的性能测试应该是VU数量上升到最终VU从发起开始到最后所有VU把交易做完后，VU数量落回零为止。

²  **吐吞量的变化**

从2.3节我们可以知道，吞吐量是随着压力/性能测试的时间而逐渐增大的，因此你的吞吐量指示应该如下图所示：



肯定是这样，你的吞吐量因该是积累的，如果你的吞吐量在上升了一段时间后突然下落，而此时你的性能测试还在跑着，如下图所示：



那么，此时代表什么事情发生了？你可以查一下你的loaderrunner或者jmeter里对于这段吞吐量回落期间的交易的response的状 态进行查看，你将会发现大量的error已经产生，因为产生了error，所以你的交易其实已经出错了，因此每次运行的数据量越来越小，**这也就意味着你的压力测试没有过关，系统被你压崩了！**

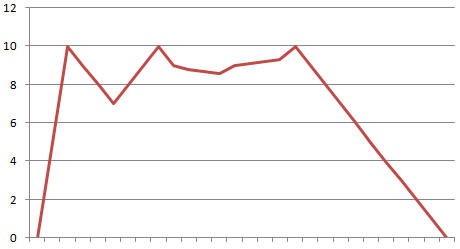
²  **平均响应时间**

平均响应时间如VU的数量增加趋势图一样，一定是一开始响应时间最短，然后一点点增高，当增高到一定的程度后，即所有的VU都发起交易时，你的响应 时间应该维持在一个水平值，然后随着VU将交易都一笔笔做完后，这个响应时间就会落下来，这段时间内的平均值就是你的系统平均响应时间。看看它，有没有符 合设计标准？

²  **内存监控**

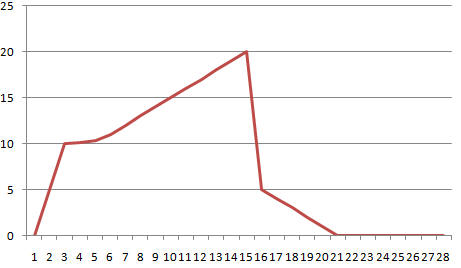
我们就来说AppServer，我们这边用的是Tomcat即SUN的JVM的内存变化，我们就用两张图例来讲解吧：

理想状态情况下的JVM内存使用趋势：



这是一个波浪型的（或者也可以说是锯齿型的）趋势图，随着VU数量的一点点增加，我们的内存使用数会不断的增加，但是JVM的垃圾回收是自动回收机 制的，因此如果你的JVM如上述样的趋势，内存上涨一段时间，随即会一点点下落，然后再上涨一点，涨到快到头了又开始下落，直到最后你的VU数量全部下降 下来时，你的JVM的内存使用也会一点点的下降。

非理想状态情况下的JVM内存使用趋势：



嘿嘿嘿，看到了吗？你的JVM随着VU 数量的上升，而直线上升，然后到了一定的点后，即到了java –Xmx后的那个值后，突然直线回落，而此时你的交易还在进行，压力测试也还在进行，可是内存突然回落了。。。因为你的JVM已经crash了，即OUT OF MEMORY鸟。

²  **CPU Load**

我们来看一份测试人员提交上来CPU得用率的报告：

|  |  |  |
| --- | --- | --- |
| **Web Server** | **App Server** | **DB Server** |
| **60%** | 98%  =\_=!(oh my god) | 6%  囧 |

**同时平均响应时间好慢啊。**

拿过来看了一下代码与设计。。。Struts+Spring+JDBC的一个框架，没啥花头的，再仔细一看Service层。

大量的复杂业务逻辑甚至报表的产生全部用的是javaobject如:List, Hashmap等操作，甚至还有在Service层进行排序、复杂查询等操作。

一看DB层的CPU利用率才6%，将一部分最复杂的业务拿出去做成Store Procedure（存储过程后），再重新运行压力测试。

|  |  |  |
| --- | --- | --- |
| **Web Server** | **App Server** | **DB Server** |
| **60%** | 57%  =\_=!(oh my god) | 26%  囧 |

**同时平均响应时间比原来快了15-16倍。**

**为什么？？**

看看第一份报告，我们当时还查看了数据库服务器的配置，和APPServer的配置是一个级别的，而利用率才6%。。。

数据库，至所以是大型的商用的关系型数据库，你只拿它做一个存储介质，你这不是浪费吗？人家里面设置的这个StoreProcedure的处理能 力，索引效率，数据分块等功能都没有去利用，而用你的代码去实现那么多复杂业务比如说多表关联、嵌套等操作，用必要吗？那要数据库干什么用呢？

是啊，我承认，原有这样的代码，跨平台能力强一点，可付出的代价是什么呢？

用户在乎你所谓的跨平台的理论还是在乎的是你系统的效率？一个系统定好了用DB2或者是SQL SERVER，你觉得过一年它会换成ORACLE或者MYSQL吗？如果1年一换，那你做的系统也只能让用户勉强使用一年，我劝你还是不要去做了。在中 国，有人统计过5年左右会有一次系统的更换，而一些银行、保险、金融行业的系统一旦采用了哪个数据库，除非这个系统彻底出了问题，负责是不会轻意换数据库 的，因此不要拿所谓的纯JAVA代码或者说我用的是Hibernate,ejb实现可以跨数据库这套来说事，效率低下的系统可以否定你所做的一切，一切！

**三、Apache服务器的优化**

上面两节，讲了大量的理论与实际工作中碰到的相关案例，现在就来讲一下在我们第一天和第二天中的ApacheHttp Server + Tomcat这样的架构，怎么来做优化吧。

**3.1 Linux/UnixLinux系统下Apache 并发数的优化**

Apache Http Server在刚安装完后是没有并发数的控制的，它采用一个默认的值，那么**我们的Web Server硬件很好，允许我们撑到1000个并发即VU**，**而因为我们没有去配置导致我们的WebServer连300个并发都撑不到**，你们认为，这是谁的责任？

Apache Http服务器采用prefork或者是worker两种并发控制模式。

²  preforkMPM

使用多个子进程，每个子进程只有一个线程。每个进程在某个确定的时间只能维持一个连接。在大多数平台上，PreforkMPM在效率上要比 Worker MPM要高，但是内存使用大得多。prefork的无线程设计在某些情况下将比worker更有优势：它可以使用那些没有处理好线程安全的第三方模块，并 且对于那些线程调试困难的平台而言，它也更容易调试一些。

²  workerMPM 使用多个子进程，每个子进程有多个线程。每个线程在某个确定的时间只能维持一个连接。通常来说，在一个高流量的HTTP服务器上，Worker MPM是个比较好的选择，因为Worker MPM的内存使用比PreforkMPM要低得多。但worker MPM也由不完善的地方，如果一个线程崩溃，整个进程就会连同其所有线程一起"死掉".由于线程共享内存空间，所以一个程序在运行时必须被系统识别为"每 个线程都是安全的"。

一般来说我们的ApacheHttp Server都是装在Unix/Linux下的，而且是采用源码编译的方式来安装的，我们能够指定在编译时Apache就采用哪种模式，为了明确我们目前 的Apache采用的是哪种模式在工作，我们还可以使用httpd –l命令即在Apache的bin目录下执行httpd –l，来确认我们使用的是哪种模式。

这边，我们使用Apache配置语言中的” IfModule”来自动选择模式的配置。

我们的ApacheHttp Server在配完后一般是没有这样的配置的，是需要你手动的添加如下这样的一块内容的，我们来看，在httpd.conf文件中定位到最后一行LoadModule，敲入回车，加入如下内容：

|  |
| --- |
| <IfModule prefork.c>  ServerLimit  20000  StartServers  5  MinSpareServers  5  MaxSpareServers  10  MaxClients  1000  MaxRequestsPerChild 0  </IfModule> |

**上述参数解释：**

ü   **ServerLimit     20000**

默认的MaxClient最大是256个线程,如果想设置更大的值，就的加上ServerLimit这个参数。20000是ServerLimit这个参数的最大值。如果需要更大，则必须编译apache,此前都是不需要重新编译Apache。

生效前提：必须放在其他指令的前面

ü   StartServers  5

指定服务器启动时建立的子进程数量，prefork默认为5。

ü   MinSpareServers  5

指定空闲子进程的最小数量，默认为5。如果当前空闲子进程数少于MinSpareServers ，那么Apache将以最大每秒一个的速度产生新的子进程。此参数不要设的太大。

ü   MaxSpareServers  10

设置空闲子进程的最大数量，默认为10。如果当前有超过MaxSpareServers数量的空闲子进程，那么父进程将杀死多余的子进程。此参数不 要设的太大。如果你将该指令的值设置为比MinSpareServers小，Apache将会自动将其修改成"MinSpareServers+1"。

ü   MaxClients  256

限定同一时间客户端最大接入请求的数量(单个进程并发线程数)，默认为256。任何超过MaxClients限制的请求都将进入等候队列,一旦一个链接被释放，队列中的请求将得到服务。要增大这个值，你必须同时增大ServerLimit。

ü   MaxRequestsPerChild10000

每个子进程在其生存期内允许伺服的最大请求数量，默认为10000.到达MaxRequestsPerChild的限制后，子进程将会结束。如果MaxRequestsPerChild为"0"，子进程将永远不会结束。

将MaxRequestsPerChild设置成非零值有两个好处：

1.可以防止(偶然的)内存泄漏无限进行，从而耗尽内存。

2.给进程一个有限寿命，从而有助于当服务器负载减轻的时候减少活动进程的数量。

**Prefork.c的工作方式：**

一个单独的控制进程(父进程)负责产生子进程，这些子进程用于监听请求并作出应答。Apache总是试图保持一些备用的(spare)或者是空闲的 子进程用于迎接即将到来的请求。这样客户端就不需要在得到服务前等候子进程的产生。在Unix系统中，父进程通常以root身份运行以便邦定80端口，而 Apache产生的子进程通常以一个低特权的用户运行。User和Group指令用于设置子进程的低特权用户。运行子进程的用户必须要对它所服务的内容有 读取的权限，但是对服务内容之外的其他资源必须拥有尽可能少的权限。

在上述的</IfModule>后再加入一个”<IfModule>”如下红色加粗（**大又粗**）内容：

|  |
| --- |
| <IfModule prefork.c>  ServerLimit  20000  StartServers  5  MinSpareServers  5  MaxSpareServers  10  MaxClients  1000  MaxRequestsPerChild 0  </IfModule>  **<IfModule worker.c>**  **ServerLimit  50**  **ThreadLimit  200**  **StartServers  5**  **MaxClients  5000**  **MinSpareThreads  25**  **MaxSpareThreads  500**  **ThreadsPerChild  100**  **MaxRequestsPerChild 0**  **</IfModule>** |

**上述参数解释：**

ü   ServerLimit16

服务器允许配置的进程数上限。这个指令和ThreadLimit结合使用设置了MaxClients最大允许配置的数值。任何在重启期间对这个指令的改变都将被忽略，但对MaxClients的修改却会生效。

ü   ThreadLimit64

每个子进程可配置的线程数上限。这个指令设置了每个子进程可配置的线程数ThreadsPerChild上限。任何在重启期间对这个指令的改变都将被忽略，但对ThreadsPerChild的修改却会生效。默认值是"64".

ü   StartServers3

服务器启动时建立的子进程数，默认值是"3"。

ü   MinSpareThreads75

最小空闲线程数,默认值是"75"。这个MPM将基于整个服务器监视空闲线程数。如果服务器中总的空闲线程数太少，子进程将产生新的空闲线程。

ü   MaxSpareThreads250

设置最大空闲线程数。默认值是"250"。这个MPM将基于整个服务器监视空闲线程数。如果服务器中总的空闲线程数太多，子进程将杀死多余的空闲线 程。MaxSpareThreads的取值范围是有限制的。Apache将按照如下限制自动修正你设置的值：worker要求其大于等于 MinSpareThreads加上ThreadsPerChild的和

ü   MaxClients400

允许同时伺服的最大接入请求数量(最大线程数量)。任何超过MaxClients限制的请求都将进入等候队列。默认值 是"400",16(ServerLimit)乘以25(ThreadsPerChild)的结果。因此要增加MaxClients的时候，你必须同时增 加ServerLimit的值。

ü   ThreadsPerChild25

每个子进程建立的常驻的执行线程数。默认值是25。子进程在启动时建立这些线程后就不再建立新的线程了。

ü   MaxRequestsPerChild  0

设置每个子进程在其生存期内允许伺服的最大请求数量。到达MaxRequestsPerChild的限制后，子进程将会结束。如果MaxRequestsPerChild为"0"，子进程将永远不会结束。

将MaxRequestsPerChild设置成非零值有两个好处：

1.可以防止(偶然的)内存泄漏无限进行，从而耗尽内存。

2.给进程一个有限寿命，从而有助于当服务器负载减轻的时候减少活动进程的数量。

**注意**

对于KeepAlive链接，只有第一个请求会被计数。事实上，它改变了每个子进程限制最大链接数量的行为。

**Worker.c的工作方式：**

每个进程可以拥有的线程数量是固定的。服务器会根据负载情况增加或减少进程数量。一个单独的控制进程(父进程)负责子进程的建立。每个子进程可以建 立ThreadsPerChild数量的服务线程和一个监听线程，该监听线程监听接入请求并将其传递给服务线程处理和应答。Apache总是试图维持一个 备用(spare)或是空闲的服务线程池。这样，客户端无须等待新线程或新进程的建立即可得到处理。在Unix中，为了能够绑定80端口，父进程一般都是 以root身份启动，随后，Apache以较低权限的用户建立子进程和线程。User和Group指令用于设置Apache子进程的权限。虽然子进程必须 对其提供的内容拥有读权限，但应该尽可能给予它较少的特权。另外，除非使用了suexec，否则，这些指令设置的权限将被CGI脚本所继承。

**公式：**

ThreadLimit>= ThreadsPerChild

MaxClients  <= ServerLimit \* ThreadsPerChild  必须是ThreadsPerChild的倍数

MaxSpareThreads>= MinSpareThreads+ThreadsPerChild

**硬限制：**

ServerLimi和ThreadLimit这两个指令决定了活动子进程数量和每个子进程中线程数量的硬限制。要想改变这个硬限制必须完全停止服务器然后再启动服务器(直接重启是不行的)。

Apache在编译ServerLimit时内部有一个硬性的限制，你不能超越这个限制。

preforkMPM最大为"ServerLimit200000"

其它MPM(包括work MPM)最大为"ServerLimit 20000

Apache在编译ThreadLimit时内部有一个硬性的限制，你不能超越这个限制。

mpm\_winnt是"ThreadLimit 15000"

其它MPM(包括work prefork)为"ThreadLimit 20000

**注意**

使用ServerLimit和ThreadLimit时要特别当心。如果将ServerLimit和ThreadLimit设置成一个高出实际需要许多的值，将会有过多的共享内存被分配。当设置成超过系统的处理能力，Apache可能无法启动，或者系统将变得不稳定。

**3.2 WindowsWindows系统下Apache 并发数的优化**

以上是Linux/Unix下的Apache的并发数优化配置，如果我们打入了httpd –l如下显示：



怎么办？

ü   步骤一

先修改/path/apache/conf/httpd.conf文件。

httpd.conf

将“#Includeconf/extra/httpd-mpm.conf”前面的 “#” 去掉，保存。

ü   步骤二

再修改/apache安装目录/conf/extra/httpd-mpm.conf文件。

**在mpm\_winnt模式下，Apache不使用prefork也不使用work工作模式，切记！**

因此，我们只要找到原文件中：

|  |
| --- |
| <IfModule mpm\_winnt\_module>      ThreadsPerChild      150      MaxRequestsPerChild    0  </IfModule> |

修改后

|  |
| --- |
| <IfModule mpm\_winnt\_module>      ThreadsPerChild      500      MaxRequestsPerChild    5000  </IfModule> |

**上述参数解释：**

ü   ThreadsPerChild

是指一个进程最多拥有的线程数（Windows版本，貌似不可以开启多个进程),一般100-500就可以，根据服务器的具体性能来决定。

ü   MaxRequestsPerChild

是指一个线程最多可以接受的连接数，默认是0，就是不限制的意思，

0极有可能会导致内存泄露。所以，可以根据实际情况，配置一个比较大的值。Apache会在几个线程之间进行轮询，找到负载最轻的一个线程来接受新的连接。

**注意：**

修改后，一定不要apacherestart，而是先 apache stop 然后再 apache start才可以。

**3.3 启用服务端图片压缩**

对于静态的html 文件，在apache 可加载mod\_deflate.so 模块，把内容压缩后输出，可节约大量的传输带宽。

打开httpd.conf文件，找到：

|  |
| --- |
| #LoadModule deflate\_module modules/mod\_deflate.so |

将前面的“#”去掉，变成：

|  |
| --- |
| LoadModule deflate\_module modules/mod\_deflate.so |

然后在最后一行的LoadModule处，加入如下的几行：

|  |
| --- |
| <IfModule mod\_deflate.c>   DeflateCompressionLevel 7   AddOutputFilterByType DEFLATE text/html text/plain text/xml application/x-httpd-php   AddOutputFilter DEFLATE css js  </IfModule> |

**注意：**

默认等级是6，而且9级需要更多的CPU时间，用默认的6级就可以了。

要注意的是,在apache 2.2.15中,我用httpd -l看,居然发现mod\_deflat已经内置了，所以其实就不用再在httpd.conf中增加loadmodule了,否则会说出错的

**3.4 Apache中将MS办公文档自动关联客户端的MS-Office**

我们经常会在web页的一个超链接上点一个指向物理文件的文档，我们一般会得到“保存，另存为，打开”，3个选项，当我们打开的如果是一个MS文 档，在选“打开”选项时IE会自动启用客户端上装有的word或者是excel等相关MS办公工具去打开，这个怎么做呢？很简单。

打开httpd.conf，找到：

|  |
| --- |
| AddType application/x-compress .Z      AddType application/x-gzip .gz .tgz |

在其后敲入一个回车，加入：

|  |
| --- |
| AddType application/vnd.openxmlformats  docx pptx xlsx doc xls ppt txt |

重启Apache服务即可。

**3.5 防止DDOS攻击**

DDOS攻击即采用自动点击机器人或者连续点击工具不断的刷新某一个网址或者网页上的按钮，造成网站在一时间收到大量的HTTP请求，进而阻塞网站正常的HTTP通道甚至造成网站瘫痪。



为了防止这一形式的攻击，我们一般把在一个按钮或者是一个请求在一秒内连续执行如：100次，可以认为是一种攻击（比如说你打开一个网页，点一下提交按钮，然后按住F5键不松开）。

在Linux下的Apache HttpServer安装后会提供一个mod\_evasive20的模块，用于防止这一形式的攻击，它的做法是：

如果认为是一个DDOS攻击，它的防范手段采用如下两种形势：

ü   把这个请求相关联的IP，封锁30分钟

ü   直接把相关的IP踢入黑名单，让其永不翻身

**设置：**

在你的Apache的httpd.conf文件中的最后一行“LoadModule”加入如下这句：

|  |
| --- |
| LoadModule evasive20\_module   /usr/lib/httpd/modules/mod\_evasive20.so |

然后加入下面这几行

|  |
| --- |
| <IfModule mod\_evasive20.c>  DOSHashTableSize 3097  DOSPageCount 15  DOSSiteCount 100  DOSPageInterval 1  DOSSiteInterval 1  DOSBlockingPeriod 36000  DOSEmailNotify 网站超级管理员@xxx.com  DOSLogDir "logs/mod\_evasive"  </IfModule> |

**核心参数解释：**

ü   DOSHashTableSize3097 记录黑名单的尺寸

ü   DOSPageCount 每个页面被判断为dos攻击的读取次数

ü   DOSSiteCount 每个站点被判断为dos攻击的读取部件(object)的个数

ü   DOSPageInterval 读取页面间隔秒

ü   DOSSiteInterval 读取站点间隔秒

ü   DOSBlockingPeriod 被封时间间隔秒

**注意：**

上述设置是针对Linux/Unix下的Apache Server，相关的Windows下的Apache见如下设置：

**为Windows下的Apache加载mod\_evasive模块**

1. 下载附件中的压缩包，解压并拷贝mod\_dosevasive22.dll到Apache安装目录下的modules目录（当然也可以是其他目录，需要自己修改路径）。

2. 修改Apache的配置文件http.conf。

添加以下内容

|  |
| --- |
| LoadModule dosevasive22\_module modules/mod\_dosevasive22.dll  DOSHashTableSize 3097  DOSPageCount 3  DOSSiteCount 50  DOSPageInterval 1  DOSSiteInterval 1  DOSBlockingPeriod 10 |

**3.6 Apache中设置URL含中文附件的下载/打开的方法（仅限Linux系统下）**

这个话题很有趣，起因是我们在工程中碰到了客户这样的一个需求：

<a href=”xxx.xxx.xx/xx/xxx/轮胎损坏情况2007-05-05.jpg”>损坏部件</a>

看看好像没啥问题，一点这个超链接，因该是在IE中打开一个叫” 轮胎损坏情况2007-05-05.jpg”，嘿嘿，大家自己动手放一个带有中文名的这样的一个图片，看看能否被解析，解析不了。

所以我们就说，真奇怪，我们上传图片都是上传时的图片名经上传组件解析过以后变成一个UUID或者是GUID一类的文件名 如：gb19070122abcxd.jpg这样一种英文加数字组合的文件名，这样的文件名，Apache当然是可以解析的，客户坚持一定我上传的图片是 中文名（连中文描述都不行），因为，客户说：我们是中国人，当然用中文图片名。。。

没办法，找了半天，找到一篇日文的教程，还好还好，N年前学过一点点日语，照着教程把它啃下来了。

这是一个日本人写的关于在Apache中支持以亚州文字命名文件名的一个“补丁”，叫“mod\_encoding”。

**相关配置：**

1.       下载完后是一个这样的压缩包：mod\_encoding-20021209.tar.gz

2.       解压后使用：

|  |
| --- |
| configure  make  make install |

在make这一行时，编译出错，报“make: \*\*\* [mod\_encoding.so] Error 1”这样的错

原因很明显，是regex.h未包含进来，解决办法也很简单：

ü   用vi打开mod\_encoding.c，

ü   在#include <httpd.h>那一段的前面加上如下一行：

#include <regex.h>然后：

           重新make再make install 搞定，CALL！！！

3.       编译后得到一个：mod\_encoding.so的文件，然后在httpd.conf文件中加入下面这几行：

|  |
| --- |
| LoadModule encoding\_module modules/mod\_encoding.so  Header add MS-Author-Via "DAV"  <IfModule mod\_encoding.c>    EncodingEngine    on    NormalizeUsername on    SetServerEncoding GBK    DefaultClientEncoding UTF-8 GBK GB2312    AddClientEncoding "(Microsoft .\* DAV $)" UTF-8 GBK GB2312    AddClientEncoding "Microsoft .\* DAV" UTF-8 GBK GB2312    AddClientEncoding "Microsoft-WebDAV\*" UTF-8 GBK GB2312  </IfModule> |

4.       重启Apache，搞定，在apache中我们的url可以是中文名的附件了。

**3.7 不可忽视的keepalive选项**

在Apache 服务器中，KeepAlive是一个布尔值，On 代表打开，Off 代表关闭，这个指令在其他众多的 HTTPD 服务器中都是存在的。

KeepAlive 配置指令决定当处理完用户发起的 HTTP 请求后是否立即关闭 TCP 连接，如果 KeepAlive 设置为On，那么用户完成一次访问后，不会立即断开连接，如果还有请求，那么会继续在这一次 TCP 连接中完成，而不用重复建立新的 TCP 连接和关闭TCP 连接，可以提高用户访问速度。

**那么我们考虑3种情况：**

　　1.用户浏览一个网页时，除了网页本身外，还引用了多个javascript 文件，多个css 文件，多个图片文件，并且这些文件都在同一个HTTP 服务器上。

　　2.用户浏览一个网页时，除了网页本身外，还引用一个javascript 文件，一个图片文件。

　　3.用户浏览的是一个动态网页，由程序即时生成内容，并且不引用其他内容。

对于上面3中情况，我认为：1 最适合打开 KeepAlive ，2 随意，3 最适合关闭 KeepAlive

　下面我来分析一下原因。

**在 Apache 中，打开和关闭 KeepAlive 功能，服务器端会有什么异同呢？**

**先看看理论分析。**

打开KeepAlive 后，意味着每次用户完成全部访问后，都要保持一定时间后才关闭会关闭TCP 连接，那么在关闭连接之前，必然会有一个Apache进程对应于该用户而不能处理其他用户，假设KeepAlive 的超时时间为10 秒种，服务器每秒处理 50个独立用户访问，那么系统中 Apache 的总进程数就是 10 \* 50 ＝ 500 个，如果一个进程占用 4M 内存，那么总共会消耗 2G内存，所以可以看出，在这种配置中，相当消耗内存，但好处是系统只处理了 50次 TCP 的握手和关闭操作。

如果关闭KeepAlive，如果还是每秒50个用户访问，如果用户每次连续的请求数为3个，那么 Apache 的总进程数就是 50 \* 3= 150 个，如果还是每个进程占用 4M 内存，那么总的内存消耗为 600M，这种配置能节省大量内存，但是，系统处理了 150 次 TCP的握手和关闭的操作，因此又会多消耗一些 CPU 资源。

再看看实践的观察。

我在一组大量处理动态网页内容的服务器中，起初打开KeepAlive功能，经常观察到用户访问量大时Apache进程数也非常多，系统频繁使用交 换内存，系统不稳定，有时负载会出现较大波动。关闭了KeepAlive功能后，看到明显的变化是：Apache 的进程数减少了，空闲内存增加了，用于文件系统Cache的内存也增加了，CPU的开销增加了，但是服务更稳定了，系统负载也比较稳定，很少有负载大范围 波动的情况，负载有一定程度的降低；变化不明显的是：访问量较少的时候，系统平均负载没有明显变化。

**总结一下：**

在内存非常充足的服务器上，不管是否关闭KeepAlive 功能，服务器性能不会有明显变化；

如果服务器内存较少，或者服务器有非常大量的文件系统访问时，或者主要处理动态网页服务，关闭KeepAlive 后可以节省很多内存，而节省出来的内存用于文件系统Cache，可以提高文件系统访问的性能，并且系统会更加稳定。

ü   补充1

关于是否应该关闭 KeepAlive 选项，我觉得可以基于下面的一个公式来判断。

　　在理想的网络连接状况下，系统的Apache 进程数和内存使用可以用如下公式表达：

HttpdProcessNumber= KeepAliveTimeout \* TotalRequestPerSecond / Average(KeepAliveRequests)

HttpdUsedMemory= HttpdProcessNumber \* MemoryPerHttpdProcess

**换成中文意思：**

总Apache进程数 = KeepAliveTimeout \* 每秒种HTTP请求数 / 平均KeepAlive请求

Apache占用内存 = 总Apache进程数 \* 平均每进程占用内存数

**需要特别说明的是：**

[平均KeepAlive请求] 数，是指每个用户连接上服务器后，持续发出的 HTTP 请求数。当 KeepAliveTimeout 等 0或者 KeepAlive 关闭时，KeepAliveTimeout 不参与乘的运算从上面的公式看，如果 [每秒用户请求]多，[KeepAliveTimeout] 的值大，[平均KeepAlive请求] 的值小，都会造成 [Apache进程数] 多和 [内存]多，但是当 [平均KeepAlive请求] 的值越大时，[Apache进程数] 和 [内存] 都是趋向于减少的。

基于上面的公式，我们就可以推算出当 平均KeepAlive请求 <= KeepAliveTimeout 时，关闭 KeepAlive 选项是划算的，否则就可以考虑打开。

ü   补充2

KeepAlive 该参数控制Apache是否允许在一个连接中有多个请求，默认打开。但对于大多数论坛类型站点来说，通常设置为off以关闭该支持。

ü   补充3

如果服务器前跑有应用squid服务，或者其它七层设备,KeepAlive On 设定要开启持续长连接

实际在 前端有squid 的情况下,KeepAlive 很关键。记得On。

Keeyalive不能随心所欲设置，而是需要根据实际情况，我们来看一个真实的在我工作中发生的搞笑一次事件：

当时我已经离开该项目了，该项目的TeamLeader看到了keepalive的概念，他只看到了关闭keeyalive可以节省web服务器的内存，当时我们的web服务器只有4gb内存，而并发请求的量很大，因此他就把这个keepalive设成了off。

然后直接导致脱机客户端（脱机客户端用的是.net然后webservice连接）的“login”每次都显示“出错”。

一查代码才知道，由于这个脱机客户端使用的是webservice访问，.net开发团队在login功能中设了一个超时，30秒，30秒 timeout后就认为服务器没有开启，结果呢由于原来的apache设的是keeyalive和timeout 15秒，现在被改成了off，好家伙，根本就没有了这个timeout概念，因此每次.net登录直接被apache弹回来，因为没有了这个 timeout的接口了。

**由此可见，学东西。。。不能一知半解，务必求全面了解哈。**

**3.8 HostnameLookups设置为off**

尽量较少DNS查询的次数。如果你使用了任何”Allow fromdomain”或”Denyfrom domain”指令(也就是domain使用的是主机名而不是IP地址)，则代价是要进行两次DNS查询(一次正向和一次反向，以确认没有作假)。所以， 为了得到最高的性能，应该避免使用这些指令(**不用域名而用IP地址也是可以的**)。