Tomcat 性能的测试

一. 引言

性能测试与分析是软件开发过程中介于架构和调整的一个广泛并比较不容易理解的领域,更是一项较为复杂的活动。就像下棋游戏一样,有效的性能测试和分析只能在一个良好的计划策略和具备了对不可预料事件的处理能力的条件下顺利地完成。一个下棋高手赢得比赛靠的不仅仅是对游戏规则的认识,更是靠他的自己的能力和不断地专注于分析自己对手的实力来更加有效地利用和发挥规则的作用。同样一个优秀的性能测试和分析人员将要面对的是来自一个全新的应用程序和环境下带来的整个项目的挑战。本文中作者结合自己的使用经验和参考文档，对 Tomcat 性能方面的调整做一简要的介绍，并给出 Tomcat 性能的测试、分析和调整优化的一些方法。

二. 测量 Web 服务器的性能

测量 web 服务器的性能是一项让人感到畏缩的任务,但是我们在这里将给出一些需要注意的地方并且指点你了解其中更多的细节性的内容。它不像一些简单的任务，如测量 CPU 的速率或者是测量程序占用 CPU 的比例，web 服务器的性能优化中包括许调整许多变量来达到目标。许多的测量策略中都包含了一个看似简单的浏览实际上是在向服务器发送大量的请求,我们称之为客户端的程序，来测量响应时间。客户端和服务器端是在同一台机器上吗？服务器在测试的时候还运行着其它的什么程序吗？客户端和服务器端的通讯是通过局域网，100baseT，10baseT 还是使用调制解调器？客户端是否一直重复请求相同的页面，还是随机地访问不同的页面？（这些影响到了服务缓存的性能）客户端发送请求的有规律的还是突发的？你是在最终的配置环境下运行服务的还是在调试的配置环境下运行服务的？客户端请求中包含图片还是只有 HTML 页面？是否有请求是通过 servlets 和 JSP 的，CGI 程序，服务端包含（Server-Side Includes ，SSI 是一个可以让你使用动态 HTML 文件的技术）？所有这些都将是我们要关心的，并且几乎我们不可能精确地把所有的问题都清楚地列出来。

1.压力测试工具

“工欲善其事，必先利其器”，压力测试只有借助于一些工具才可得以实施。

大多数 web 压力测试工具的实现原理都是通过重复的大量的页面请求来模拟多用户对被测系统的并发访问,以此达到产生压力的目的。产生压力的手段都是通过录制或者是编写压力脚本,这些脚本以多个进程或者线程的形式在客户端运行，这样通过人为制造各种类型的压力，我们可以观察被测系统在各种压力状况下的表现，从而定位系统瓶颈，作为系统调优的基础。目前已经存在的性能测试工具林林总总,数量不下一百种,从单一的开放源码的免费小工具如 Aapache 自带的 web 性能测试工具 Apache Benchmark、开源的 Jmeter 到大而全的商业性能测试软件如 Mercury 的 LoadRunner 等等。任何性能测试工具都有其优缺点，我们可以根据实际情况挑选用最合适的工具。您可以在这里找到一些 web 压力测试工具
http://www.softwareqatest.com/qatweb1.html#LOAD

这里我们所使用的工具要支持 web 应用服务认证才可以，要支持接收发送 cookies，不仅如此 Tomcat 支持多种认证方式，比如基本认证、基于表单的认证、相互认证和客户端认证，而一些工具仅仅支持 HTTP 基本认证。真实地模拟用户认证是性能测试工具的一个重要的部分，因为

认证机制将对一个 web 站点的性能特征产生重要的影响。基于你在产品中使用的不同的认证方式，你需要从上面的工具列表中选择使用这种特性的测试工具。

Apache Benchmark 和 http_load 是命令行形式的工具，非常易于使用。Apache Benchmark 可以模仿单独的 URL 请求并且重复地执行，可以使用不同的命令行参数来控制执行迭代的次数，并发用户数等等。它的一个特点是可以周期性地打印出处理过程的信息，而其它工具只能给出一个全局的报告。

2.压力测试工具介绍

三. 外部环境的调整

在 Tomcat 和应用程序进行了压力测试后，如果您对应用程序的性能结果不太满意，就可以采取一些性能调整措施了，当然了前提是应用程序没有问题，我们这里只讲 Tomcat 的调整。由于 Tomcat 的运行依赖于 JVM，所以在这里我们把 Tomcat 的调整可以分为两类来详细描述：

外部环境调整

调整非 Tomcat 组件，例如 Tomcat 运行的操作系统和运行 Tomcat 的 java 虚拟机。

自身调整

修改 Tomcat 自身的参数，调整 Tomcat 配置文件中的参数。

下面我们将详细讲解外部环境调整的有关内容，Tomcat 自身调整的内容将在第 2 部分中阐述。1.JAVA 虚拟机性能优化

Tomcat 本身不能直接在计算机上运行，需要依赖于硬件基础之上的操作系统和一个 java 虚拟机。您可以选择自己的需要选择不同的操作系统和对应的 JDK 的版本（只要是符合 Sun 发布的 Java 规范的），但我们推荐您使用 Sun 公司发布的 JDK。确保您所使用的版本是最新的，因为 Sun 公司和其它一些公司一直在为提高性能而对 java 虚拟机做一些升级改进。一些报告显示 JDK1.4 在性能上比 JDK1.3 提高了将近 10%到 20%。

可以给 Java 虚拟机设置使用的内存，但是如果你的选择不对的话，虚拟机不会补偿。可通过命令行的方式改变虚拟机使用内存的大小。如下表所示有两个参数用来设置虚拟机使用内存的大小。

参数
 描述

-Xms<size>
 JVM 初始化堆的大小

-Xmx<size>
 JVM 堆的最大值

这两个值的大小一般根据需要进行设置。初始化堆的大小执行了虚拟机在启动时向系统申请

的内存的大小。一般而言，这个参数不重要。但是有的应用程序在大负载的情况下会急剧地占用更多的内存，此时这个参数就是显得非常重要，如果虚拟机启动时设置使用的内存比较小而在这种情况下有许多对象进行初始化，虚拟机就必须重复地增加内存来满足使用。由于这种原因，我们一般把-Xms 和-Xmx 设为一样大，而堆的最大值受限于系统使用的物理内存。一般使用数据量较大的应用程序会使用持久对象，内存使用有可能迅速地增长。当应用程序需要的内存超出堆的最大值时虚拟机就会提示内存溢出，并且导致应用服务崩溃。因此一般建议堆的最大值设置为可用内存的最大值的 80%。

Tomcat 默认可以使用的内存为 128MB，在较大型的应用项目中，这点内存是不够的，需要调大。

Windows 下，在文件{tomcat_home}/bin/catalina.bat，Unix 下，在文件{tomcat_home}/bin/catalina.sh 的前面，增加如下设置：

JAVA_OPTS='-Xms【初始化内存大小】 -Xmx【可以使用的最大内存】'

需要把这个两个参数值调大。例如：

JAVA_OPTS='-Xms256m -Xmx512m'

表示初始化内存为 256MB，可以使用的最大内存为 512MB。

另外需要考虑的是 Java 提供的垃圾回收机制。虚拟机的堆大小决定了虚拟机花费在收集垃圾上的时间和频度。收集垃圾可以接受的速度与应用有关，应该通过分析实际的垃圾收集的时间和频率来调整。如果堆的大小很大，那么完全垃圾收集就会很慢，但是频度会降低。如果你把堆的大小和内存的需要一致，完全收集就很快，但是会更加频繁。调整堆大小的的目的是最小化垃圾收集的时间，以在特定的时间内最大化处理客户的请求。在基准测试的时候，为保证最好的性能，要把堆的大小设大，保证垃圾收集不在整个基准测试的过程中出现。

如果系统花费很多的时间收集垃圾，请减小堆大小。一次完全的垃圾收集应该不超过 3-5 秒。如果垃圾收集成为瓶颈，那么需要指定代的大小，检查垃圾收集的详细输出，研究 垃圾收集参数对性能的影响。一般说来，你应该使用物理内存的 80% 作为堆大小。当增加处理器时，记得增加内存，因为分配可以并行进行，而垃圾收集不是并行的。

2.操作系统性能优化

这里说的操作系统是指运行 web 服务器的系统软件，当然，不同的操作系统是为不同的目的而设计的。比如 OpenBSD 是面向安全的，因此在它的内核中有许多的限制来防止不同形式的服务攻击（OpenBSD 的一句座右铭是“默认是最安全的”）。这些限制或许更多地用来运行活跃的 web 服务器。

而我们常用的 Linux 操作系统的目标是易用使用，因此它有着更高的限制。使用 BSD 内核的系统都带有一个名为“Generic”的内核，表明所有的驱动器都静态地与之相连。这样就使系统易于使用，但是如果你要创建一个自定义的内核来加强其中某些限制，那就需要排除不需要的设备。Linux 内核中的许多驱动都是动态地加载的。但是换而言之，内存现在变得越来越便宜，所

以因为加载额外的设备驱动就显得不是很重要的。重要的是要有更多的内存，并且在服务器上腾出更多的可用内存。

小提示：虽然现在内存已经相当的便宜，但还是尽量不要购买便宜的内存。那些有牌子的内存虽然是贵一点，但是从可靠性上来说，性价比会更高一些。

如果是在 Windows 操作系统上使用 Tomcat，那么最好选择服务器版本。因为在非服务器版本上，最终用户授权数或者操作系统本身所能承受的用户数、可用的网络连接数或其它方面的一些方面都是有限制的。并且基于安全性的考虑，必须经常给操作系统打上最新的补丁。

3.Tomcat 与其它 web 服务器整合使用

虽然 tomcat 也可以作 web 服务器,但其处理静态 html 的速度比不上 apache,且其作为 web 服务器的功能远不如 apache,因此我们想把 apache 和 tomcat 集成起来，将 html 与 jsp 的功能部分进行明确分工，让 tomcat 只处理 jsp 部分，其它的由 apache,IIS 等这些 web 服务器处理，由此大大节省了 tomcat 有限的工作“线程”。

4.负载均衡

在负载均衡的思路下，多台服务器为对称方式，每台服务器都具有同等的地位，可以单独对外提供服务而无须其他服务器的辅助。通过负载分担技术，将外部发送来的请求按一定规则分配到对称结构中的某一台服务器上，而接收到请求的服务器都独立回应客户机的请求。

提供服务的一组服务器组成了一个应用服务器集群(cluster),并对外提供一个统一的地址。当一个服务请求被发至该集群时，根据一定规则选择一台服务器，并将服务转定向给该服务器承担，即将负载进行均衡分摊。

通过应用负载均衡技术，使应用服务超过了一台服务器只能为有限用户提供服务的限制，可以利用多台服务器同时为大量用户提供服务。当某台服务器出现故障时，负载均衡服务器会自动进行检测并停止将服务请求分发至该服务器，而由其他工作正常的服务器继续提供服务，从而保证了服务的可靠性。

负载均衡实现的方式大概有四种：第一是通过 DNS，但只能实现简单的轮流分配，不能处理故障，第二如果是基于 MS IIS，Windows 2003 server 本身就带了负载均衡服务，第三是硬件方式，通过交换机的功能或专门的负载均衡设备可以实现，第四种是软件方式，通过一台负载均衡服务器进行，上面安装软件。使用 Apache Httpd Server 做负载平衡器，Tomcat 集群节点使用 Tomcat 就可以做到以上第四种方式。这种方式比较灵活，成本相对也较低。另外一个很大的优点就是可以根据应用的情况和服务器的情况采取一些策略。

四. 自身调整

本节将向您详细介绍一些加速可使 Tomcat 实例加速运行的技巧和方法，无论是在什么操作系统或者何种 Java 虚拟机上。在有些情况下，您可能没有控制部署环境上的操作系统或者 Java 虚拟机。在这种情况下，您就需要逐行了解以下的的一些建议，然而你应该在修改后使之生效。我认为以下方法是 Tomcat 性能自身调整的最佳方式。

1.禁用 DNS 查询

当 web 应用程序向要记录客户端的信息时，它也会记录客户端的 IP 地址或者通过域名服务器查找机器名转换为 IP 地址。DNS 查询需要占用网络，并且包括可能从很多很远的服务器或者不起作用的服务器上去获取对应的 IP 的过程，这样会消耗一定的时间。为了消除 DNS 查询对性能的影响我们可以关闭 DNS 查询，方式是修改 server.xml 文件中的 enableLookups 参数值：

Tomcat4

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector" port="80"
minProcessors="5" maxProcessors="75" enableLookups="false" redirectPort="8443"
acceptCount="100" debug="0" connectionTimeout="20000" useURIValidationHack="false"
disableUploadTimeout="true" />
```

Tomcat5

```
<Connector port="80" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" redirectPort="8443" acceptCount="100" debug="0"
connectionTimeout="20000" disableUploadTimeout="true"/>
```

除非你需要连接到站点的每个 HTTP 客户端的机器名，否则我们建议在生产环境上关闭 DNS 查询功能。可以通过 Tomcat 以外的方式来获取机器名。这样不仅节省了网络带宽、查询时间和内存，而且更小的流量会使日志数据也会变得更少，显而易见也节省了硬盘空间。对流量较小的站点来说禁用 DNS 查询可能没有大流量站点的效果明显，但是此举仍不失为一良策。谁又见到一个低流量的网站一夜之间就流量大增呢？

2.调整线程数

另外一个可通过应用程序的连接器（Connector）进行性能控制的的参数是创建的处理请求的线程数。Tomcat 使用线程池加速响应速度来处理请求。在 Java 中线程是程序运行时的路径，是在一个程序中与其它控制线程无关的、能够独立运行的代码段。它们共享相同的地址空间。多线程帮助程序员写出 CPU 最大利用率的高效程序，使空闲时间保持最低，从而接受更多的请求。

Tomcat4 中可以通过修改 minProcessors 和 maxProcessors 的值来控制线程数。这些值在安装后就已经设定为默认值并且是足够使用的,但是随着站点的扩容而改大这些值。minProcessors 服务器启动时创建的处理请求的线程数应该足够处理一个小量的负载。也就是说，如果一天内每秒仅发生 5 次单击事件,并且每个请求任务处理需要 1 秒钟,那么预先设置线程数为 5 就足够了。但在你的站点访问量较大时就需要设置更大的线程数，指定为参数 maxProcessors 的值。maxProcessors 的值也是有上限的，应防止流量不可控制（或者恶意的服务攻击），从而导致超出了虚拟机使用内存的大小。如果要加大并发连接数，应同时加大这两个参数。web server 允

许的最大连接数还受制于操作系统的内核参数设置,通常 Windows 是 2000 个左右,Linux 是 1000 个左右。

在 Tomcat5 对这些参数进行了调整，请看下表：

属性名

描述

maxThreads

Tomcat 使用线程来处理接收的每个请求。这个值表示 Tomcat 可创建的最大的线程数。

acceptCount

指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理。

connnectionTimeout

网络连接超时，单位：毫秒。设置为 0 表示永不超时，这样设置有隐患的。通常可设置为 30000 毫秒。

minSpareThreads

Tomcat 初始化时创建的线程数。

maxSpareThreads

一旦创建的线程超过这个值，Tomcat 就会关闭不再需要的 socket 线程。

最好的方式是多设置几次并且进行测试，观察响应时间和内存使用情况。在不同的机器、操作系统或虚拟机组合的情况下可能会不同，而且并不是所有人的 web 站点的流量都是一样的，因此没有一刀切的方案来确定线程数的值。

3.加速 JSP 编译速度

当第一次访问一个 JSP 文件时，它会被转换为 Java serverlet 源码，接着被编译成 Java 字节码。你可以控制使用哪个编译器，默认情况下，Tomcat 使用使用命令行 javac 进行使用的编译器。也可以使用更快的编译器，但是这里我们将介绍如何优化它们。

另外一种方法是不要把所有的实现都使用 JSP 页面，而是使用一些不同的 java 模板引擎变量。显然这是一个跨越很大的决定，但是事实证明至少这种方法是只得研究的。如果你想了解更多有关在 Tomcat 可使用的模板语言,你可以参考 Jason Hunter 和 William Crawford 合著的《Java Servlet Programming 》一书（O'Reilly 公司出版）。

在 Tomcat 4.0 中可以使用流行而且免费的 Jikes 编译器。Jikes 编译器的速度要由于 Sun 的 Java 编译器。首先要安装 Jikes（可访问 http://oss.software.ibm.com/pub/jikes 获得更多的信息),接着需要在环境变量中设置 JIKESPATH 包含系统运行时所需的 JAR 文件。装好 Jikes 以后还需要设置让 JSP 编译 servlet 使用 Jikes，需要修改 web.xml 文件中 jspCompilerPlugin 的值：

```
<servlet>
<servlet-name>jsp</servlet-name>
<servlet-class>
org.apache.jasper.servlet.JspServlet
</servlet-class><init-param>
<param-name>logVerbosityLevel</param-name>
<param-value>WARNING</param-value>
</init-param>
<init-param>
<param-name>jspCompilerPlugin</param-name>
<param-value>
org.apache.jasper.compiler.JikesJavaCompiler
</param-value>
</init-param>
<init-param>
<!-- <param-name>
org.apache.catalina.jsp_classpath
</param-name> -->
<param-name>classpath</param-name>
<param-value>
/usr/local/jdk1.3.1-linux/jre/lib/rt.jar:
/usr/local/lib/java/servletapi/servlet.ja
r</param-value>
</init-param>
<load-on-startup>3</load-on-startup>
</servlet>
```

在 Tomcat 4.1（或更高版本），JSP 的编译由包含在 Tomcat 里面的 Ant 程序控制器直接执行。这听起来有一点点奇怪，但这正是 Ant 有意为之的一部分，有一个 API 文档指导开发者在没有启动一个新的 JVM 的情况下，使用 Ant。这是使用 Ant 进行 Java 开发的一大优势。另外，这也意味着你现在能够在 Ant 中使用任何 javac 支持的编译方式，这里有一个关于 Apache Ant 使用手册的 javac page 列表。使用起来是容易的，因为你只需要在 元素中定义一个名字叫“compiler”，并且在 value 中有一个支持编译的编译器名字，示例如下：

```
<servlet>
<servlet-name>jsp</servlet-name>
<servlet-class>
org.apache.jasper.servlet.JspServlet
</servlet-class>
<init-param>
<param-name>logVerbosityLevel</param-name>
```

```
<param-value>WARNING</param-value>
</init-param>
<init-param>
<param-name>compiler</param-name>
<param-value>jikes</param-value>
</init-param>
<load-on-startup>3</load-on-startup>
</servlet>
```

Ant 可用的编译器

名称
 别名
 调用的编译器

classic
 javac1.1, javac1.2
 Standard JDK 1.1/1.2 compiler

modern
 javac1.3, javac1.4
 Standard JDK 1.3/1.4 compiler

jikes
 The Jikes compiler

JVC Microsoft
 Microsoft command-line compiler from the Microsoft SDK for Java/Visual J++

KJC The kopi compiler

GCJ The gcj compiler (included as part of gcc)

SJ Symantec
 Symantec's Java compiler

extJavac
 Runs either the modern or classic compiler in a JVM of its own

由于 JSP 页面在第一次使用时已经被编译，那么你可能希望在更新新的 jsp 页面后马上对它进行编译。实际上，这个过程完全可以自动化，因为可以确认的是新的 JSP 页面在生产服务器和在测试服务器上的运行效果是一样的。

在 Tomcat4 的 bin 目录下有一个名为 jspc 的脚本。它仅仅是运行翻译阶段，而不是编译阶段，使用它可以在当前目录生成 Java 源文件。它是调试 JSP 页面的一种有力的手段。

可以通过浏览器访问再确认一下编译的结果。这样就确保了文件被转换成 serverlet，被编译了可直接执行。这样也准确地模仿了真实用户访问 JSP 页面，可以看到给用户提供的功能。也抓紧这最后一刻修改出现的 bug 并且修改它 J

Tomcat 提供了一种通过请求来编译 JSP 页面的功能。例如，你可以在浏览器地址栏中输入 http://localhost:8080/examples/jsp/dates/date.jsp?jsp_precompile=true，这样 Tomcat 就会编译 data.jsp 而不是执行它。此举唾手可得，不失为一种检验页面正确性的捷径。

4. 其它

前面我们提到过操作系统通过一些限制手段来防止恶意的服务攻击，同样 Tomcat 也提供了防止恶意攻击或禁止某些机器访问的设置。

Tomcat 提供了两个参数供你配置：RemoteHostValve 和 RemoteAddrValve。

通过配置这两个参数,可以让你过滤来自请求的主机或 IP 地址,并允许或拒绝哪些主机/IP。与之类似的，在 Apache 的 httpd 文件里有对每个目录的允许/拒绝指定。

例如你可以把 Admin Web application 设置成只允许本地访问，设置如下：

```
<Context path="/path/to/secret_files" ...>
<Valve className="org.apache.catalina.valves.RemoteAddrValve"

allow="127.0.0.1" deny=""/>
</Context>
```

如果没有给出允许主机的指定，那么与拒绝主机匹配的主机就会被拒绝，除此之外的都是允许的。与之类似，如果没有给出拒绝主机的指定，那么与允许主机匹配的主机就会被允许，除此之外的都是拒绝的。

五. 容量计划

容量计划是在生产环境中使用 Tomcat 不得不提的提高性能的另一个重要的话题。如果你没有对预期的网络流量下的硬件和带宽做考虑的话那么无论你如何做配置修改和测试都无济于事。

这里先对提及的容量计划作一个简要的定义:容量计划是指评估硬件、操作系统和网络带宽，确定应用服务的服务范围，寻求适合需求和软件特性的软硬件的一项活动。因此这里所说的软件不仅包括 Tomcat，也包括与 Tomcat 结合使用的任何第三方 web 服务器软件。

如果在购买软硬件或部署系统前你对容量计划一无所知，不知道现有的软硬件环境能够支撑多少的访问量,甚至更糟直到你已经交付并且在生产环境上部署产品后才意识到配置有问题时再

进行变更可能为时已晚。此时只能增加硬件投入，增加硬盘容量甚至购买更好的服务器。如果事先做了容量计划那么就不会搞的如此焦头烂额了。

我们这里只介绍与 Tomcat 相关的内容。

首先为了确定 Tomcat 使用机器的容量计划，你应该从一下列表项目种着手研究和计划：

1. 硬件

采用什么样的硬件体系？需要多少台计算机？使用一个大型的,还是使用多台小型机？每个计算机上使用几个 CPU？使用多少内存？使用什么样的存储设备，I/O 的处理速度有什么要求？怎样维护这些计算机？不同的 JVM 在这些硬件上运行的效果如何（比如 IBM AIX 系统只能在其设计的硬件系统上运行）？

2. 网络带宽

带宽的使用极限是多少？web 应用程序如何处理过多的请求？

3. 服务端操作系统

采用哪种操作系统作为站点服务器最好？在确定的操作系统上使用哪个 JVM 最好？例如，JVM 在这种系统上是否支持本地多线程，对称多处理？哪种系统可使 web 服务器更快、更稳定，并且更便宜。是否支持多 CPU?

4. Tomcat 容量计划

以下介绍针对 Tomcat 做容量计划的步骤：

1） 量化负载。如果站点已经建立并运行，可以使用前面介绍的工具模仿用户访问，确定资源的需求量。

2） 针对测试结果或测试过程中进行分析。需要知道那些请求造成了负载过重或者使用过多的资源，并与其它请求做比较，这样就确定了系统的瓶颈所在。例如：如果 servlet 在查询数据库的步骤上耗用较长的时间，那么就需要考虑使用缓冲池来降低响应时间。

3） 确定性能最低标准。例如，你不想让用户花 20 秒来等待结果页面的返回，也就是说甚至在达到访问量的极限时，用户等待的时间也不能超过 20 秒种（从点击链接到看到返第一条返回数据）。这个时间中包含了数据库查询时间和文件访问时间。同类产品性能在不同的公司可能有不同的标准，一般最好采取同行中的最低标准或对这个标准做出评估。

4） 确定如何合理使用底层资源，并逐一进行测试。底层资源包括 CPU、内存、存储器、带宽、操作系统、JVM 等等。在各种生产环境上都按顺序进行部署和测试，观察是否符合需求。在测试 Tomcat 时尽量多采用几种 JVM，并且调整 JVM 使用内存和 Tomcat 线程池的大小进行测试。同时为了达到资源充分合理稳定地使用的效果,还需针对测试过程中出现的硬件系统瓶颈进行处理确定合理的资源配置。这个过程最为复杂，而且一般由于没有可参考的值所以只能靠理论推断和经验总结。

5） 如果通过第 4 步的反复测试如果达到了最优的组合，就可以在相同的生产环境上部署产品了。

此外应牢记一定要文档化你的测试过程和结果，因为此后可能还会进行测试，这样就可以拿以前的测试结果做为参考。另外测试过程要反复多次进行，每次的条件可能都不一样，因此只有记录下来才能进行结果比较和最佳条件的选择。

这样我们通过测试找到了最好的组合方式，各种资源得到了合理的配置，系统的性能得到了极大的提升。

六. 附加资料

很显然本文也很难全面而详尽地阐述性能优化过程。如果你进行更多研究的话可能会把性能调优做的更好，比如 Java 程序的性能调整、操作系统的调整、各种复杂环境与应用系统和其它所有与应用程序相关的东西。在这里提供一些文中提到的一些资源、文中提到的相关内容的链接以及本文的一些参考资料。

1. Web 性能测试资料及工具

1） Jmeter Wiki 首页，Jmeter 为一个开源的 100%Java 开发的性能测试工具
http://wiki.apache.org/jakarta-jmeter/

2） Apache Benchmark 使用说明
http://httpd.apache.org/docs-2.0/programs/ab.html

3） 一些 Java 相关测试工具的介绍，包含可以与 Tomcat 集成进行测试的工具
http://blog.csdn.net/wyingquan/

4） 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。它通过模拟数据以千万计用户来实施并发负载来对整个企业架构进行测试，来帮助您更

快的查找和发现问题。
http://www.mercury.com/us/products/performance-center/loadrunner/

2. 文中介绍的相关内容的介绍

1） Apache 2.x + Tomcat 4.x 做负载均衡，描述了如何利用 jk 配置集群的负载均衡。
http://raibledesigns.com/tomcat/index.html

2） 容量计划的制定，收集了许多有关制定 web 站点容量计划的例子：
http://www.capacityplanning.com/

3） 评测 Tomcat5 负载平衡与集群，
http://www.javaresearch.org/article/showarticle.jsp?column=556&thread=19777

4） Apache 与 Tomcat 的安装与整合之整合篇
http://www.javaresearch.org/article/showarticle.jsp?column=23&thread=18139

5） 性能测试工具之研究，介绍了性能测试工具的原理与思路
http://www.51testing.com/emagzine/No2_2.htm

6） Java 的内存泄漏
http://www.matrix.org.cn/resource/article/409.html

7） Web 服务器和应用程序服务器有什么区别？
http://www.matrix.org.cn/resource/article/1429.html

8） 详细讲解性能中数据库集群的问题
http://www.theserverside.com/articles/article.tss?l=DB_Break