**性能调优**

## 一、JVM调优

**1 Java对象的大小**

基本数据的类型的大小是固定的，这里就不多说了。对于非基本类型的Java对象，其大小就值得商榷。在Java中，一个空Object对象的大小是8byte，这个大小只是保存堆中一个没有任何属性的对象的大小。看下面语句：

Object ob = new Object();

这样在程序中完成了一个Java对象的生命，但是它所占的空间为：4byte+8byte。4byte是上面部分所说的Java栈中保存引用的所需要的空间。而那8byte则是Java堆中对象的信息。因为所有的Java非基本类型的对象都需要默认继承Object对象，因此不论什么样的Java对象，其大小都必须是大于8byte。

有了Object对象的大小，我们就可以计算其他对象的大小了。

Class NewObject

{

int count;

boolean flag;

Object ob;

}

其大小为：

空对象大小(8byte)+int大小(4byte)+Boolean大小(1byte)+空Object引用的大小(4byte)=17byte。

但是因为Java在对对象内存分配时都是以8的整数倍来分，因此大于17byte的最接近8的整数倍的是24，因此此对象的大小为24byte。

**2 基本类型的包装类型的大小**

因为这种包装类型已经成为对象了，因此需要把他们作为对象来看待。包装类型的大小至少是12byte（声明一个空Object至少需要的空间），而且12byte没有包含任何有效信息，同时，因为Java对象大小是8的整数倍，因此一个基本类型包装类的大小至少是16byte。这个内存占用是很恐怖的，**它是使用基本类型的N倍**（N>2），有些类型的内存占用更是夸张。因此，可能的话应尽量少使用包装类。在JDK5.0以后，因为加入了自动类型装换，因此，Java虚拟机会在存储方面进行相应的优化。

**3 基本垃圾回收算法**

**1 引用计数（Reference Counting）:**

比较古老的回收算法。原理是此对象有一个引用，即增加一个计数，删除一个引用则减少一个计数。垃圾回收时，只用收集计数为0的对象。此算法最致命的是无法处理循环引用的问题。

**2 标记-清除（Mark-Sweep）:**

此算法执行分两阶段。第一阶段从引用根节点开始标记所有被引用的对象，第二阶段遍历整个堆，把未标记的对象清除。此算法需要暂停整个应用，同时，会产生内存碎片。

**3 复制法（Copying）:**

此算法把内存空间划为两个相等的区域，每次只使用其中一个区域。垃圾回收时，遍历当前使用区域，把正在使用中的对象复制到另外一个区域中。次算法每次只处理正在使用中的对象，因此复制成本比较小，同时复制过去以后还能进行相应的内存整理，不会出现“碎片”问题。当然，此算法的缺点也是很明显的，就是需要两倍内存空间。

**4 标记-整理（Mark-Compact）:**

此算法结合了“标记-清除”和“复制”两个算法的优点。也是分两阶段，第一阶段从根节点开始标记所有被引用对象，第二阶段遍历整个堆，把清除未标记对象并且把存活对象“压缩”到堆的其中一块，按顺序排放。此算法避免了“标记-清除”的碎片问题，同时也避免了“复制”算法的空间问题。

**4 如何区分垃圾**

上面说到的“引用计数”法，通过统计控制生成对象和删除对象时的引用数来判断。垃圾回收程序收集计数为0的对象即可。但是这种方法无法解决循环引用。所以，后来实现的垃圾判断算法中，都是从程序运行的根节点出发，遍历整个对象引用，查找存活的对象。那么在这种方式的实现中，垃圾回收从哪儿开始的呢？即，从哪儿开始查找哪些对象是正在被当前系统使用的。上面分析的堆和栈的区别，其中栈是真正进行程序执行地方，所以要获取哪些对象正在被使用，则需要从Java栈开始。

同时，一个栈是与一个线程对应的，因此，如果有多个线程的话，则必须对这些线程对应的所有的栈进行检查。同时，除了栈外，还有系统运行时的寄存器等，也是存储程序运行数据的。

**这样，以栈或寄存器中的引用为起点，我们可以找到堆中的对象，又从这些对象找到对堆中其他对象的引用，这种引用逐步扩展，最终以null引用或者基本类型结束**，这样就形成了一颗以Java栈中引用所对应的对象为根节点的一颗对象树，如果栈中有多个引用，则最终会形成多颗对象树。在这些对象树上的对象，都是当前系统运行所需要的对象，不能被垃圾回收。而其他剩余对象，则可以视为无法被引用到的对象，可以被当做垃圾进行回收。因此，垃圾回收的起点是一些根对象（java栈, 静态变量, 寄存器...）。而最简单的Java栈就是Java程序执行的main函数。这种回收方式，也是上面提到的“标记-清除”的回收方式

**5 如何处理碎片**

由于不同Java对象存活时间是不一定的，因此，在程序运行一段时间以后，如果不进行内存整理，就会出现零散的内存碎片。碎片最直接的问题就是会导致无法分配大块的内存空间，以及程序运行效率降低。

**所以，在上面提到的基本垃圾回收算法中，“复制”方式和“标记-整理”方式，都可以解决碎片的问题。**

**6 分代垃圾回收**

分代的垃圾回收策略，是基于这样一个事实：不同的对象的生命周期是不一样的。因此，不同生命周期的对象可以采取不同的收集方式，以便提高回收效率。

在Java程序运行的过程中，会产生大量的对象，其中有些对象是与业务信息相关，比如Http请求中的Session对象、线程、Socket连接，这类对象跟业务直接挂钩，因此生命周期比较长。但是还有一些对象，主要是程序运行过程中生成的临时变量，这些对象生命周期会比较短，比如：String对象，由于其不变类的特性，系统会产生大量的这些对象，有些对象甚至只用一次即可回收。

试想，在不进行对象存活时间区分的情况下，每次垃圾回收都是对整个堆空间进行回收，花费时间相对会长，同时，因为每次回收都需要遍历所有存活对象，但实际上，对于生命周期长的对象而言，这种遍历是没有效果的，因为可能进行了很多次遍历，但是他们依旧存在。因此，分代垃圾回收采用分治的思想，进行代的划分，把不同生命周期的对象放在不同代上，不同代上采用最适合它的垃圾回收方式进行回收。

**7 如何分代？**

**虚拟机中的共划分为三个代：年轻代（Young Generation）、年老点（Old Generation）和持久代（PermanentGeneration）。**

**1 年轻代:**

所有新生成的对象首先都是放在年轻代的。年轻代的目标就是尽可能快速的收集掉那些生命周期短的对象。年轻代分三个区。**一个Eden区，两个Survivor区(一般而言)。大部分对象在Eden区中生成。**当Eden区满时，还存活的对象将被复制到Survivor区（两个中的一个），当这个Survivor区满时，此区的存活对象将被复制到另外一个Survivor区，当这个Survivor去也满了的时候，从第一个Survivor区复制过来的并且此时还存活的对象，将被复制“年老区(Tenured)”。需要注意，Survivor的两个区是对称的，没先后关系，所以同一个区中可能同时存在从Eden复制过来对象，和从前一个Survivor复制过来的对象，而复制到年老区的只有从第一个Survivor去过来的对象。而且，Survivor区总有一个是空的。

**2 年老代:**

在年轻代中经历了N次垃圾回收后仍然存活的对象，就会被放到年老代中。因此，可以认为年老代中存放的都是一些生命周期较长的对象。

**3 持久代:**

用于存放静态文件，如Java类、方法等。持久代对垃圾回收没有显著影响，但是有些应用可能动态生成或者调用一些class，例如Hibernate等，在这种时候需要设置一个比较大的持久代空间来存放这些运行过程中新增的类。持久代大小通过-XX:MaxPermSize=<N>进行设置。

**8 什么情况下触发垃圾回收**

由于对象进行了分代处理，因此垃圾回收区域、时间也不一样。GC有两种类型：Scavenge GC和Full GC。

**1 Scavenge GC**

一般情况下，当新对象生成，并且在Eden申请空间失败时，就会触发Scavenge GC，对Eden区域进行GC，清除非存活对象，并且把尚且存活的对象移动到Survivor区。

然后整理Survivor的两个区。这种方式的GC是对年轻代的Eden区进行，不会影响到年老代。因为大部分对象都是从Eden区开始的，同时Eden区不会分配的很大，所以Eden区的GC会频繁进行。因而，一般在这里需要使用速度快、效率高的算法，使Eden去能尽快空闲出来。

**2 Full GC**

对整个堆进行整理，包括Young、Tenured和Perm。Full GC因为需要对整个对进行回收，所以比ScavengeGC要慢，因此应该尽可能减少FullGC的次数。在对JVM调优的过程中，很大一部分工作就是对于FullGC的调节。有如下原因可能导致Full GC：

@年老代（Tenured）被写满

@持久代（Perm）被写满

@System.gc()被显示调用

@上一次GC之后Heap的各域分配策略动态变化

**9 JVM调优总结举例1**

以下配置主要针对分代垃圾回收算法而言。堆大小设置年轻代的设置很关键

JVM中最大堆大小有三方面限制：相关操作系统的数据模型（32-bt还是64-bit）限制；系统的可用虚拟内存限制；系统的可用物理内存限制。32位系统下，一般限制在1.5G~2G；64为操作系统对内存无限制。在Windows Server 2003 系统，3.5G物理内存，JDK5.0下测试，最大可设置为1478m。

典型设置：

**java -Xmx3550m -Xms3550m -Xmn2g –Xss128k**

-Xmx3550m：设置JVM最大可用内存为3550M。

-Xms3550m：设置JVM初始内存为3550m。此值可以设置与-Xmx相同，以避免每次垃圾回收完成后JVM重新分配内存。

-Xmn2g：设置年轻代大小为2G。整个堆大小=年轻代大小 + 年老代大小 + 持久代大小。

持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。

-Xss128k：设置每个线程的堆栈大小。JDK5.0以后每个线程堆栈大小为1M，以前每个线程堆栈大小为256K。更具应用的线程所需内存大小进行调整。

在相同物理内存下，减小这个值能生成更多的线程。但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在3000~5000左右。

**java -Xmx3550m -Xms3550m -Xss128k -XX:NewRatio=4 -XX:SurvivorRatio=4 XX:MaxPermSize=16m -XX:MaxTenuringThreshold=0**

-XX:NewRatio=4:设置年轻代（包括Eden和两个Survivor区）与年老代的比值（除去持久代）。设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5

-XX:SurvivorRatio=3：设置年轻代中Eden区与Survivor区的大小比值。设置为3，则两个Survivor区与一个Eden区的比值为2:3

当我们设置参数-Xmn500m(表示：设置年轻代为500m)时，我们知道年轻代中有两个Survivor和Eden

Survivor:Eden=1:3，所以,Surivivor占年轻代的2/5,Eden占3/5。

Survivor=500 \* 2/5=200m

Eden=500 \* 3/5=300m

-XX:MaxPermSize=16m:设置持久代大小为16m。

-XX:MaxTenuringThreshold=0：设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代。对于年老代比较多的应用，可以提高效率。

如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论。

**10 JVM调优总结举例2**

常见配置汇总

-Xms:初始堆大小 -Xmx:最大堆大小 -XX:NewSize=n:设置年轻代大小 -XX:NewRatio=n:设置年轻代和年老代的比值。如:为3，表示年轻代与年老代比值为1：3，年轻代占整个年

轻代年老代和的1/4

-XX:SurvivorRatio=n:年轻代中Eden区与两个Survivor区的比值。注意Survivor区有两个。如：3，表示Eden：Survivor=3：2，一个Survivor区占整个年轻代的1/5

-XX:MaxPermSize=n:设置持久代大小

收集器设置

-XX:+UseSerialGC:设置串行收集器

-XX:+UseParallelGC:设置并行收集器

-XX:+UseParalledlOldGC:设置并行年老代收集器

-XX:+UseConcMarkSweepGC:设置并发收集器垃圾回收统计信息

-XX:+PrintGC

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-Xloggc:filename

并行收集器设置

-XX:ParallelGCThreads=n:设置并行收集器收集时使用的CPU数。并行收集线程数。

-XX:MaxGCPauseMillis=n:设置并行收集最大暂停时间

-XX:GCTimeRatio=n:设置垃圾回收时间占程序运行时间的百分比。公式为1/(1+n)并发收集器设置

-XX:+CMSIncrementalMode:设置为增量模式。适用于单CPU情况。

-XX:ParallelGCThreads=n:设置并发收集器年轻代收集方式为并行收集时，使用的CPU数。并行收集线程数。

调优总结

年轻代大小选择

响应时间优先的应用：尽可能设大，直到接近系统的最低响应时间限制（根据实际情况选择）。在此种情况下，年轻代收集发生的频率也是最小的。同时，减少到达年老代的对象。

吞吐量优先的应用：尽可能的设置大，可能到达Gbit的程度。因为对响应时间没有要求，垃圾收集可以并行进行，一般适合8CPU以上的应用。

年老代大小选择

响应时间优先的应用：年老代使用并发收集器，所以其大小需要小心设置，一般要考虑并发会话率和会话持续时间等一些参数。如果堆设置小了，可以会造成内存碎片、高回收频率以及应用暂停而使用传统的标记清除方式；如果堆大了，则需要较长的收集时间。最优化的方案，一般需要参考以下数据获得：

1. 并发垃圾收集信息

2. 持久代并发收集次数

3. 传统GC信息

4. 花在年轻代和年老代回收上的时间比例减少年轻代和年老代花费的时间，一般会提高应用的效率

吞吐量优先的应用

一般吞吐量优先的应用都有一个很大的年轻代和一个较小的年老代。原因是，这样可以尽可能回收掉大部分短期对象，减少中期的对象，而年老代尽存放长期存活对象。

较小堆引起的碎片问题

因为年老代的并发收集器使用标记、清除算法，所以不会对堆进行压缩。当收集器回收时，他会把相邻的空间进行合并，这样可以分配给较大的对象。但是，当堆空间较小时，运行一段时间以后，就会出现“碎片”，如果并发收集器找不到足够的空间，那么并发收集器将会停止，然后使用传统的标记、清除方式进行回收。如果出现“碎片”，可能需要进行如下配置：

1.-XX:+UseCMSCompactAtFullCollection：使用并发收集器时，开启对年老代的压缩。

2.-XX:CMSFullGCsBeforeCompaction=0：上面配置开启的情况下，这里设置多少次Full GC后，对年老代进行压缩

## 二、MySQL性能调优

**1 分析大致原因**

(1)硬件配置低：查看应用设备的使用率CPU内存存储(硬盘) 用top,sar,uptime,free

(2)网络带宽： 使用网络测速软件 网速;

(3)提供数据库服务软件版本低：

**2 确定MySQL的最大连接数**

在MySQL数据库中，MySQL的最大并发连接数是存储在全局变量max\_connections中的。

mysql> show variables like “**max\_connections**”; #允许的最大并发连接数

mysql> show variables like "**connect\_timeout**"; #建立连接时，三次握手的超时时间

mysql> show variables like "**wait\_timeout**"; #建立连接后，等待断开连接的超时时间

查看当前已使用的连接数：mysql>show global status like “max\_used\_connections”;

查看默认的最大连接数：mysql> show variables like "max\_connections";

**85%原则。**

**3 为临时表分配足够的内存**

在某些情况下，服务器在处理语句时会创建内部临时表。临时表用于内部操作如GROUP BY和distinct，还有一些ORDER BY查询以及UNION和FROM子句(派生表)中的子查询。

这些都是在内存中创建的内存表。内存中临时表的最大大小由tmp\_table\_size和max\_heap\_table\_size中较小的值确定。如果临时表的大小超过这个阈值，则将其转换为磁盘上的InnoDB或MyISAM表。此外，如果查询涉及BLOB或TEXT列，而这些列不能存储在内存表中，临时表总是直接指向磁盘。

这种转换的代价很大，所以考虑增加**max\_heap\_table\_size和tmp\_table\_size**变量的大小来帮助减少在磁盘上创建临时表的数量。

当增加max\_heap\_table\_size和tmp\_table\_size 变量的大小时，一定要监视服务器的内存使用情况，因为内存中的临时表可能会增加达到服务器内存容量的风险。

一般来说，**32M到64M**是建议值，从这两个变量开始并根据需要进行调优。

允许的最大值：显示tmp\_table\_size服务器变量的值，它定义了在内存中创建的临时表的最大大小。与max\_heap\_table\_size一起，定义了可以在内存中创建的临时表的最大大小。如果内存临时表大于此大小则将其存储在磁盘上。

内存表的最大大小：显示max\_heap\_table\_size服务器变量的值，该值定义了显式创建的MEMORY存储引擎表的最大大小。

创建的临时表总数：显示created\_tmp\_tables服务器变量的值，它定义了在内存中创建的临时表的数量。

**4 增加线程缓存大小**

连接管理器线程处理服务器监听的网络接口上的客户端连接请求。连接管理器线程将每个客户端连接与专用于它的线程关联，该线程负责处理该连接的身份验证和所有请求处理。

因此，线程和当前连接的客户端之间是一对一的比例。确保线程缓存足够大以容纳所有传入请求是非常重要的。

MySQL提供了许多与连接线程相关的服务器变量：

线程缓存大小由thread\_cache\_size系统变量决定。默认值为0(无缓存)，这将导致为每个新连接设置一个线程，并在连接终止时需要处理该线程。

如果希望服务器每秒接收数百个连接请求，那么应该将thread\_cache\_size设置的足够高，以便大多数新连接可以使用缓存线程。可以在服务器启动或运行时设置max\_connections的值。

thread\_cache\_size：可以缓存的线程数。

Threads\_cached：缓存中的线程数。

Threads\_created：创建用于处理连接的线程。

**5 选择合适的存储引擎: InnoDB**

MyISAM 只缓存索引，而 InnoDB 缓存数据和索引，MyISAM 不支持事务

InnoDB缓冲池大小： InnoDB缓冲池大小在使用InnoDB的MySQL数据库中起着至关重要的作用。缓冲池同时缓存数据和索引。它的值应该尽可能的大，以确保数据库使用内存而不是硬盘驱动器进行读取操作。

InnoDB日志缓冲区大小： MySQL每次写入日志文件时，它都会利用可用于处理销售数据的重要系统资源。因此，将InnoDB日志缓冲区大小设置为较大值才有意义。

这样，服务器在大型事务中写入磁盘的次数就减少了，从而最大限度地减少了这些耗时的操作。64M是这个变量的一个很好的起点。

**6 足够大的 innodb\_buffer\_pool\_size**

推荐将数据完全保存在 innodb\_buffer\_pool\_size ，即按存储量规划 innodb\_buffer\_pool\_size 的容量。这样你可以完全从内存中读取数据，最大限度减少磁盘操作。

如何确定 innodb\_buffer\_pool\_size 足够大，数据是从内存读取而不是硬盘？

mysql> SHOW GLOBAL STATUS LIKE 'innodb\_buffer\_pool\_pages\_%';

+----------------------------------+--------+

| Variable\_name | Value |

+----------------------------------+--------+

| Innodb\_buffer\_pool\_pages\_data | 129037 |

| Innodb\_buffer\_pool\_pages\_dirty | 362 |

| Innodb\_buffer\_pool\_pages\_flushed | 9998 |

| Innodb\_buffer\_pool\_pages\_free | 0 | !!!!!!!!

| Innodb\_buffer\_pool\_pages\_misc | 2035 |

| Innodb\_buffer\_pool\_pages\_total | 131072 |

+----------------------------------+--------+

6 rows in set (0.00 sec)

**发现 Innodb\_buffer\_pool\_pages\_free 为 0，则说明 buffer pool 已经被用光，需要增大 innodb\_buffer\_pool\_size**

**7 减少磁盘写入操作**

使用足够大的写入缓存 innodb\_log\_file\_size

但是需要注意如果用 1G 的 innodb\_log\_file\_size ，假如服务器当机，需要 10 分钟来恢复。

推荐 innodb\_log\_file\_size 设置为 0.25 \* innodb\_buffer\_pool\_size

innodb\_flush\_log\_at\_trx\_commit这个选项和写磁盘操作密切相关：

innodb\_flush\_log\_at\_trx\_commit = 1 则每次修改写入磁盘

innodb\_flush\_log\_at\_trx\_commit = 0/2 每秒写入磁盘

如果你的应用不涉及很高的安全性 (金融系统)，或者基础架构足够安全，或者 事务都很小，都可以用 0 或者 2 来降低磁盘操作。

**8 提高磁盘读写速度**

RAID0 尤其是在使用 EC2 这种虚拟磁盘 (EBS) 的时候，使用软 RAID0 非常重要。

**9 充分使用索引**

索引是提高查询速度的唯一方法，比如搜索引擎用的倒排索引是一样的原理。

索引的添加需要根据查询来确定，比如通过慢查询日志或者查询日志,或者通过 EXPLAIN 命令分析查询。

**10 分析查询日志和慢查询日志**

记录所有查询，这在用 ORM 系统或者生成查询语句的系统很有用。

log=/var/log/mysql.log

注意不要在生产环境用，否则会占满你的磁盘空间。

记录执行时间超过 1 秒的查询：

long\_query\_time=1

log-slow-queries=/var/log/mysql/log-slow-queries.log

**11 用 NOSQL 的方式使用 MYSQL**

B-TREE 仍然是最高效的索引之一，所有 MYSQL 仍然不会过时。

用 HandlerSocket 跳过 MYSQL 的 SQL 解析层，MYSQL 就真正变成了 NOSQL。

## 三、Nginx优化

**1 隐藏 Nginx 版本号**

为什么要隐藏 Nginx 版本号：一般来说，软件的漏洞都与版本有关，隐藏版本号是为了防止恶意用户利用软件漏洞进行攻击

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

worker\_processes 1;

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server\_tokens off; # 隐藏版本号

server {

listen 80;

server\_name www.abc.com;

location / {

root html/www;

index index.html index.htm;

}

}

}

[root@localhost ~]# /usr/local/nginx/sbin/nginx -t

[root@localhost ~]# /usr/local/nginx/sbin/nginx -s reload

[root@localhost ~]# curl -I 127.0.0.1 # 查看是否隐藏版本号

HTTP/1.1 404 Not Found

Server: nginx

Date: Thu, 25 May 2017 05:23:03 GMT

Content-Type: text/html

Content-Length: 162

Connection: keep-alive

**2 更改 Nginx 服务的默认用户**

为什么要更改 Nginx 服务的默认用户：就像更改 ssh 的默认 22 端口一样，增加安全性，Nginx 服务的默认用户是 nobody ，我们更改为 nginx

添加 nginx 用户

useradd -s /sbin/nologin -M nginx

更改 Nginx 配置文件

[root@localhost ~]# vim /usr/local/nginx/conf/nginx.conf

worker\_processes 1;

user nginx nginx; # 指定Nginx服务的用户和用户组

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server\_tokens off;

server {

listen 80;

server\_name www.abc.com;

location / {

root html/www;

index index.html index.htm;

}

}

}

重新加载 Nginx

[root@localhost ~]# /usr/local/nginx/sbin/nginx -t

[root@localhost ~]# /usr/local/nginx/sbin/nginx -s reload

验证是否生效

[root@localhost ~]# ps aux | grep nginx

root 8901 0.0 0.1 45036 1784 ? Ss 13:54 0:00 nginx: master process /usr/local/nginx/sbin/nginx

nginx 8909 0.0 0.1 45460 1828 ? S 13:59 0:00 nginx: worker process # Nginx进程的所属用户为nginx

**3 优化 Nginx worker 进程数**

Nginx 有 Master 和 worker 两种进程，**Master 进程用于管理 worker 进程**，worker 进程用于 Nginx 服务。**worker 进程数应该设置为等于 CPU 的核数，高流量并发场合也可以考虑将进程数提高至 CPU 核数 \* 2**

[root@localhost ~]# grep -c processor /proc/cpuinfo # 查看CPU核数



[root@localhost ~]# vim /usr/local/nginx/conf/nginx.conf # 设置worker进程数

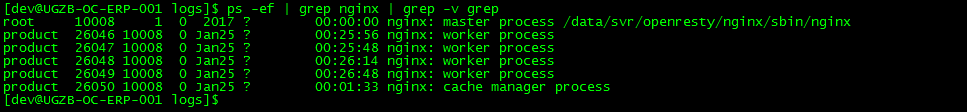
worker\_processes 2;

user nginx nginx;

[root@localhost ~]# /usr/local/nginx/sbin/nginx -t # 重新加载Nginx

[root@localhost ~]# /usr/local/nginx/sbin/nginx -s reload

[root@localhost ~]# ps -ef | grep nginx | grep -v grep # 验证是否为设置的进程数



**4. 绑定 Nginx 进程到不同的 CPU 上**

为什么要绑定 Nginx 进程到不同的 CPU 上 ：默认情况下，Nginx 的多个进程有可能跑在某一个 CPU 或 CPU 的某一核上，导致 Nginx 进程使用硬件的资源不均，

因此绑定 Nginx 进程到不同的 CPU 上是为了充分利用硬件的多 CPU 多核资源的目的。

[root@localhost ~]# grep -c processor /proc/cpuinfo # 查看CPU核数

worker\_processes 2; # 2核CPU的配置

worker\_cpu\_affinity 01 10;

worker\_processes 4; # 4核CPU的配置

worker\_cpu\_affinity 0001 0010 0100 1000;

worker\_processes 8; # 8核CPU的配置

worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 1000000;

[root@localhost ~]# /usr/local/nginx/sbin/nginx -t

[root@localhost ~]# /usr/local/nginx/sbin/nginx -s reload

**5. 优化 Nginx 处理事件模型**

Nginx 的连接处理机制在不同的操作系统会采用不同的 I/O 模型，要根据不同的系统选择不同的事件处理模型，可供选择的事件处理模型有：kqueue 、rtsig 、epoll 、/dev/poll 、select 、poll ，其中 select 和 epoll 都是标准的工作模型，kqueue 和 epoll 是高效的工作模型，不同的是 epoll 用在 Linux 平台上，而 kqueue 用在 BSD 系统中。

(1) 在 Linux 下，Nginx 使用 epoll 的 I/O 多路复用模型

(2) 在 Freebsd 下，Nginx 使用 kqueue 的 I/O 多路复用模型

(3) 在 Solaris 下，Nginx 使用 /dev/poll 方式的 I/O 多路复用模型

(4) 在 Windows 下，Nginx 使用 icop 的 I/O 多路复用模型

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

......

events {

use epoll;

}

......

**6. 优化 Nginx 单个进程允许的最大连接数**

(1) 控制 Nginx 单个进程允许的最大连接数的参数为 worker\_connections ，这个参数要根据服务器性能和内存使用量来调整

(2) 进程的最大连接数受 Linux 系统进程的最大打开文件数限制，只有执行了 "ulimit -HSn 65535" 之后，worker\_connections 才能生效

(3) 连接数包括代理服务器的连接、客户端的连接等，Nginx 总并发连接数 = worker 数量 \* worker\_connections, 总数保持在3w左右

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

worker\_processes 2;

worker\_cpu\_affinity 01 10;

user nginx nginx;

events {

use epoll;

worker\_connections 15000;

}

......

**7. 优化 Nginx worker 进程最大打开文件数**

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

worker\_processes 2;

worker\_cpu\_affinity 01 10;

worker\_rlimit\_nofile 65535; # worker 进程最大打开文件数，可设置为优化后的 ulimit -HSn 的结果

user nginx nginx;

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server\_tokens off;

server {

listen 80;

server\_name www.abc.com;

location / {

root html/www;

index index.html index.htm;

}

}

}

**8. 优化服务器域名的散列表大小**

如下，如果在 server\_name 中配置了一个很长的域名，那么重载 Nginx 时会报错，因此需要使用 server\_names\_hash\_max\_size 来解决域名过长的问题，该参数的作用是设置存放域名的最大散列表的存储的大小，根据 CPU 的一级缓存大小来设置。

server {

listen 80;

server\_name www.abcdefghijklmnopqrst.com; # 配置一个很长的域名

location / {

root html/www;

index index.html index.htm;

}

}

[root@localhost conf]# /usr/local/nginx/sbin/nginx -t # 如果配置的域名很长会出现如下错误

nginx: [emerg] could not build the server\_names\_hash, you should increase server\_names\_hash\_bucket\_size: 64

nginx: configuration file /usr/local/nginx/conf/nginx.conf test failed

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

......

http {

include mime.types;

**server\_names\_hash\_bucket\_size 512; # 配置在 http 区块，默认是 512kb ，一般设置为 cpu 一级缓存的 4-5 倍，一级缓存大小可以用 lscpu 命令查看**

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server\_tokens off;

include vhosts/\*.conf;

}

**9. 开启高效文件传输模式**

(1) sendfile 参数用于开启文件的高效传输模式，该参数实际上是激活了 sendfile() 功能，sendfile() 是作用于两个文件描述符之间的数据拷贝函数，这个拷贝操作是在内核之中的，被称为 "零拷贝" ，sendfile() 比 read 和 write 函数要高效得多，因为 read 和 write 函数要把数据拷贝到应用层再进行操作

(2) tcp\_nopush 参数用于激活 Linux 上的 TCP\_CORK socket 选项，此选项仅仅当开启 sendfile 时才生效，tcp\_nopush 参数可以允许把 http response header 和文件的开始部分放在一个文件里发布，以减少网络报文段的数量

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

......

http {

include mime.types;

server\_names\_hash\_bucket\_size 512;

default\_type application/octet-stream;

**sendfile on; # 开启文件的高效传输模式**

**tcp\_nopush on; # 激活 TCP\_CORK socket 选择**

tcp\_nodelay on; #数据在传输的过程中不进缓存

keepalive\_timeout 65;

server\_tokens off;

include vhosts/\*.conf;

}

**10. 优化 Nginx 连接超时时间**

**<1>. 什么是连接超时**

(1) 举个例子，某饭店请了服务员招待顾客，但是现在饭店不景气，因此要解雇掉一些服务员，这里的服务员就相当于 Nginx 服务建立的连接

(2) 当服务器建立的连接没有接收处理请求时，可以在指定的时间内让它超时自动退出

**<2>.连接超时的作用**

(1) 将无用的连接设置为尽快超时，可以保护服务器的系统资源（CPU、内存、磁盘）

(2) 当连接很多时，及时断掉那些建立好的但又长时间不做事的连接，以减少其占用的服务器资源

(3) 如果黑客攻击，会不断地和服务器建立连接，因此设置连接超时以防止大量消耗服务器的资源

(4) 如果用户请求了动态服务，则 Nginx 就会建立连接，请求 FastCGI 服务以及后端 MySQL 服务，设置连接超时，使得在用户容忍的时间内返回数据

**<3>.连接超时存在的问题**

(1) 服务器建立新连接是要消耗资源的，因此，连接超时时间不宜设置得太短，否则会造成并发很大，导致服务器瞬间无法响应用户的请求

(2) 有些 PHP 站点会希望设置成短连接，因为 PHP 程序建立连接消耗的资源和时间相对要少些

(3) 有些 Java 站点会希望设置成长连接，因为 Java 程序建立连接消耗的资源和时间要多一些，这时由语言的运行机制决定的

<4>.设置连接超时

(1) keepalive\_timeout ：该参数用于设置客户端连接保持会话的超时时间，超过这个时间服务器会关闭该连接

(2) client\_header\_timeout ：该参数用于设置读取客户端请求头数据的超时时间，如果超时客户端还没有发送完整的 header 数据，服务器将返回 "Request time out (408)" 错误

(3) client\_body\_timeout ：该参数用于设置读取客户端请求主体数据的超时时间，如果超时客户端还没有发送完整的主体数据，服务器将返回 "Request time out (408)" 错误

(4) send\_timeout ：用于指定响应客户端的超时时间，如果超过这个时间，客户端没有任何活动，Nginx 将会关闭连接

(5) tcp\_nodelay ：默认情况下当数据发送时，内核并不会马上发送，可能会等待更多的字节组成一个数据包，这样可以提高 I/O 性能，但是，在每次只发送很少字节的业务场景中，使用 tcp\_nodelay 功能，等待时间会比较长

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

......

http {

include mime.types;

server\_names\_hash\_bucket\_size 512;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

tcp\_nodelay on;

client\_header\_timeout 15;

client\_body\_timeout 15;

send\_timeout 25;

include vhosts/\*.conf;

}

**11. 限制上传文件的大小**

client\_max\_body\_size 用于设置最大的允许客户端请求主体的大小，在请求首部中有 "Content-Length" ，如果超过了此配置项，客户端会收到 413 错误，即请求的条目过大

worker\_processes 2;

worker\_cpu\_affinity 01 10;

user nginx nginx;

error\_log logs/error.log error;

events {

use epoll;

worker\_connections 20480;

}

http {

include mime.types;

server\_names\_hash\_bucket\_size 512;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server\_tokens off;

**client\_max\_body\_size 8m; # 设置客户端最大的请求主体大小为8M**

include vhosts/\*.conf;

}

**12.配置 Nginx gzip 压缩**

Nginx gzip 压缩模块提供了压缩文件内容的功能，用户请求的内容在发送到客户端之前，Nginx 服务器会根据一些具体的策略实施压缩，以节约网站出口带宽，同时加快数据传输效率，来提升用户访问体验，需要压缩的对象有 html 、js 、css 、xml 、shtml ，图片和视频尽量不要压缩，因为这些文件大多都是已经压缩过的，如果再压缩可能反而变大，另外，压缩的对象必须大于 1KB，由于压缩算法的特殊原因，极小的文件压缩后可能反而变大

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

......

http {

**gzip on; # 开启压缩功能**

gzip\_min\_length 1k; # 允许压缩的对象的最小字节

gzip\_buffers 4 32k; # 压缩缓冲区大小，表示申请4个单位为16k的内存作为压缩结果的缓存

gzip\_http\_version 1.1; # 压缩版本，用于设置识别HTTP协议版本

gzip\_comp\_level 9; # 压缩级别，1级压缩比最小但处理速度最快，9级压缩比最高但处理速度最慢

gzip\_types text/css text/xml application/javascript; # 允许压缩的媒体类型

gzip\_vary on; # 该选项可以让前端的缓存服务器缓存经过gzip压缩的页面，例如用代理服务器缓存经过Nginx压缩的数据

}

检查压缩：可以用 Google 浏览器按 F12 查看，也可以在 Google 浏览器安装 yslow 插件（yslow.org）

**13.配置 Nginx expires 缓存**

(1) Nginx expires 的功能就是为用户访问的网站内容设定一个过期时间，当用户第一次访问这些内容时，会把这些内容存储在用户浏览器本地，这样用户第二次及以后继续访问该网站时，

浏览器会检查加载已经缓存在用户浏览器本地的内容，就不会去服务器下载了，直到缓存的内容过期或被清除为止

(2) 不希望被缓存的内容：广告图片、网站流量统计工具、更新很频繁的文件

(3) 缓存日期参考：51CTO 缓存 1 周，新浪缓存 15 天，京东缓存 25 年，淘宝缓存 10 年

server {

listen 80;

server\_name www.abc.com abc.com;

root html/www;

location ~ .\*\.(gif|jpg|jpeg|png|bmp|swf|js|css)$ # 缓存的对象

{

**expires 3650d; # 缓存的时间，3650天，**即10年

}

}

**14.优化 Nginx access 日志**

**(1). 配置日志切割**

[root@localhost ~]# vim /usr/local/nginx/conf/cut\_nginx\_log.sh

#!/bin/bash

savepath\_log='/usr/local/clogs'

nglogs='/usr/local/nginx/logs'

mkdir -p $savepath\_log/$(date +%Y)/$(date +%m)

mv $nglogs/access.log $savepath\_log/$(date +%Y)/$(date +%m)/access.$(date +%Y%m%d).log

mv $nglogs/error.log $savepath\_log/$(date +%Y)/$(date +%m)/error.$(date +%Y%m%d).log

kill -USR1 `cat /usr/local/nginx/logs/nginx.pid`

[root@localhost ~]# crontab -e # 每天凌晨0点执行脚本

0 0 \* \* \* /bin/sh /usr/local/nginx/conf/cut\_nginx\_log.sh > /dev/null 2>&1

**(2) 不记录不需要的访问日志**

location ~ .\*\.(js|jpg|JPG|jpeg|JPEG|css|bmp|gif|GIF)$ {

access\_log off;

}

**(3) 设置访问日志的权限**

chown -R root.root /usr/local/nginx/logs

chmod -R 700 /usr/local/nginx/logs

**15. 优化 Nginx 站点目录**

(1). 禁止解析指定目录下的指定程序

location ~ ^/data/.\*\.(php|php5|sh|pl|py)$ {

# 根据实际来禁止哪些目录下的程序，且该配置必须写在 Nginx 解析 PHP 的配置前面

deny all;

}

(2). 禁止访问指定目录

location ~ ^/data/.\*\.(php|php5|sh|pl|py)$ {

# 根据实际来禁止哪些目录下的程序，且该配置必须写在 Nginx 解析 PHP 的配置前面

deny all;

}

(3). 限制哪些 IP 不能访问网站

location ~ ^/wordpress {

**# 相对目录，表示只允许 192.168.1.1 访问网站根目录下的 wordpress 目录**

allow 192.168.1.1/24;

deny all;

}

**16. 配置 Nginx 防盗链**

什么是防盗链：简单地说，就是某些不法网站未经许可，通过在其自身网站程序里非法调用其他网站的资源，然后在自己的网站上显示这些调用的资源，使得被盗链的那一端消耗带宽资源

(1) 根据 HTTP referer 实现防盗链：referer 是 HTTP的一个首部字段，用于指明用户请求的 URL 是从哪个页面通过链接跳转过来的

(2) 根据 cookie 实现防盗链：cookie 是服务器贴在客户端身上的 "标签" ，服务器用它来识别客户端

根据 referer 配置防盗链：

#第一种,匹配后缀

location ~ .\*\.(gif|jpg|jpeg|png|bm|swf|flv|rar|zip|gz|bz2)$ { # 指定需要使用防盗链的媒体资源

access\_log off; # 不记录防盗链的日志

expires 15d; # 设置缓存时间

valid\_referers none blocked \*.test.com \*.abc.com; # 表示这些地址可以访问上面的媒体资源

if ($invalid\_referer) { # 如果地址不如上面指定的地址就返回403

return 403

}

}

#第二种,绑定目录

location /images {

root /web/www/img;

vaild\_referers nono blocked \*.spdir.com \*.spdir.top;

if ($invalid\_referer) {

return 403;

}

}

**17.配置 Nginx 错误页面优雅显示**

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

......

http {

location / {

root html/www;

index index.html index.htm;

error\_page 400 401 402 403 404 405 408 410 412 413 414 415 500 501 502 503 506 = http://www.xxxx.com/error.html;

# 将这些状态码的页面链接到 http://www.xxxx.com/error.html ，也可以单独指定某个状态码的页面，如 error\_page 404 /404.html

}

}

**18. 优化 Nginx 文件权限**

为了保证网站不受木马入侵，所有文件的用户和组都应该为 root ，所有目录的权限是 755 ，所有文件的权限是 644

[root@localhost ~]# chown -R root.root /usr/local/nginx/.... # 根据实际来调整

[root@localhost ~]# chmod 755 /usr/local/nginx/....

[root@localhost ~]# chmod 644 /usr/local/nginx/....

**19.Nginx 防爬虫优化**

我们可以根据客户端的 user-agents 首部字段来阻止指定的爬虫爬取我们的网站

if ($http\_user\_agent ~\* "qihoobot|Baiduspider|Googlebot|Googlebot-Mobile|Googlebot-Image|Mediapartners-Google|Adsbot-Google|Yahoo! Slurp China|YoudaoBot|Sosospider|Sogou spider|Sogou web spider|MSNBot") {

return 403;

}

**20.控制 Nginx 并发连接数**

(1). 限制单个 IP 的并发连接数

[root@localhost ~]# cat /usr/local/nginx/conf/nginx.conf

....

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

limit\_conn\_zone $binary\_remote\_addr zone=addr:10m; # 用于设置共享内存区域，addr 是共享内存区域的名称，10m 表示共享内存区域的大小

server {

listen 80;

server\_name www.abc.com;

location / {

root html/www;

index index.html index.htm;

**limit\_conn addr 1; # 限制单个IP的并发连接数为1**

}

}

}

(2)限制虚拟主机总连接数

...

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

limit\_conn\_zone $server\_name zone=perserver:10m;

server {

listen 80;

server\_name www.abc.com;

location / {

root html/www;

**index index.html index.htm;**

**limit\_conn perserver 2; # 设置虚拟主机连接数为2**

}

}

}

**21. 集群代理优化**

upstream bbs\_com\_pool{ #定义服务器池

ip\_hash; #会话保持(当服务器集群中没有会话池时，且代理的是动态数据就必须写ip\_hash,反之什么也不用写)

#fair #智能分配(第三方，需要下载upstream\_fair模块)根据后端服务器的响应时间来调度

#url\_hash #更具URL的结果来分配请求(每个url定向到同一个服务器,提高后端缓存服务器的效率，本身不支持，需要安装nginx\_hash)

#当算法为ip\_hash不能有 weight backup

server 192.168.10.1:80; #默认weight为1

server 192.168.10.3:80 weight=5; #weight表示权重

server 192.168.10.4:80 down; #down:不参与本次轮询

server 192.168.10.5:80 down backup; #backup:当任何一台主机出现故障，将进行切换替换

server 192.168.10.6:80 max\_fails=3 fail\_timeout=20s; #max\_fails最大失败请求次数(默认为1)，fail\_timeout失败超时时间

server 192.168.10.7:8080;

}

stream{

upstream cluster {

# hash $remote\_addr consistent; //保持 session 不变,四层开启

server 192.168.1.2:80 max\_fails=3 fail\_timeout=30s;

server 192.168.1.3:80 max\_fails=3 fail\_timeout=30s;

}

server {

listen 80;

proxy\_pass cluster;

proxy\_connect\_timeout 1s;

proxy\_timeout 3s;

}

location {

proxy\_next\_upstream http\_500 http\_502 http\_503 error timeout invalid\_header; 当发生其中任何一种错误，将转交给下一个服务器

proxy\_redirect off;

proxy\_set\_header Host &$host; #设置后端服务器的真实地址

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded\_For &proxy\_add\_x\_forwarded\_for;

client\_body\_buffer\_size 128k; #缓冲区大小，本地保存大小

proxy\_connect\_timeout 90; #发起握手等待的响应时间

proxy\_read\_timeout 90; #建立连接后等待后端服务器响应时间(其实是后端等候处理的时间)

proxy\_send\_timeout 90; #给定时间内后端服务器必须响应，否则断开

proxy\_buffer\_size 4k; #proxy缓冲区大小

proxy\_buffers 4 32k; #缓冲区个数和大小

proxy\_busy\_buffers\_size 64k; #系统繁忙时buffer的临时大小，官方要求proxy\_buffer\_size\*2

proxy\_temp\_file\_write\_size 64k; #proxy临时文件的大小

}

}

系统内核参数优化 **vim /etc/sysctl.conf**

net.ipv4.tcp\_syncookies = 1

fs.file-max = 999999

net.ipv4.tcp\_max\_tw\_buckets = 6000

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_tw\_recycle = 1

net.core.somaxconn=262114

net.core.netdev\_max\_backlog=262114

net.ipv4.tcp\_max\_syn\_backlog = 262114

net.ipv4.tcp\_max\_orphans=262114

net.ipv4.tcp\_synack\_retries=1

net.ipv4.tcp\_syn\_retries=1

net.ipv4.tcp\_keepalive\_time = 600

net.ipv4.tcp\_fin\_timeout = 30

net.ipv4.ip\_local\_port\_range = 1024 65000

net.ipv4.tcp\_rmem = 10240 87380 12582912

net.ipv4.tcp\_wmem = 10240 87380 12582912

net.core.netdev\_max\_backlog = 8096

net.core.rmem\_default = 6291456

net.core.wmem\_default = 6291456

net.core.rmem\_max = 12582912

net.core.wmem\_max = 12582912

参数详解

net.ipv4.tcp\_syncookies = 1：选项用于设置开启SYN cookies，当出现SYN等待队列溢出时，启用cookies进行处理;

fs.file-max = 999999：这个参数表示进程（比如一个worker进程）可以同时打开的最大句柄数，这个参数直线限制最大并发连接数，需根据实际情况配置;

net.ipv4.tcp\_max\_tw\_buckets =6000：这个参数用来设定timewait数量，如果超过这个数字，TIME\_WAIT套接字将立刻被清除并打印警告信息。该参数默认为180 000，过多的TIME\_WAIT套接字会使Web服务器变慢;

net.ipv4.tcp\_tw\_recycle=1：这个参数用于设置启用timewait快速回收;

net.ipv4.tcp\_tw\_reuse = 1：这个参数设置为1，表示允许将TIME-WAIT状态的socket重新用于新的TCP连接，这对于服务器来说很有意义，因为服务器上总会有大量TIME-WAIT状态的连接;

net.core.somaxconn=262114 ：选项默认值是128，这个参数用于调节系统同时发起的TCP连接数，在高并发的请求中，默认的值可能会导致链接超时或者重传，因此需要结合高并发请求数来调节此值;

net.core.netdev\_max\_backlog=262114：该选项表示每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许发送到队列数据包的最大数目;

net.ipv4.tcp\_max\_syn\_backlog =262114：这个参数表示TCP三次握手建立阶段接受SYN请求队列的最大长度，默认为1024，将其设置得大一些可以使出现Nginx繁忙来不及accept新连接的情况时，Linux不至于丢失客户端发起的连接请求;

net.ipv4.tcp\_max\_orphans=262114：选项用于设定系统中最多有多少个TCP套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤立链接将立即被复位并输出警告信息。这个限制指示为了防止简单的DOS攻击，不用过分依靠这个限制甚至认为的减小这个值，更多的情况是增加这个值;

net.ipv4.tcp\_synack\_retries=1：内核放弃连接之前发送SYN+ACK包的数量;

net.ipv4.tcp\_syn\_retries=1：内核放弃连接之前发送SYN包的数量;

net.ipv4.tcp\_keepalive\_time = 600：这个参数表示当keepalive启用时，TCP发送keepalive消息的频度。默认是2小时，若将其设置的小一些，可以更快地清理无效的连接;

net.ipv4.tcp\_fin\_timeout = 30：这个参数表示当服务器主动关闭连接时，socket保持在FIN-WAIT-2状态的最大时间;

net.ipv4.ip\_local\_port\_range = 1024 61000：这个参数定义了在UDP和TCP连接中本地（不包括连接的远端）端口的取值范围;

net.ipv4.tcp\_rmem = 10240 87380 12582912：这个参数定义了TCP接受缓存（用于TCP接受滑动窗口）的最小值、默认值、最大值;

net.ipv4.tcp\_wmem = 10240 87380 12582912：这个参数定义了TCP发送缓存（用于TCP发送滑动窗口）的最小值、默认值、最大值;

net.core.netdev\_max\_backlog = 8096：当网卡接受数据包的速度大于内核处理的速度时，会有一个队列保存这些数据包。这个参数表示该队列的最大值;

net.core.rmem\_default = 6291456：这个参数表示内核套接字接受缓存区默认的大小;

net.core.wmem\_default = 6291456：这个参数表示内核套接字发送缓存区默认的大小;

net.core.rmem\_max = 12582912：这个参数表示内核套接字接受缓存区的最大大小;

net.core.wmem\_max = 12582912：这个参数表示内核套接字发送缓存区的最大大小;

net.ipv4.tcp\_syncookies = 1：该参数与性能无关，用于解决TCP的SYN攻击;

**注意：滑动窗口的大小与套接字缓存区会在一定程度上影响并发连接的数目。每个TCP连接都会为维护TCP滑动窗口而消耗内存，这个窗口会根据服务器的处理速度收缩或扩张。 参数net.core.wmem\_max = 12582912的设置，需要平衡物理内存的总大小、Nginx并发处理的最大连接数量而确定。当然，如果仅仅为了提供并发量使服务器不出现Out Of Memory问题而去降低滑动窗口大小，那么并不合适，因为滑动窗过小会影响大数据量的传输速度。net.core.rmem\_default = 6291456、net.core.wmem\_default = 6291456、 net.core.rmem\_max = 12582912和net.core.wmem\_max = 12582912这4个参数的设置需要根据我们的业务特性以及实际的硬件成本来综合考虑。 Nginx并发处理的最大连接量：由nginx.conf中的work\_processes和work\_connections参数决定。**

## 四、Tomcat性能调优

**1 .Tomcat内存优化**

　　Tomcat内存优化主要是对 tomcat 启动参数优化，我们可以在 tomcat 的启动脚本 catalina.sh 中设置 java\_OPTS 参数。

　　JAVA\_OPTS参数说明

　　-server 启用jdk 的 server 版；

　　-Xms java虚拟机初始化时的最小内存；

　　-Xmx java虚拟机可使用的最大内存；

　　-XX: PermSize 内存永久保留区域

　　-XX:MaxPermSize 内存最大永久保留区域

　　服务器参数配置

　　现公司服务器内存一般都可以加到最大2G ，所以可以采取以下配置：

**JAVA\_OPTS=’-Xms1024m -Xmx2048m -XX: PermSize=256M -XX:MaxNewSize=256m -XX:MaxPermSize=256m’**

**2 Tomcat并发优化**

　　1.Tomcat连接相关参数

　　在Tomcat 配置文件 server.xml 中的

　　<Connector port="9027"

　　protocol="HTTP/1.1"

　　maxHttpHeaderSize="8192"

　　minProcessors="100"

　　maxProcessors="1000"

　　acceptCount="1000"

　　redirectPort="8443"

　　disableUploadTimeout="true"/>

　　2.调整连接器connector的并发处理能力

　　1>参数说明

　　maxThreads 客户请求最大线程数

　　minSpareThreads Tomcat初始化时创建的 socket 线程数

　　maxSpareThreads Tomcat连接器的最大空闲 socket 线程数

　　enableLookups 若设为true, 则支持域名解析，可把 ip 地址解析为主机名

　　redirectPort 在需要基于安全通道的场合，把客户请求转发到基于SSL 的 redirectPort 端口

　　acceptAccount 监听端口队列最大数，满了之后客户请求会被拒绝（不能小于maxSpareThreads ）

　　connectionTimeout 连接超时

　　minProcessors 服务器创建时的最小处理线程数

　　maxProcessors 服务器同时最大处理线程数

　　URIEncoding URL统一编码

　　2>Tomcat中的配置示例

　　<Connector port="9027"

　　protocol="HTTP/1.1"

　　maxHttpHeaderSize="8192"

　　maxThreads="1000"

　　minSpareThreads="100"

　　maxSpareThreads="1000"

　　minProcessors="100"

　　maxProcessors="1000"

　　enableLookups="false"

　　URIEncoding="utf-8"

　　acceptCount="1000"

　　redirectPort="8443"

　　disableUploadTimeout="true"/>

**3 Tomcat缓存优化**

　　1>参数说明

　　c ompression 打开压缩功能

　　compressionMinSize 启用压缩的输出内容大小，这里面默认为2KB

　　compressableMimeType 压缩类型

　　connectionTimeout 定义建立客户连接超时的时间. 如果为 -1, 表示不限制建立客户连接的时间

　　2>Tomcat中的配置示例

　　<Connector port="9027"

　　protocol="HTTP/1.1"

　　maxHttpHeaderSize="8192"

　　maxThreads="1000"

　　minSpareThreads="100"

　　maxSpareThreads="1000"

　　minProcessors="100"

　　maxProcessors="1000"

　　enableLookups="false"

　　compression="on"

　　compressionMinSize="2048"

　　compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"

　　connectionTimeout="20000"

　　URIEncoding="utf-8"

　　acceptCount="1000"

　　redirectPort="8443"

　　disableUploadTimeout="true"/>

**4 调优实战**

　　1>旧有的配置

　　<Connector port="9027"

　　protocol="HTTP/1.1"

　　maxHttpHeaderSize="8192"

　　maxThreads="1000"

　　minSpareThreads="25"

　　maxSpareThreads="75"

　　enableLookups="false"

　　compression="on"

　　compressionMinSize="2048"

　　compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"

　　connectionTimeout="20000"

　　URIEncoding="utf-8"

　　acceptCount="200"

　　redirectPort="8443"

　　disableUploadTimeout="true" />

　　2>**发现在访问量达到3 百万多的时候出现性能瓶颈。**

更改后的配置

　　<Connector port="9027"

　　protocol="HTTP/1.1"

　　maxHttpHeaderSize="8192"

　　maxThreads="1000"

　　minSpareThreads="100"

　　maxSpareThreads="1000"

　　minProcessors="100"

　　maxProcessors="1000"

　　enableLookups="false"

　　compression="on"

　　compressionMinSize="2048"

　　compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"

　　connectionTimeout="20000"

　　URIEncoding="utf-8"

　　acceptCount="1000"

　　redirectPort="8443"

　　disableUploadTimeout="true"/>

## 五、应用系统架构优化

**1 网站前端的优化**

**1 建立web 集群**

反向代理服务器工作在HTTP层，类似代理服务器，与普通的代理服务器不同的是，服务器在代理的后端，而不是客户端在代理的后端，可以考虑使用 eginx做反向代理服务器实现 web 服务器的负载均衡。

**2 Css放到head 之间，js放到页面的最下面。**

因为页面的加载是从上往下的顺序执行的，js文件在加载的时候会阻塞页面的执行，所以放到页面最下面可以先看到页面的内容，再去加载相应的js文件。

**3 Css和js文件要合并和压缩。**

因为浏览器的连接数是有限的，如果css 和 js 文件过多会导致浏览器的延迟等待。

**4 使用独立的图片服务器和网站域名。**

可以提高网站的并发能力。

**5 网站的频道用独立的二级域名。**

多个小图标可以考虑放到一个大图片上，然后用css 定位取到相应的内容

**6 CDN 内容分发**

对于像js，css，image, html 等静态内容可以做内容分发，可以把用户定位到最近的服务器上面。

**7 减小cookie 的大小**

因为页面每一个的回传都会把cookie文件提交过去，如果cookie 过大会浪费过多的网络资源。

**8 网站动静文件分离**

把静态文件放到一个独立的服务器上，可以考虑用eginx web服务器，eginx 静态文件的处理速度非常快，不是iis 所能比的。

**9 页面内容进行压缩**

页面内容要进行压缩，较少网络传输的压力，可以考虑用Gzip压缩。

**10 对页面进行缓存**

因内容而异，如果是不经常改变的内容，设置缓存的时间要长一些，从而减少服务器的压力。

**11 不要使用iframe 控件**

iframe 会阻碍页面的执行，最好不要使用，况且也不利于搜索引擎的收录。

**12 页面的html标签不要放到一个大标签里面**

因为一个标签在结束之前是不会呈现内容的，如果都把页面的内容都到一个表格中，只有到</table> 表格中的内容才会显示出来。

**2 后台的优化**

**1 使用缓存**

建立多级别的缓存策略，可以考虑先单机缓存，然后再考虑分布式缓存。分布式缓存可以考虑使用 memcached 或者 redis，redis 提供了更多的存储类型，并支持数据写入磁盘的功能。

**2 网站考虑使用消息队列**

消息队列可以对系统进行解耦，并可提高系统的处理能力，减少数据库的压力。如果是微软路线可以考虑用MSMQ。

**3 业务处理层不要使用单例模式**

因为单例模式的话，会阻碍多线程和多核系统的处理，降低系统的吞吐量和处理能力。

**4 Stringbulder 代替 string**

如果字符串内容多的话，考虑使用Stringbulder 增加字符串的处理能力。

**3 数据库的优化**

**1 数据库读写分离**

这种方式是指利用数据库的复制或镜像功能，同时在多台数据库上保存相同的数据，并且将读操作和写操作分开，写操作集中在一台主数据库上，读操作集中在多台 从数据库上，对于读取比写更多的站点适合使用这种方式。

**2 使用分区表**

表进行分区后，逻辑上表仍然是一张完整的表，只是将表中的数据在物理上存放到多个表空间(物理文件上)，这样查询数据时，不至于每次都扫描整张表。提高数据库的查询性能。

**3 分库分表**

划分不同的业务模块放到不同的数据库上，如 订单库、会员库、商品库等。

**4 建立合适的索引**

在经常查询或分组的列上，建立相应的索引来提高查询性能。

## 六、去中心化调优

**1 去中心化”是一种现象或结构**

其只能出现在拥有众多用户或众多节点的系统中，每个用户都可连接并影响其他节点。**通俗地讲，就是每个人都是中心，每个人都可以连接并影响其他节点，这种扁平化、开源化、平等化的现象或结构，称之为“去中心化”。**

同时“去中心化”是区块链的典型特征之一，其使用分布式储存与算力，整个网络节点的权利与义务相同，系统中数据本质为全网节点共同维护，从而区块链不再依靠于中央处理节点，实现数据的分布式存储、记录与更新。而每个区块链都遵循统一规则，该规则基于密码算法而不是信用证书，且数据更新过程都需用户批准，由此奠定区块链不需要中介与信任机构背书。

**2 去中心化的特点**

去中心化首先体现在多样化上，在网络世界不再是有几个门户网站说了算，各种各样的网站开始有了自己的声音，表达不同的选择，不同的爱好，这些网站分布在网络世界的各个角落里张扬着个性。去中心化其次体现在人的中心化上，去内容中心化成为趋势，人成为决定网站生存的关键力量。以缺乏互动的个别人建站变成了以圈子的形式来聚合人才贡献自己的智慧，这是一个巨大的变革。即用户为本，人性化。

**3 去中心化的内容**

去中心化是互联网发展过程中形成的社会化关系形态和内容产生形态，是相对于“中心化”而言的新型网络内容生产过程。相对于早期的互联网（Web1.0）时代，**今天的网络（Web2.0）内容不再是由专业网站或特定人群所产生，而是由全体网民共同参与、权级平等的共同创造的结果。**任何人，都可以在网络上表达自己的观点或创造原创的内容，共同生产信息。随着网络服务形态的多元化，去中心化网络模型越来越清晰，也越来越成为可能。Web2.0兴起后，Wikipedia、Flickr、Blogger等网络服务商所提供的服务都是去中心化的，任何参与者，均可提交内容，网民共同进行内容协同创作或贡献。之后随着更多简单易用的去中心化网络服务的出现，Web2.0的特点越发明显，例如Twitter、Facebook等更加适合普通网民的服务的诞生，使得为互联网生产或贡献内容更加简便、更加多元化，从而提升了网民参与贡献的积极性、降低了生产内容的门槛。最终使得每一个网民均成为了一个微小且独立的信息提供商，使得互联网更加扁平、内容生产更加多元化。