# Tomcat、Nginx优化

## Tomcat优化

### Linux安装APR

众所周知，Tomcat的I/O有三种，BIO、NIO、AIO（APR），性能依次递升。BIO是默认的，NIO改起来也简单：

|  |
| --- |
| <Connector port="8080" protocol="**org.apache.coyote.http11.Http11NioProtocol**"  connectionTimeout="20000"  redirectPort="8443" /> |

生产环境中应该用APR，具体的原理分析就不多讲，直接实操。

我的环境是CentOS6.8+JDK8+Tomcat7。

|  |
| --- |
| wget http://mirrors.shuosc.org/apache/apr/apr-1.6.3.tar.gz  wget http://mirrors.shuosc.org/apache/apr/apr-util-1.6.1.tar.gz |

安装apr

|  |
| --- |
| tar zxvf apr-1.6.3.tar.gz  cd apr-1.6.3  ./configure --prefix=/usr/local/apr ;make;make install |

安装apt-util，make的时候会报错：

xml/apr\_xml.c:35:19: 错误：expat.h：没有那个文件或目录

应该是少了expat-devel

|  |
| --- |
| yum install expat-devel |

安装apr-util

|  |
| --- |
| tar zxvf apr-util-1.6.1.tar.gz  cd apr-util-1.6.1  ./configure --prefix=/usr/local/apr/apr-util --with-apr=/usr/local/apr;make;make install |

安装tomcat-native，make的时候可能会报错：

Your version of OpenSSL is not compatible with this version of tcnative

信息：

Found OPENSSL\_VERSION\_NUMBER 0x1000105f (OpenSSL 1.0.1e 11 Feb 2013)

Require OPENSSL\_VERSION\_NUMBER 0x1000200f or greater (1.0.2)

人家要求OpenSSL版本是1.0.2，我的是1.0.1e

要升级OpenSSL

|  |
| --- |
| cd /usr/local/src  wget https://www.openssl.org/source/openssl-1.0.2m.tar.gz  tar -xzxf openssl-1.0.2m.tar.gz  ./config --prefix=/usr/local/openssl -fPIC  make  make install  mv /usr/bin/openssl ~  ln -s /usr/local/openssl/bin/openssl /usr/bin/openssl  openssl version |

版本应该是新的1.0.2m了

安装tomcat-native

|  |
| --- |
| tar zxvf tomcat-native.tar.gz  cd /home/leo/tomcat7/bin/tomcat-native-1.2.12-src/native  ./configure --with-apr=/usr/local/apr --with-ssl=/usr/local/openssl;make;make install |

安装完毕后，去启动Tomcat，很有可能会出现以下日志：

信息: The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: /usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib

这时候去/usr/local/apr/lib目录执行

|  |
| --- |
| ln -s /usr/local/apr/lib/libtcnative-1.so /usr/lib64/libtcnative-1.so |

这时候再去启动Tomcat，会出现

Loaded APR based Apache Tomcat Native library 1.2.12 using APR version 1.6.3.

搞定！

### 并发优化

|  |
| --- |
| <Connector port="8080" protocol="**org.apache.coyote.http11.Http11NioProtocol**"  connectionTimeout="20000"  redirectPort="8443" /> |

在这里可以对并发做一些优化：

**URIEncoding**：URI编码字符集，默认是ISO-8859-1，一般都改成UTF-8。

**maxThreads**：请求最大线程数，默认值是200。

**minSpareThreads**：Tomcat启动时，默认创建的线程数，并且会确保以后至少会有这些线程可用，默认是4。

**maxSpareThreads**：允许存在的空闲线程的最大数，默认50。

注意：大家在查资料的时候经常会看到minProcessors、maxProcessors参数，这俩在Tomcat4以上的版本就不再用了。

**connectionTimeout**：连接超时时间，默认60000毫秒，1分钟。

**acceptCount**：当所有的线程（maxThreads）都在运行的时候，再进来的请求都放在队列中，队列的数目就是这个值，如果超过这个值了，请求就会被拒绝。默认10。

**compression**="on"：开启压缩。

**compressionMinSize**="xxxx"：输出内容大小，默认是2KB，小于这个值就不压缩。

**compressableMimeType**="text/html等"：需要压缩的类型。

多说一句，压缩其实就是用CPU资源换取带宽资源，在实际生产环境下，要按照实际情况选择，多做测试。

### 内存优化

在catalina.sh文件注释文字之后第一行进行JVM的设置。

JAVA\_OPTS='-Dfile.encoding=UTF-8 -server -Xms1024m -Xmx1024m -XX:PermSize=128m -XX:MaxPermSize=256m'

-Xms：JVM初始化时的内存大小，默认是物理内存1/64。

-Xmx:JVM最大内存，默认是物理内存的1/4.

这两个应该设置为相同数，避免每次GC后再去调整，白费时间。

-XX:PermSize：Perm内存大小。

-XX:MaxPermSize：Perm内存最大空间，默认32M。

PermGen是用来存放Class和Meta信息的，如果出现“java.lang.OutOfMemoryError:PermGen space”这种异常，就说明我们的WEB项目类、jar过多。

其他参数：

-XX:NewSize：年轻代初始化所占内存大小。

-XX:MaxNewSize：年轻代所占内存最大空间。

二者应设置为相同书，避免再调整。

-XX:+UseParallelOldGC：老年代使用并行收集算法。

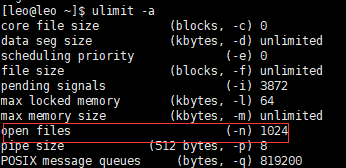
-XX:ParallelGCThreads：并行收集线程数，最好与处理器数目相同。

## Nginx优化

查看一下Nginx所在用户下，最大允许打开文件的数量。

|  |
| --- |
| ulimit -a |

结果中“open files”就是。



临时增加的办法：

|  |
| --- |
| ulimit -n 数字 |

永久增加的办法：

|  |
| --- |
| vim /etc/security/limits.conf  用户名 soft nofile 数字  用户名 hard nofile 数字 |

多说两句，这个其实应该算是Linux系统调优了，有几个应该遵循的基本原则：

1. 别用root用户部署应用。这个很好理解吧，万一线程被应用全占了，到时候我们都登录不了服务器。
2. 做好规划。一台服务器的内存大小，决定了线程的上限。准备放多少应用，就决定了每个应用所占的线程数上限。假设服务器能支撑的线程数是1024，如果我们确定本机只放一个应用，那么在配置的时候就可以将1024都分给它。千万别闷头往里加应用，20个应用，每个线程都分配100个，这样系统肯定会出问题。

修改Nginx的配置文件：

**worker\_rlimit\_nofile 数字;** 应该与ulimit –n相同。

**worker\_processes CPU内核数;**

**keepalive\_timeout 默认1分钟;**一般来说1万个空连接占内存2~3M，如果我们的网站流量比较大，可以将时间缩小一些。

在**events**模块中：

**use epoll;**

**worker\_connections 数字;**每个线程运行的连接数。

通过worker\_processes\*worker\_connections我们可以得出Nginx服务器的最大连接数。

如果我们将Nginx作为反向代理，则需要再除以4。主要是因为浏览器默认开2个连接，Nginx又用File Descriptors（文件描述符）建立2个连接，一个指向客户端，一个指向代理服务器。

**在http模块中：**

**sendfile on;** 直接将数据从磁盘读到OS缓存，因为是在内核中完成这个操作，所以效率更高。

**tcp\_nopush on;**在一个数据包里发送所有头文件。

**multi\_accept on;** Nginx在已经得到一个新连接的通知时，接收尽可能更多的连接。

**access\_log off;**如果我们不需要请求信息，或者用其他方式来搜集信息，关闭它，减少I/O。

**error\_log 文件路径 级别;**级别从最详到最简[debug | info | notice | warn | error | crit]。越简单，I/O越少。

还有GZIP压缩的相关优化，还是那句话：压缩是用CPU资源换网络资源，根据实际情况调优吧。