# Tomcat

## 性能优化

参考链接：

<https://www.cnblogs.com/zhuawang/p/5213192.html>

**一:Tomcat内存优化,启动时告诉JVM我要一块大内存(调优内存是最直接的方式)**

Windows 下的catalina.bat

Linux 下的catalina.sh 如:

JAVA\_OPTS='-Xms256m -Xmx512m'

-Xms<size> JVM初始化堆的大小

-Xmx<size> JVM堆的最大值 实际参数大小根据服务器配置或者项目具体设置.

**二:Tomcat 线程优化** 在server.xml中 如:

<Connector port="80" protocol="HTTP/1.1" maxThreads="600" minSpareThreads="100" maxSpareThreads="500" acceptCount="700"

connectionTimeout="20000" />

maxThreads="X" 表示最多同时处理X个连接

minSpareThreads="X" 初始化X个连接

maxSpareThreads="X" 表示如果最多可以有X个线程，一旦超过X个,则会关闭不在需要的线程

acceptCount="X" 当同时连接的人数达到maxThreads时,还可以排队,队列大小为X.超过X就不处理

**三:Tomcat IO优化**

1:同步阻塞IO（JAVA BIO） 同步并阻塞，服务器实现模式为一个连接一个线程(one connection one thread 想想都觉得恐怖,线程可是非常宝贵的资源)，当然可以通过线程池机制改善.

2:JAVA NIO:又分为同步非阻塞IO,异步阻塞IO 与BIO最大的区别one request one thread.可以复用同一个线程处理多个connection(多路复用).

3:,异步非阻塞IO(Java NIO2又叫AIO) 主要与NIO的区别主要是操作系统的底层区别.可以做个比喻:比作快递，NIO就是网购后要自己到官网查下快递是否已经到了(可能是多次)，然后自己去取快递；AIO就是快递员送货上门了(不用关注快递进度)。

BIO方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4以前的唯一选择，但程序直观简单易理解.

NIO方式适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4开始支持.

AIO方式使用于连接数目多且连接比较长（重操作）的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂，JDK7开始支持.

在server.xml中

<Connector port="80" protocol="org.apache.coyote.http11.Http11NioProtocol"

connectionTimeout="20000"

URIEncoding="UTF-8"

useBodyEncodingForURI="true"

enableLookups="false"

redirectPort="8443" />

实现对Tomcat的IO切换.

## 配置

参考链接：

<https://www.cnblogs.com/xuwc/p/8523681.html>

配置文件  
 server.xml：主要的配置文件。  
 web.xml：缺省的web app配置，WEB-INF/web.xml会覆盖该配置。  
 context.xml：不清楚跟server.xml里面的context是否有关系。  
   
 server.xml配置  
 server标签  
 port：指定一个端口，这个端口负责监听关闭tomcat的请求。  
 shutdown：指定向端口发送的命令字符串。  
   
 service标签  
 name：指定service的名字。  
   
 Connector(表示客户端和service之间的连接)标签

port：指定服务器端要创建的端口号，并在这个端口监听来自客户端的请求。  
 minProcessors：服务器启动时创建的处理请求的线程数。  
 maxProcessors：最大可以创建的处理请求的线程数。  
 enableLookups：如果为true，则可以通过调用request.getRemoteHost()进行DNS查询来得到远程客户端的实际主机名，若为false则不进行DNS查询，而是返回其ip地址。  
 redirectPort：指定服务器正在处理http请求时收到了一个SSL传输请求后重定向的端口号。  
 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。  
 connectionTimeout：指定超时的时间数(以毫秒为单位)。  
   
 Engine(表示指定service中的请求处理机，接收和处理来自Connector的请求)标签  
 defaultHost：指定缺省的处理请求的主机名，它至少与其中的一个host元素的name属性值是一样的。  
   
 Context(表示一个web应用程序，通常为WAR文件，关于WAR的具体信息见servlet规范)标签  
 docBase：该web应用的文档基准目录（Document Base，也称为Context Root），或者是WAR文件的路径。可以使用绝对路径，也可以使用相对于context所属的Host的appBase路径。  
 path：表示此web应用程序的url的前缀，这样请求的url为http://localhost:8080/path/\*\*\*\*。  
 **reloadable：这个属性非常重要，如果为true，则tomcat会自动检测应用程序的/WEB-INF/lib和/WEB-INF/classes目录的变化，自动装载新的应用程序，我们可以在不重起tomcat的情况下改变应用程序。**  
 useNaming：如果希望Catalina为该web应用使能一个JNDI InitialContext对象，设为true。该InitialialContext符合J2EE平台的约定，缺省值为true。  
 workDir：Context提供的临时目录的路径，用于servlet的临时读/写。利用javax.servlet.context.tempdir属性，servlet可以访问该目录。如果没有指定，使用$CATALINA\_HOME/work下一个合适的目录。  
 swallowOutput：如果该值为true，System.out和System.err的输出被重定向到web应用的logger。如果没有指定，缺省值为false  
 debug：与这个Engine关联的Logger记录的调试信息的详细程度。数字越大，输出越详细。如果没有指定，缺省为0。  
   
 host(表示一个虚拟主机)标签  
 name：指定主机名。  
 appBase：应用程序基本目录，即存放应用程序的目录。  
 unpackWARs：如果为true，则tomcat会自动将WAR文件解压，否则不解压，直接从WAR文件中运行应用程序。  
   
 Logger(表示日志，调试和错误信息)标签  
 className：指定logger使用的类名，此类必须实现org.apache.catalina.Logger接口。  
 prefix：指定log文件的前缀。  
 suffix：指定log文件的后缀。  
 timestamp：如果为true，则log文件名中要加入时间，如下例:localhost\_log.2001-10-04.txt。  
   
 Realm(表示存放用户名，密码及role的数据库)标签  
 className：指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口。  
   
 Valve(功能与Logger差不多，其prefix和suffix属性解释和Logger  中的一样)标签  
 className：指定Valve使用的类名，如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息。  
 directory：指定log文件存放的位置。  
 pattern：有两个值，common方式记录远程主机名或ip地址，用户名，日期，第一行请求的字符串，HTTP响应代码，发送的字节数。combined方式比common方式记录的值更多。