前言

在使用tomcat时，经常会遇到连接数、线程数之类的配置问题，要真正理解这些概念，必须先了解Tomcat的连接器（Connector）。

在前面的文章 详解Tomcat配置文件server.xml 中写到过：Connector的主要功能，是接收连接请求，创建Request和Response对象用于和请求端交换数据；然后分配线程让Engine（也就是Servlet容器）来处理这个请求，并把产生的Request和Response对象传给Engine。当Engine处理完请求后，也会通过Connector将响应返回给客户端。

可以说，Servlet容器处理请求，是需要Connector进行调度和控制的，Connector是Tomcat处理请求的主干，因此Connector的配置和使用对Tomcat的性能有着重要的影响。这篇文章将从Connector入手，讨论一些与Connector有关的重要问题，包括NIO/BIO模式、线程池、连接数等。

根据协议的不同，Connector可以分为HTTP Connector、AJP Connector等，本文只讨论HTTP Connector。

一、Nio、Bio、APR

1、Connector的protocol

Connector在处理HTTP请求时，会使用不同的protocol。不同的Tomcat版本支持的protocol不同，其中最典型的protocol包括BIO、NIO和APR（Tomcat7中支持这3种，Tomcat8增加了对NIO2的支持，而到了Tomcat8.5和Tomcat9.0，则去掉了对BIO的支持）。

BIO是Blocking IO，顾名思义是阻塞的IO；NIO是Non-blocking IO，则是非阻塞的IO。而APR是Apache Portable Runtime，是Apache可移植运行库，利用本地库可以实现高可扩展性、高性能；Apr是在Tomcat上运行高并发应用的首选模式，但是需要安装apr、apr-utils、tomcat-native等包。

2、如何指定protocol

Connector使用哪种protocol，可以通过<connector>元素中的protocol属性进行指定，也可以使用默认值。

指定的protocol取值及对应的协议如下：

* HTTP/1.1：默认值，使用的协议与Tomcat版本有关
* org.apache.coyote.http11.Http11Protocol：BIO
* org.apache.coyote.http11.Http11NioProtocol：NIO
* org.apache.coyote.http11.Http11Nio2Protocol：NIO2
* org.apache.coyote.http11.Http11AprProtocol：APR

如果没有指定protocol，则使用默认值HTTP/1.1，其含义如下：在Tomcat7中，自动选取使用BIO或APR（如果找到APR需要的本地库，则使用APR，否则使用BIO）；在Tomcat8中，自动选取使用NIO或APR（如果找到APR需要的本地库，则使用APR，否则使用NIO）。

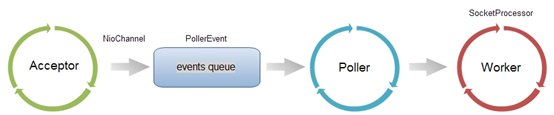
3、BIO/NIO有何不同

无论是BIO，还是NIO，Connector处理请求的大致流程是一样的：

在accept队列中接收连接（当客户端向服务器发送请求时，如果客户端与OS完成三次握手建立了连接，则OS将该连接放入accept队列）；在连接中获取请求的数据，生成request；调用servlet容器处理请求；返回response。为了便于后面的说明，首先明确一下连接与请求的关系：连接是TCP层面的（传输层），对应socket；请求是HTTP层面的（应用层），必须依赖于TCP的连接实现；一个TCP连接中可能传输多个HTTP请求。

在BIO实现的Connector中，处理请求的主要实体是JIoEndpoint对象。JIoEndpoint维护了Acceptor和Worker：Acceptor接收socket，然后从Worker线程池中找出空闲的线程处理socket，如果worker线程池没有空闲线程，则Acceptor将阻塞。其中Worker是Tomcat自带的线程池，如果通过<Executor>配置了其他线程池，原理与Worker类似。

在NIO实现的Connector中，处理请求的主要实体是NIoEndpoint对象。NIoEndpoint中除了包含Acceptor和Worker外，还是用了Poller，处理流程如下图所示（图片来源：http://gearever.iteye.com/blog/1844203）。

[](http://www.importnew.com/27309.html/1174710-20171108203711513-1728828893)

Acceptor接收socket后，不是直接使用Worker中的线程处理请求，而是先将请求发送给了Poller，而Poller是实现NIO的关键。Acceptor向Poller发送请求通过队列实现，使用了典型的生产者-消费者模式。在Poller中，维护了一个Selector对象；当Poller从队列中取出socket后，注册到该Selector中；然后通过遍历Selector，找出其中可读的socket，并使用Worker中的线程处理相应请求。与BIO类似，Worker也可以被自定义的线程池代替。

通过上述过程可以看出，在NIoEndpoint处理请求的过程中，无论是Acceptor接收socket，还是线程处理请求，使用的仍然是阻塞方式；但在“读取socket并交给Worker中的线程”的这个过程中，使用非阻塞的NIO实现，这是NIO模式与BIO模式的最主要区别（其他区别对性能影响较小，暂时略去不提）。而这个区别，在并发量较大的情形下可以带来Tomcat效率的显著提升：

目前大多数HTTP请求使用的是长连接（HTTP/1.1默认keep-alive为true），而长连接意味着，一个TCP的socket在当前请求结束后，如果没有新的请求到来，socket不会立马释放，而是等timeout后再释放。如果使用BIO，“读取socket并交给Worker中的线程”这个过程是阻塞的，也就意味着在socket等待下一个请求或等待释放的过程中，处理这个socket的工作线程会一直被占用，无法释放；因此Tomcat可以同时处理的socket数目不能超过最大线程数，性能受到了极大限制。而使用NIO，“读取socket并交给Worker中的线程”这个过程是非阻塞的，当socket在等待下一个请求或等待释放时，并不会占用工作线程，因此Tomcat可以同时处理的socket数目远大于最大线程数，并发性能大大提高。

二、3个参数：acceptCount、maxConnections、maxThreads

再回顾一下Tomcat处理请求的过程：在accept队列中接收连接（当客户端向服务器发送请求时，如果客户端与OS完成三次握手建立了连接，则OS将该连接放入accept队列）；在连接中获取请求的数据，生成request；调用servlet容器处理请求；返回response。

相对应的，Connector中的几个参数功能如下：

1、acceptCount

accept队列的长度；当accept队列中连接的个数达到acceptCount时，队列满，进来的请求一律被拒绝。默认值是100。

2、maxConnections

Tomcat在任意时刻接收和处理的最大连接数。当Tomcat接收的连接数达到maxConnections时，Acceptor线程不会读取accept队列中的连接；这时accept队列中的线程会一直阻塞着，直到Tomcat接收的连接数小于maxConnections。如果设置为-1，则连接数不受限制。

默认值与连接器使用的协议有关：NIO的默认值是10000，APR/native的默认值是8192，而BIO的默认值为maxThreads（如果配置了Executor，则默认值是Executor的maxThreads）。

在windows下，APR/native的maxConnections值会自动调整为设置值以下最大的1024的整数倍；如设置为2000，则最大值实际是1024。

3、maxThreads

请求处理线程的最大数量。默认值是200（Tomcat7和8都是的）。如果该Connector绑定了Executor，这个值会被忽略，因为该Connector将使用绑定的Executor，而不是内置的线程池来执行任务。

maxThreads规定的是最大的线程数目，并不是实际running的CPU数量；实际上，maxThreads的大小比CPU核心数量要大得多。这是因为，处理请求的线程真正用于计算的时间可能很少，大多数时间可能在阻塞，如等待数据库返回数据、等待硬盘读写数据等。因此，在某一时刻，只有少数的线程真正的在使用物理CPU，大多数线程都在等待；因此线程数远大于物理核心数才是合理的。

换句话说，Tomcat通过使用比CPU核心数量多得多的线程数，可以使CPU忙碌起来，大大提高CPU的利用率。

4、参数设置

（1）maxThreads的设置既与应用的特点有关，也与服务器的CPU核心数量有关。通过前面介绍可以知道，maxThreads数量应该远大于CPU核心数量；而且CPU核心数越大，maxThreads应该越大；应用中CPU越不密集（IO越密集），maxThreads应该越大，以便能够充分利用CPU。当然，maxThreads的值并不是越大越好，如果maxThreads过大，那么CPU会花费大量的时间用于线程的切换，整体效率会降低。

（2）maxConnections的设置与Tomcat的运行模式有关。如果tomcat使用的是BIO，那么maxConnections的值应该与maxThreads一致；如果tomcat使用的是NIO，那么类似于Tomcat的默认值，maxConnections值应该远大于maxThreads。

（3）通过前面的介绍可以知道，虽然tomcat同时可以处理的连接数目是maxConnections，但服务器中可以同时接收的连接数为maxConnections+acceptCount 。acceptCount的设置，与应用在连接过高情况下希望做出什么反应有关系。如果设置过大，后面进入的请求等待时间会很长；如果设置过小，后面进入的请求立马返回connection refused。

三、线程池Executor

Executor元素代表Tomcat中的线程池，可以由其他组件共享使用；要使用该线程池，组件需要通过executor属性指定该线程池。

Executor是Service元素的内嵌元素。一般来说，使用线程池的是Connector组件；为了使Connector能使用线程池，Executor元素应该放在Connector前面。Executor与Connector的配置举例如下：

|  |  |
| --- | --- |
| 1  2 | <Executor name="tomcatThreadPool" namePrefix ="catalina-exec-" maxThreads="150" minSpareThreads="4" />  <Connector executor="tomcatThreadPool" port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" acceptCount="1000" /> |

Executor的主要属性包括：

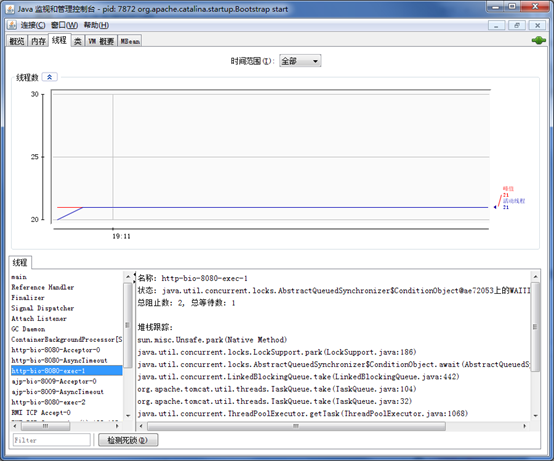
* name：该线程池的标记
* maxThreads：线程池中最大活跃线程数，默认值200（Tomcat7和8都是）
* minSpareThreads：线程池中保持的最小线程数，最小值是25
* maxIdleTime：线程空闲的最大时间，当空闲超过该值时关闭线程（除非线程数小于minSpareThreads），单位是ms，默认值60000（1分钟）
* daemon：是否后台线程，默认值true
* threadPriority：线程优先级，默认值5
* namePrefix：线程名字的前缀，线程池中线程名字为：namePrefix+线程编号

四、查看当前状态

上面介绍了Tomcat连接数、线程数的概念以及如何设置，下面说明如何查看服务器中的连接数和线程数。

查看服务器的状态，大致分为两种方案：（1）使用现成的工具，（2）直接使用Linux的命令查看。

现成的工具，如JDK自带的jconsole工具可以方便的查看线程信息（此外还可以查看CPU、内存、类、JVM基本信息等），Tomcat自带的manager，收费工具New Relic等。下图是jconsole查看线程信息的界面：

[](http://www.importnew.com/27309.html/1174710-20171108204908856-1788241471)

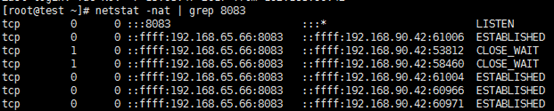
下面说一下如何通过Linux命令行，查看服务器中的连接数和线程数。

1、连接数

假设Tomcat接收http请求的端口是8083，则可以使用如下语句查看连接情况：

|  |  |
| --- | --- |
| 1 | netstat –nat | grep 8083 |

结果如下所示：

[](http://www.importnew.com/27309.html/1174710-20171108205043325-1785419465)

可以看出，有一个连接处于listen状态，监听请求；除此之外，还有4个已经建立的连接（ESTABLISHED）和2个等待关闭的连接（CLOSE\_WAIT）。

2、线程

ps命令可以查看进程状态，如执行如下命令：

|  |  |
| --- | --- |
| 1 | ps –e | grep java |

结果如下图：

[http://incdn1.b0.upaiyun.com/2017/11/9a67893beceea416d01e148b290f5896.png](http://www.importnew.com/27309.html/1174710-20171108205102403-750285056)

可以看到，只打印了一个进程的信息；27989是线程id，java是指执行的java命令。这是因为启动一个tomcat，内部所有的工作都在这一个进程里完成，包括主线程、垃圾回收线程、Acceptor线程、请求处理线程等等。

通过如下命令，可以看到该进程内有多少个线程；其中，nlwp含义是number of light-weight process。

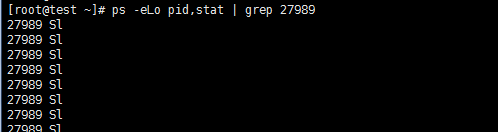
|  |  |
| --- | --- |
| 1 | ps –o nlwp 27989 |

[http://incdn1.b0.upaiyun.com/2017/11/54fb0f99030ec3b630bcf4d1c7bcef8d.png](http://www.importnew.com/27309.html/1174710-20171108205109356-1650242943)

可以看到，该进程内部有73个线程；但是73并没有排除处于idle状态的线程。要想获得真正在running的线程数量，可以通过以下语句完成：

|  |  |
| --- | --- |
| 1 | ps -eLo pid ,stat | grep 27989 | grep running | wc -l |

其中ps -eLo pid ,stat可以找出所有线程，并打印其所在的进程号和线程当前的状态；两个grep命令分别筛选进程号和线程状态；wc统计个数。其中，ps -eLo pid ,stat | grep 27989输出的结果如下：

[](http://www.importnew.com/27309.html/1174710-20171108205116559-949248660)

图中只截图了部分结果；Sl表示大多数线程都处于空闲状态。