

35 | 如何监控Tomcat的性能？

2019-08-01 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



讲述：李号双

时长 09:04 大小 8.31M



专栏上一期我们分析了 JVM GC 的基本原理以及监控和分析工具，今天我们接着来聊如何监控 Tomcat 的各种指标，因为只有我们掌握了这些指标和信息，才能对 Tomcat 内部发生的事情一目了然，让我们明白系统的瓶颈在哪里，进而做出调优的决策。

在今天的文章里，我们首先来看看到底都需要监控 Tomcat 哪些关键指标，接着来具体学习如何通过 JConsole 来监控它们。如果系统没有暴露 JMX 接口，我们还可以通过命令行来查看 Tomcat 的性能指标。

Web 应用的响应时间是我们关注的一个重点，最后我们通过一个实战案例，来看看 Web 应用的下游服务响应时间比较长的情况下，Tomcat 的各项指标是什么样子的。

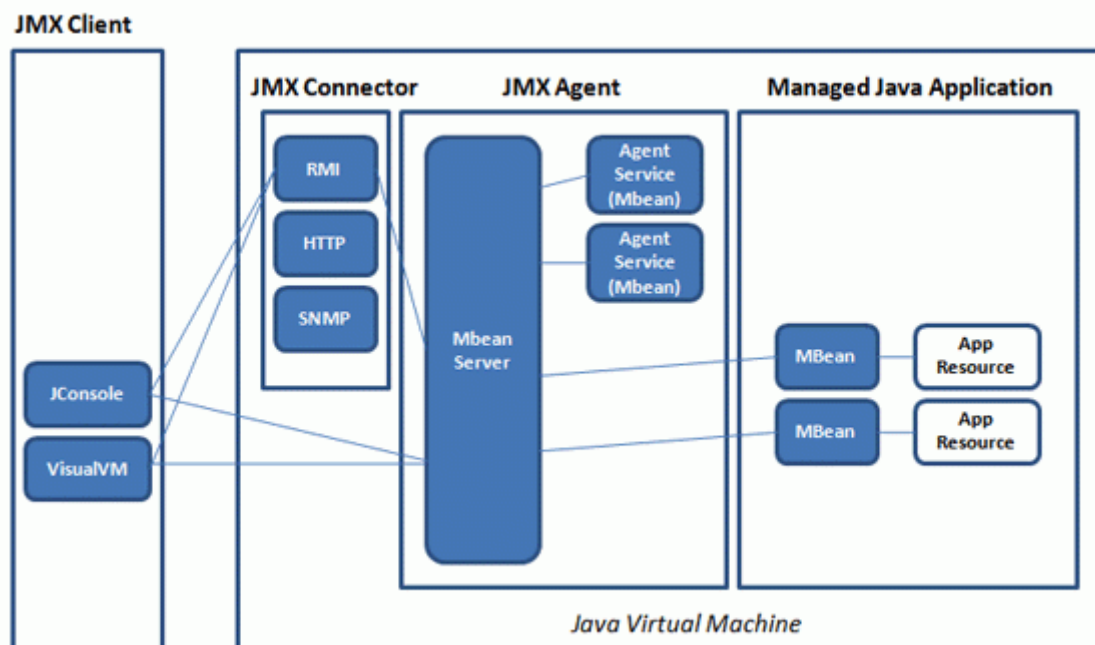
Tomcat 的关键指标

Tomcat 的关键指标有**吞吐量、响应时间、错误数、线程池、CPU 以及 JVM 内存**。

我来简单介绍一下这些指标背后的意义。其中前三个指标是我们最关心的业务指标，Tomcat 作为服务器，就是要能够又快又好地处理请求，因此吞吐量要大、响应时间要短，并且错误数要少。

而后面三个指标是跟系统资源有关的，当某个资源出现瓶颈就会影响前面的业务指标，比如线程池中的线程数量不足会影响吞吐量和响应时间；但是线程数太多会耗费大量 CPU，也会影响吞吐量；当内存不足时会触发频繁地 GC，耗费 CPU，最后也会反映到业务指标上来。


那如何监控这些指标呢？Tomcat 可以通过 JMX 将上述指标暴露出来的。JMX (Java Management Extensions，即 Java 管理扩展) 是一个为应用程序、设备、系统等植入监控管理功能的框架。JMX 使用管理 MBean 来监控业务资源，这些 MBean 在 JMX MBean 服务器上注册，代表 JVM 中运行的应用程序或服务。每个 MBean 都有一个属性列表。JMX 客户端可以连接到 MBean Server 来读写 MBean 的属性值。你可以通过下面这张图来理解一下 JMX 的工作原理：



Tomcat 定义了一系列 MBean 来对外暴露系统状态，接下来我们来看看如何通过 JConsole 来监控这些指标。

通过 JConsole 监控 Tomcat

首先我们需要开启 JMX 的远程监听端口，具体来说就是设置若干 JVM 参数。我们可以在 Tomcat 的 bin 目录下新建一个名为 `setenv.sh` 的文件（或者 `setenv.bat`，根据你的操作系统类型），然后输入下面的内容：

 复制代码

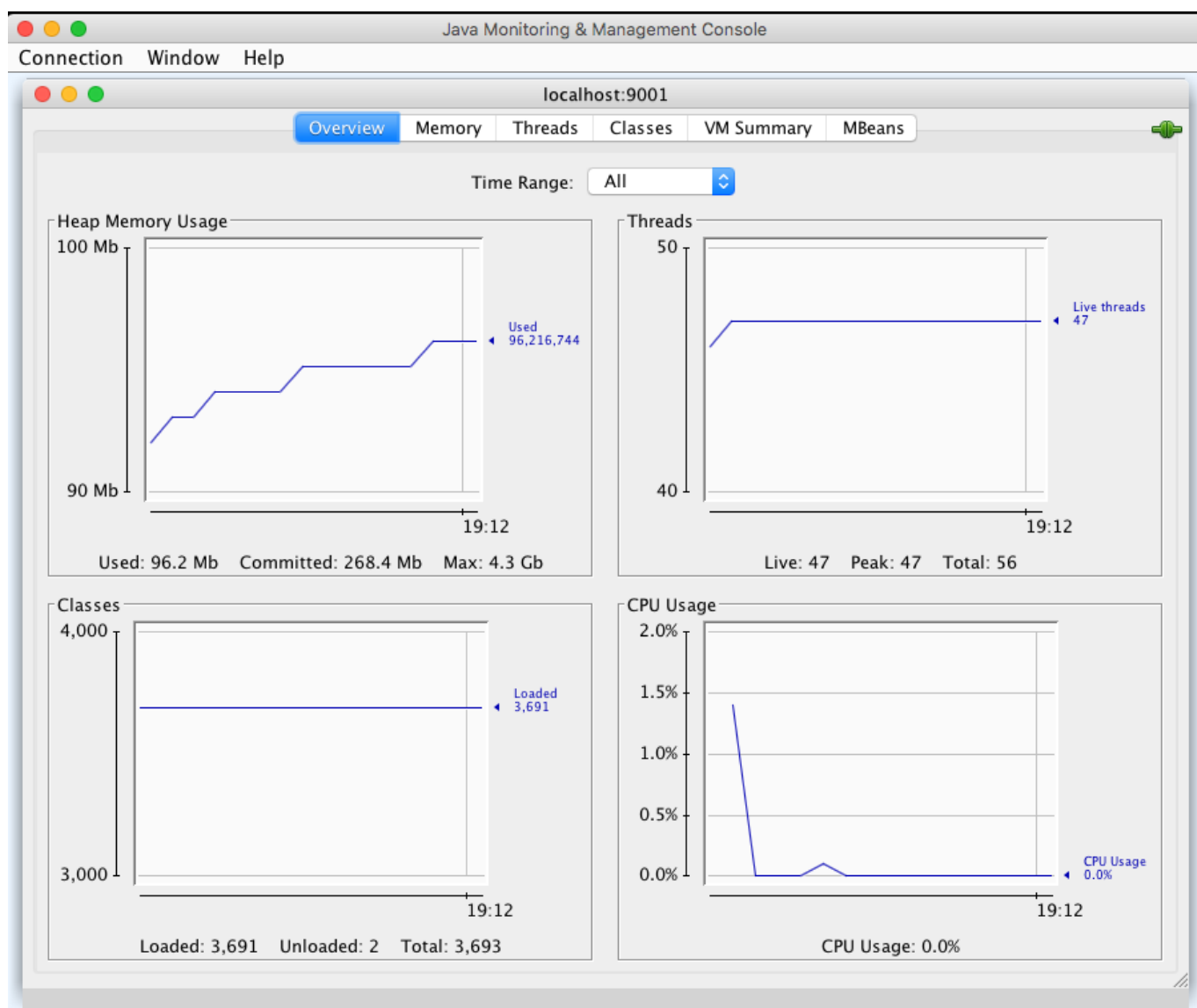
```
1 export JAVA_OPTS="${JAVA_OPTS} -Dcom.sun.management.jmxremote"
2 export JAVA_OPTS="${JAVA_OPTS} -Dcom.sun.management.jmxremote.port=9001"
3 export JAVA_OPTS="${JAVA_OPTS} -Djava.rmi.server.hostname=x.x.x.x"
4 export JAVA_OPTS="${JAVA_OPTS} -Dcom.sun.management.jmxremote.ssl=false"
5 export JAVA_OPTS="${JAVA_OPTS} -Dcom.sun.management.jmxremote.authenticate=false"
```

重启 Tomcat，这样 JMX 的监听端口 9001 就开启了，接下来通过 JConsole 来连接这个端口。

 复制代码

```
1 jconsole x.x.x.x:9001
```

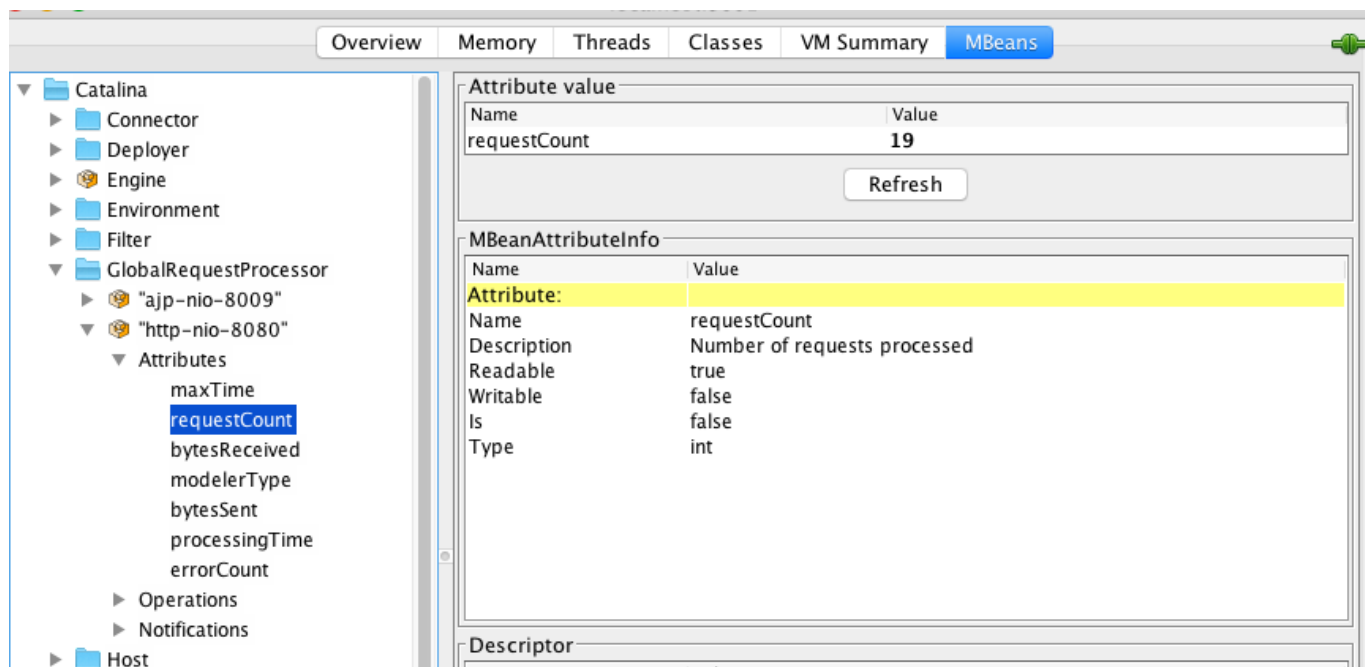
我们可以看到 JConsole 的主界面：



前面我提到的需要监控的关键指标有**吞吐量**、**响应时间**、**错误数**、**线程池**、**CPU** 以及 **JVM 内存**，接下来我们就来看看怎么在 JConsole 上找到这些指标。

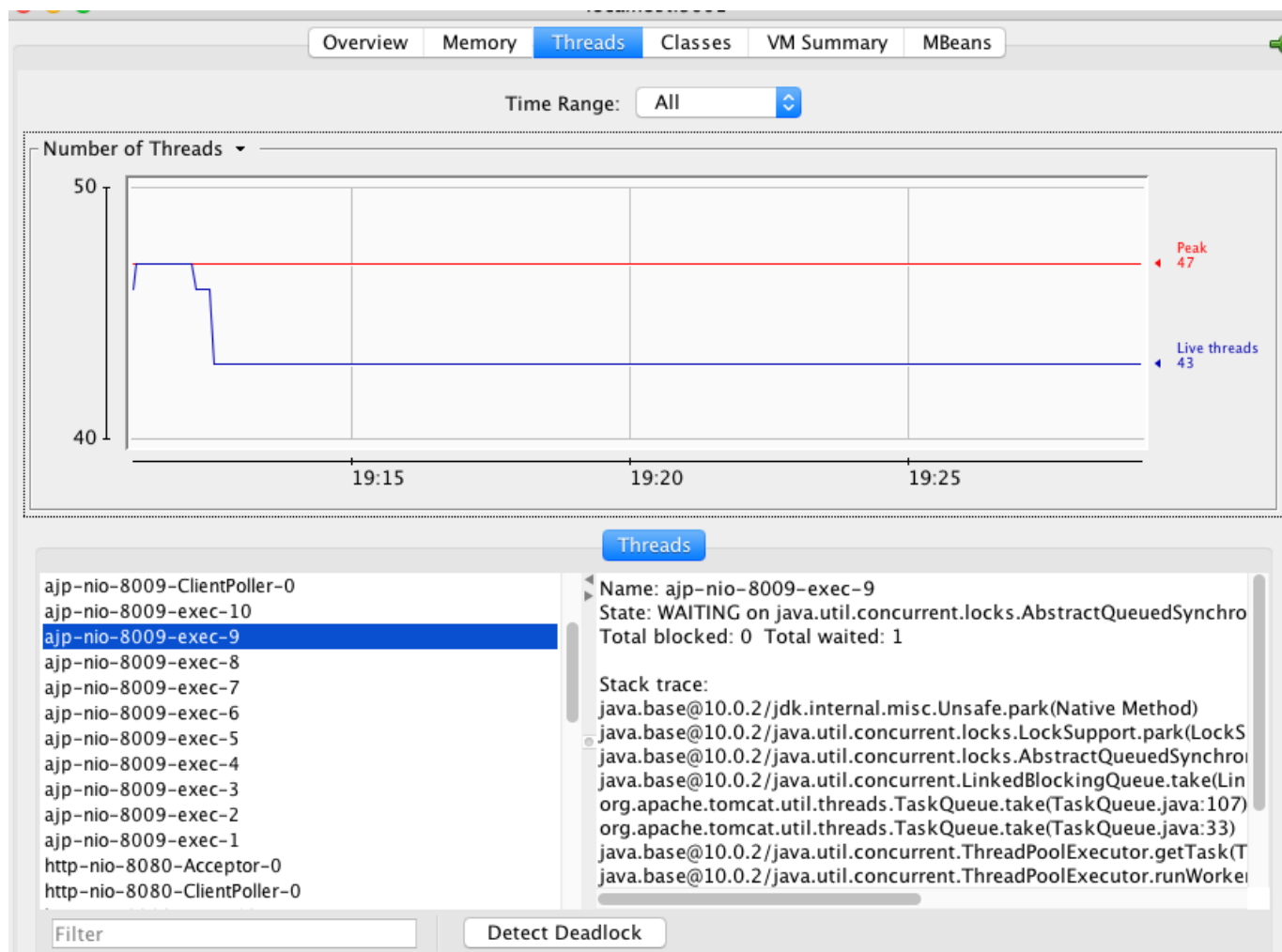
吞吐量、响应时间、错误数

在 MBeans 标签页下选择 GlobalRequestProcessor，这里有 Tomcat 请求处理的统计信息。你会看到 Tomcat 中的各种连接器，展开“http-nio-8080”，你会看到这个连接器上的统计信息，其中 maxTime 表示最长的响应时间，processingTime 表示平均响应时间，requestCount 表示吞吐量，errorCount 就是错误数。



线程池

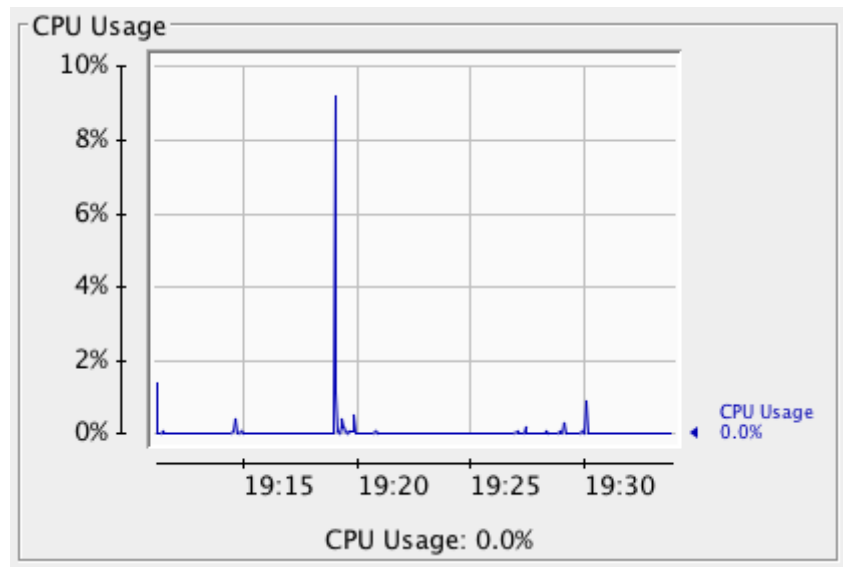
选择“线程”标签页，可以看到当前 Tomcat 进程中有多少线程，如下图所示：



图的左下方是线程列表，右边是线程的运行栈，这些都是非常有用的信息。如果大量线程阻塞，通过观察线程栈，能看到线程阻塞在哪个函数，有可能是 I/O 等待，或者是死锁。

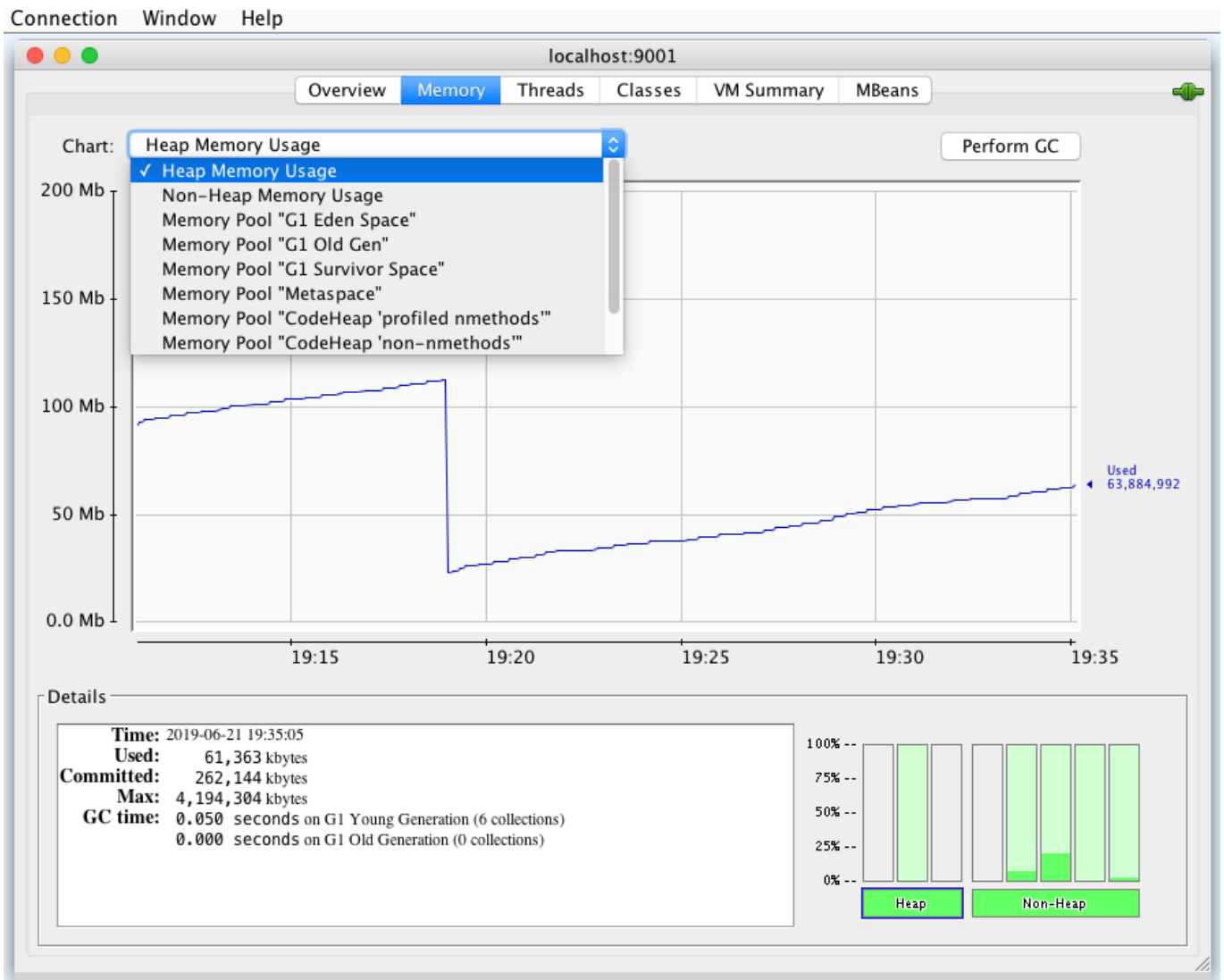
CPU

在主界面可以找到 CPU 使用率指标，请注意这里的 CPU 使用率指的是 Tomcat 进程占用的 CPU，不是主机总的 CPU 使用率。

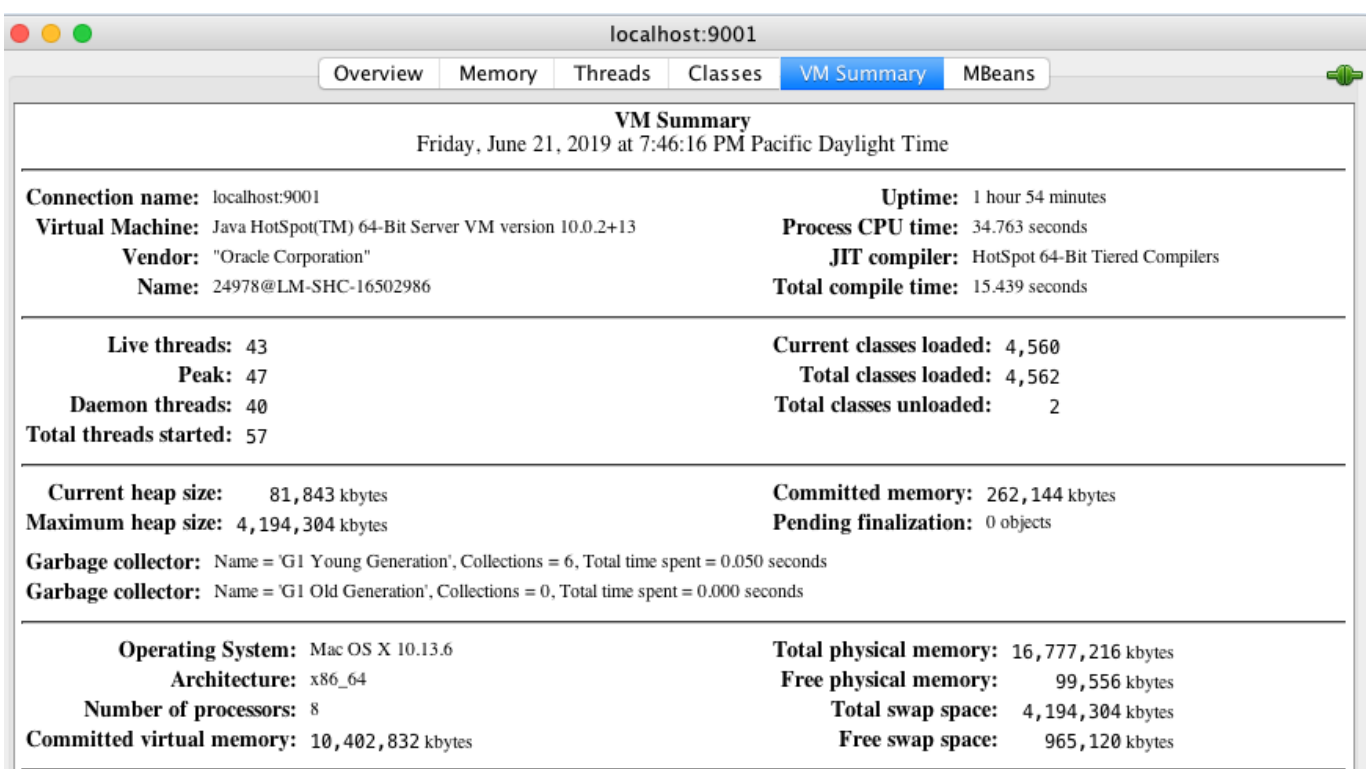


JVM 内存

选择“内存”标签页，你能看到 Tomcat 进程的 JVM 内存使用情况。



你还可以查看 JVM 各内存区域的使用情况，大的层面分堆区和非堆区。堆区里有分为 Eden、Survivor 和 Old。选择“VM Summary”标签，可以看到虚拟机内的详细信息。



命令行查看 Tomcat 指标

极端情况下如果 Web 应用占用过多 CPU 或者内存，又或者程序中发生了死锁，导致 Web 应用对外没有响应，监控系统上看不到数据，这个时候需要我们登陆到目标机器，通过命令行来查看各种指标。

1. 首先我们通过 ps 命令找到 Tomcat 进程，拿到进程 ID。

```
$ ps -ef | grep tomcat
```

2. 接着查看进程状态的大致信息，通过 `cat/proc/<pid>/status` 命令：

```
$ cat /proc/30943/status
Name:   java
State:  S (sleeping)
Tgid:   30943
Ngid:   0
Pid:    30943
PPid:   30940
TracerPid: 0
Uid:    78402    78402    78402    78402
Gid:    61000    61000    61000    61000
FDSize: 8192
Groups: 4 61000
NStgid: 30943
NSpid:  30943
NSpgid: 30939
NSSid:  30939
VmPeak: 16113056 kB
VmSize: 16092192 kB
VmLck:  0 kB
VmPin:  0 kB
VmHWM:  8002064 kB
VmRSS:  7974888 kB
VmData: 16026468 kB
VmStk:  136 kB
VmExe:  4 kB
VmLib:  16916 kB
VmPTE:  26696 kB
VmPMD:  72 kB
VmSwap: 0 kB
HugetlbPages: 0 kB
Threads: 5873
SigQ: 0/48003
```

3. 监控进程的 CPU 和内存资源使用情况：

```
$ top -p 30943
top - 21:10:23 up 59 days,  2:24,  1 user,  load average: 0.14, 0.28, 0.30
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.9 us,   0.4 sy,   0.0 ni, 95.6 id,   0.0 wa,   0.0 hi,   0.1 si,   0.0 st
KiB Mem: 12303860 total, 11363072 used,   940788 free,   132292 buffers
KiB Swap:   0 total,   0 used,   0 free. 2287284 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
30943	cronusa+	20	0	15.347g	7.605g	14816	S	17.0	64.8	2071:47	java

4. 查看 Tomcat 的网络连接，比如 Tomcat 在 8080 端口上监听连接请求，通过下面的命令查看连接列表：

```
$ netstat -na | grep 8080
tcp        0      0 0.0.0.0:8080          0.0.0.0:*             LISTEN
tcp        0      0 127.0.0.1:8080        127.0.0.1:44420        TIME_WAIT
tcp        0      0 10.199.1.103:8080     10.102.153.28:40590    TIME_WAIT
tcp        0      0 10.199.1.103:8080     10.173.117.164:26385   TIME_WAIT
tcp        0      0 10.199.1.103:8080     10.173.117.164:10129   ESTABLISHED
tcp        0      0 10.199.1.103:8080     10.199.73.250:17460    TIME_WAIT
tcp        0      0 10.199.1.103:8080     10.199.73.250:11936    TIME_WAIT
tcp        0      0 10.199.1.103:8080     10.199.105.232:44204   TIME_WAIT
tcp        0      0 10.199.1.103:8080     10.102.153.25:41058    TIME_WAIT
```

你还可以分别统计处在“已连接”状态和“TIME_WAIT”状态的连接数：

```
$ netstat -na | grep ESTAB | grep 8080 | wc -l
12

$ netstat -na | grep TIME_WAIT | grep 8080 | wc -l
43
```

5. 通过 ifstat 来查看网络流量，大致可以看出 Tomcat 当前的请求数和负载状况。


```
$ ifstat
      eth0
KB/s in  KB/s out
  34.19   105.27
  37.62   143.22
  36.38   122.61
  35.40   115.54
```

实战案例

在这个实战案例中，我们会创建一个 Web 应用，根据传入的参数 latency 来休眠相应的秒数，目的是模拟当前的 Web 应用在访问下游服务时遇到的延迟。然后用 JMeter 来压测这个服务，通过 JConsole 来观察 Tomcat 的各项指标，分析和定位问题。

主要的步骤有：


1. 创建一个 Spring Boot 程序，加入下面代码所示的一个 RestController：

 复制代码

```
1 @RestController
2 public class DownStreamLatency {
3
4     @RequestMapping("/greeting/latency/{seconds}")
5     public Greeting greeting(@PathVariable long seconds) {
6
7         try {
8             Thread.sleep(seconds * 1000);
9         } catch (InterruptedException e) {
10             e.printStackTrace();
11         }
12
13         Greeting greeting = new Greeting("Hello World!");
14
15         return greeting;
16     }
17 }
```

从上面的代码我们看到，程序会读取 URL 传过来的 seconds 参数，先休眠相应的秒数，再返回请求。这样做的目的是，客户端压测工具能够控制服务端的延迟。

为了方便观察 Tomcat 的线程数跟延迟之间的关系，还需要加大 Tomcat 的最大线程数，我们可以在 application.properties 文件中加入这样一行：

 复制代码

```
1 server.tomcat.max-threads=1000server.tomcat.max-threads=1000
```

2. 启动 JMeter 开始压测，这里我们将压测的线程数设置为 100：

Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

☒ Continue
 ☐ Start Next Thread Loop
 ☐ Stop Thread
 ☐ Stop Test
 ☐ Stop Test Now

Thread Properties

Number of Threads (users): 100

Ramp-Up Period (in seconds): 1

Loop Count: ☒ Forever

☐ Delay Thread creation until needed

☐ Scheduler

Scheduler Configuration

⚠ If Loop Count is not -1 or Forever, duration will be min(Duration, Loop Count * iteration duration)

Duration (seconds)

Startup delay (seconds)

请你注意的是，我们还需要将客户端的 Timeout 设置为 1000 毫秒，这是因为 JMeter 的测试线程在收到响应之前，不会发出下一次请求，这就意味我们没法按照固定的吞吐量向服务端加压。而加了 Timeout 以后，JMeter 会有固定的吞吐量向 Tomcat 发送请求。

HTTP Request

Name: HTTP Request

Comments:

Basic Advanced

Client implementation

Implementation: HttpClient4

Timeouts (milliseconds)

Connect: Response: 1000

Embedded Resources from HTML Files

☐ Retrieve All Embedded Resources ☐ Parallel downloads. Number: 6 URLs must match:

Source address

IP/Hostname

Proxy Server

Scheme: Server Name or IP: Port Number: Username: Password:

Optional Tasks

☐ Save response as MD5 hash?

3. 开启测试，这里分三个阶段，第一个阶段将服务端休眠时间设为 2 秒，然后暂停一段时间。第二和第三阶段分别将休眠时间设置成 4 秒和 6 秒。

Basic Advanced

Web Server

Protocol [http]: Server Name or IP: localhost Port Number: 8080

HTTP Request

Method: GET Path: /greeting/latency/2 Content encoding:

☐ Redirect Automatically ☒ Follow Redirects ☒ Use KeepAlive ☐ Use multipart/form-data ☒ Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name	Value	URL Encode?	Content-Type	Include Equals?
------	-------	-------------	--------------	-----------------

4. 最后我们通过 JConsole 来观察结果：



下面我们从线程数、内存和 CPU 这三个指标来分析 Tomcat 的性能问题。

首先看线程数，在第一阶段时间之前，线程数大概是 40，第一阶段压测开始后，线程数增长到 250。为什么是 250 呢？这是因为 JMeter 每秒会发出 100 个请求，每一个请求休眠 2 秒，因此 Tomcat 需要 200 个工作线程来干活；此外 Tomcat 还有一些其他线程用来处理网络通信和后台任务，所以总数是 250 左右。第一阶段压测暂停后，线程数又下降到 40，这是因为线程池会回收空闲线程。第二阶段测试开始后，线程数涨到了 420，这是因为每个请求休眠了 4 秒；同理，我们看到第三阶段测试的线程数是 620。

我们再来看 CPU，在三个阶段的测试中，CPU 的峰值始终比较稳定，这是因为 JMeter 控制了总体的吞吐量，因为服务端用来处理这些请求所需要消耗的 CPU 基本也是一样的。

各测试阶段的内存使用量略有增加，这是因为线程数增加了，创建线程也需要消耗内存。

从上面的测试结果我们可以得出一个结论：对于一个 Web 应用来说，下游服务的延迟越大，Tomcat 所需要的线程数越多，但是 CPU 保持稳定。所以如果你在实际工作碰到线程数飙升但是 CPU 没有增加的情况，这个时候你需要怀疑，你的 Web 应用所依赖的下游服务是不是出了问题，响应时间是否变长了。

本期精华

今天我们学习了 Tomcat 中的关键的性能指标以及如何监控这些指标：主要有**吞吐量、响应时间、错误数、线程池、CPU 以及 JVM 内存**。

在实际工作中，我们需要通过观察这些指标来诊断系统遇到的性能问题，找到性能瓶颈。如果我们监控到 CPU 上升，这时我们可以看看吞吐量是不是也上升了，如果是那说明正常；如果不是的话，可以看看 GC 的活动，如果 GC 活动频繁，并且内存居高不下，基本可以断定是内存泄漏。

课后思考

请问工作中你如何监控 Web 应用的健康状态？遇到性能问题的时候是如何做问题定位的呢？

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，也欢迎你把你的课后思考和心得记录下来，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。



深入拆解 Tomcat & Jetty

从源码角度深度探索 Java 中间件

李号双

eBay 技术主管



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 34 | JVM GC原理及调优的基本思路

下一篇 36 | Tomcat I/O和线程池的并发调优

精选留言 (6)

写留言



nightmare

2019-08-01

先查看日志，那些方法耗时较大，用阿里爸爸开源arthas监控有问题的方法，排查问题

1

2



许童童

2019-08-01

使用prometheus + grafana 监控各种指标，每天上班前看一下昨天的情况，设置好阈值，如果达到就报警。

1

1



陆离

2019-08-01

老师，不是很明白线程sleep时间越长，为什么tomcat启动的线程就越多

作者回复: 这是Tomcat需要从线程池拿出一个工作线程来处理请求，请求处理（休眠）的时间越长，这些线程被阻塞，休眠时间越长，被阻塞的线程越多，这些线程无法被线程池回收，Tomcat线程池不得不创建更多的线程来处理新的请求。

1

1



echo_陈

2019-08-01

生产环境网络隔离.....没办法适用jmx

1

1



-W.LI-

2019-08-01

李老师好。

第二三阶段，有很多TIME_WAIT状态的线程。可是CPU使用率并没有增加很多。老师说CPU使用率和吞吐量有关，吞吐量由JMETETER控制一直没变。频繁线程切换会消耗很多CPU支援，如果只是阻塞，切换不频繁。对CPU使用率还是比较小的是么？

展开

作者回复: 对的



a、

2019-08-01

监控系统会每隔一段时间，ping下我们系统，我们系统会pong回监控系统，并带上ip地址，jvm当前使用率，cpu使用率等信息，如果超过一定数值，监控系统就会发出预警信息，我们就需要去生产管理通过日志和命令查看，到底出了什么问题。

展开 ∨

