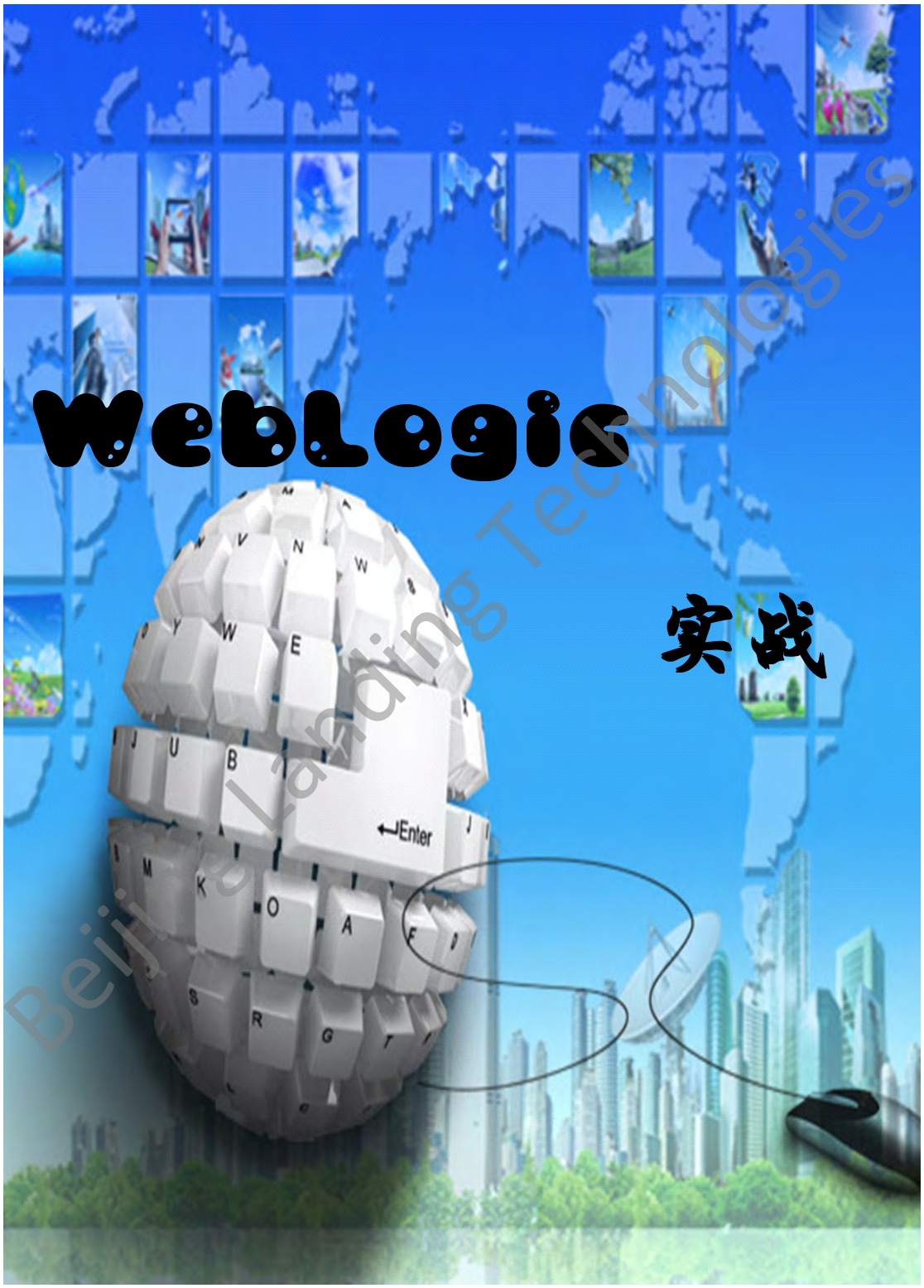
联动北方
LANDING

企业级 IT 运维宝典系列



联动北方 出品

目　录

第 13 章 异常高 CPU 占用率故障

上章我们讲到了常规的服务器挂起故障，表现为系统异常缓慢和阻塞；有些时候，我们会发现，服务器的线程分布其实很正常，但系统还是运行非常缓慢，这个时候，常常会伴随发现系统 CPU 占用率异常高，导致整个系统 CPU 资源争用紧张。

13.1 异常高 CPU 占用率故障

这里我们先对故障本身的相关知识进行一下必要的梳理。

13.1.1 回顾：进程、线程和 CPU 占用率

- **进程**：以操作系统进程 ID(process ID, PID) 来标识，可以是单线程或多线程。
- **线程**：是进程的子任务，与其它线程各占用一部分资源（如 CPU），可能可以映射到操作系统 PID，可以使用轻量型进程 ID(light-weight process id,LWPID) 来识别，该 ID 与父进程关联。
- **CPU 占用率**：计算机的 CPU 资源使用的百分比，CPU 的调度算法一般是按时间片来切分，CPU 的时间片的资源竞争者，可以是计算机上的多个进程，也可以是一个进程内的多个线程。

13.1.2 异常高 CPU 占用率的故障症状

当一个进程或线程占用的 CPU 资源百分百异常高时发生，可能会是其他进程或线程丧失或缺乏执行要求的处理任务所需的 CPU 处理能力，应进行定期检查（监视）。

异常高 CPU 占用率的故障症状：

- 用户响应时间长；
- Weblogic 服务器运行速度异常缓慢；
- 请求或操作开始出现超时。

13.2 异常高 CPU 占用率探查

当能够确认当前系统缓慢，是由于异常高 CPU 占用率引起的时，就需要运用下述的各种手段进行问题定位和探查。

13.2.1 探查概述

首先要确定哪个（些）WebLogic 进程线程导致了异常高 CPU 占用率，做法是在发生高占用率情况时捕捉 Thread Dump，然后找到服务器 Thread Dump 中的高占用率线程(这一过程视所在平台而有所不同，该过程可能包括若干个步骤)

不断重复执行上述过程来确保已建立了模式并找对了线程，可以利用一次或多次 Thread Dump 探查发生异常高 CPU 占用率的具体原因。

13.2.2 Solaris 平台上探查

发生异常高 CPU 占用率时，请重复这些步骤来捕捉服务器活动的快照：

1) 使用以下命令捕捉哪些线程 (LWPID) 正在使用 CPU：prstat -L -p <WLSpid> 11；

2) 使用以下命令获得 LWPID 到（十进制）PID 的映射：pstack <WLSpid> ；

3) 使用以下命令获得服务器 Thread Dump：kill -3 <WLSpid>；

然后按下述方法利用收集到的输出：

1) 在 prstat 输出中找到使用率最高（最高频率条目）的 LWPID；

2) 在 pstack 输出中找到该 LWPID，获得对应的线程编号；

3) 将线程编号转换为十六进制号码；

4) 在服务器 Thread Dump 中找到像“nid=<hexNum>”这样的十六进制线程编号；

5) 确定该线程执行的哪一项任务导致了异常高 CPU 占用率。

Solaris prstat 输出示例：

```
$ prstat -L -p 9499 1 1
PID USERNAME    SIZE RSS STATE PRI NICE TIME CPU PROCESS/LWPID
9499 landingbj  153M 100M sleep 58 0 0:00.22 0.6% java/8
9499 landingbj  153M 100M sleep 58 0 0:00.10 0.2% java/10
9499 landingbj  153M 100M sleep 58 0 0:00.11 0.1% java/9
9499 landingbj  153M 100M sleep 58 0 0:00.03 0.0% java/5
9499 landingbj  153M 100M sleep 58 0 0:01.01 0.0% java/1
9499 landingbj  153M 100M sleep 58 0 0:00.00 0.0% java/12
9499 landingbj  153M 100M sleep 58 0 0:00.00 0.0% java/11
9499 landingbj  153M 100M sleep 58 0 0:00.00 0.0% java/14
9499 landingbj  153M 100M sleep 58 0 0:00.00 0.0% java/13
9499 landingbj  153M 100M sleep 59 0 0:00.07 0.0% java/7
9499 landingbj  153M 100M sleep 59 0 0:00.00 0.0% java/6
9499 landingbj  153M 100M sleep 59 0 0:00.00 0.0% java/4
9499 landingbj  153M 100M sleep 58 0 0:00.11 0.0% java/3
9499 landingbj  153M 100M sleep 58 0 0:00.00 0.0% java/2
```

示例 13-1

说明：在上面的示例中可以看出 LWPID 8 对 CPU 的占用率最高

Solaris pstack 输出示例：

```
----------------- lwp# 8 / thread# 76 --------------------
ff29d190 poll (e2e81548, 0, bb8)
ff24d154 select (0, 0, 0, e2e81548, ff2bf1b4, e2e81548) + 348
ff36b134 select (0, bb8, 7fffffff, fe4c8000, 0, bb8) + 34
fe0f62e4 __1cCosFsleep6FpnGThread_xl_i_ (0, bb8, fe4c8000, 1, 0,
 1e2fd8) + 234
fe23f050 JVM_Sleep (2, 0, bb8, fe4de978, fe4c8000, 1e2fd8) + 22c
0008f7ac ???????? (e2e818d4, bb8, 1e2fd8, 984a4, 0, 109a0)
0008c914 ???????? (e2e8194c, 1, fe4d6a80, 98564, 8, e2e81868)
fe5324e8 __1cMStubRoutinesG_code1_ (e2e819d8, e2e81c10, a, f6cb5000, 4,
 e2e818f0) + 3ec
fe0cbe94 __1cJJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_
pnRJavaCallArguments_pnGThread__v_ (e2e81c08,fe4c8000, e2e81b54, 1e2fd8,
 8e764, e2e81c10) +308
fe1f6dbc __1cJJavaCallsMcall_virtual6FpnJJavaValue_nLKlassHandle_
```

```
    nMsymbolHandlee81c08, e2e81b54) + 150pnGThread__v_(f6cb64b8, e2e81b40,
     e2e81b44, fe4c8000, e2d8) + 60e_5pnGThread__v_ (e2e81c08, e2e81c04,
     e2e81c00,e2e81bf4, e2e81bec, 1e2f8000, e2e81d10, 1e, e) +
     120FpnKJavaThread_pnGThread__v_ (f6817ff8, 1e2fd8, fe4c 7fd70) +
     3d8cKJavaThreadDrun6M_v_ (e2e02000, fe4d3e34, fe4c8000, 7fd70,1e2fd8,
    fe213ec8 _start (fe4c8000, fe625d10, 0, 5, 1, fe401000) + 20
    ff36b728 _thread_start (1e2fd8, 0, 0, 0, 0, 0) + 40
```

示例 13-2

说明：运行 pstack 命令得到一个关于 Light Weight Process ID(LWPID)和 PID(Process ID)的映射。

例如，pstack 9499 并重定向到一个输出文件中，在上面的示例中 pstack 命令输出”lwp#8”映射到”thread#76”。

Solaris Thread Dump 示例：

```
    $ kill -3 p 9499
    . . .
    "Thread-6" prio=5 tid=0x1e2fd8 nid=0x4c waiting on monitor
[0xe2e81000..0xe2e819d8]
    at java.lang.Thread.sleep(Native Method)
    at weblogic.management.deploy.GenericAppPoller.run
    (GenericAppPoller.java:139)
    . . .
```

示例 13-3

说明：由于 lwp#8 映射到 thread#76，你需要将十进制的 76 转换成 16 进制格式，即 0x4c。

执行 kill -3 <WLSpid>得到 Thread Dump，在 Thread Dump 中找到本地线程“nid=0x4c”。

在这个示例里，占用 CPU 最高的线程正在 sleeping，可见这个 Thread Dump 并没有及时捕获线程活动状态即线程正在执行的操作。为快速、重复的捕捉数据，可编写一个脚本；下边是脚本示例：

```
    for loopnum in 1 2 3
    do
    prstat -L -p $1 1 1 >> dump_high_cpu.txt
    pstack $1 >> dump_high_cpu.txt
    kill -3 $1
    echo "prstat, pstack, and thread dump done. #" $loopnum
    sleep 1
    echo "Done sleeping."
    done
```

示例 13-4

说明：携带参数（WebLogic 进程的 PID），重复执行三次。此脚本可将 prstat 和 pstack 信息追加到文件 dump_high_cpu.txt。Thread Dump 信息会出现在将 stdout 重定向到的文件中或输出到屏幕上。

经过脚本的快速执行捕获，一般就能从 Thread Dump 中可以判断出现场在执行什么操作引起高 CPU 利用。

13.2.3 HP-UX 平台上探查

可以使用比较容易下载到的，原 BEA 技术支持部门开发的 hp_prstat 使用程序，在发生异常高 CPU 占用率时，请重复这些步骤来确定服务器活动的模式：

1）使用以下命令捕捉那些正在使用 CPU 的线程（LWPID）：hp_prstat <WLSpid>；

2）使用以下命令获得服务器 Thread Dump：kill -3 <WLSpid>。

然后按下述方法利用收集到的输出：

1）找到“用户时间”增加最快的一个或多个 LWPID；

2）在服务器 Thread Dump 中找到像“lwp_id=<LWPID>”这样的对应 LWPID 号码；

3）确定该线程执行的哪一项任务导致了异常高 CPU 占用率。

HP-UX hp_pstat 示例：

```
$ hp_prstat  4426
lwpid  pid  pri status UsrTime SysTime
285365 4426 154  1     29      3
. . .
285415 4426 154  1     0       7
285416 4426 154  1     0       7
285417 4426 154  1     0       7


. . .
$ hp_prstat  4426
lwpid  pid  pri status UsrTime SysTime
285365 4426 154  1     29      3
. . .
285415 4426 154  1     0       7
285416 4426 154  1     13      7
285417 4426 154  1     0       7
```

示例 13-5

说明：查找 UsrTime 增长最快的那一行获取 LWPID。（可能有多个增长速度较快的可疑线程，全部取出）

HP-UX Thread Dump 示例：

```
$ kill -3 4426
"Thread-6" prio=8 tid=0x0004f620 nid=75 lwp_id=285475 waiting on monitor
[0x66d5e000..0x66d5e500]
    at java.lang.Thread.sleep(Native Method)
    at weblogic.management.deploy.GenericAppPoller.run
    (GenericAppPoller.java:139)
    "ExecuteThread: '11' for queue: 'default'" daemon prio=10 tid=0x0004ad00
nid=23 lwp_id=285416 runnable [0x67874000..0x67874500]
    at java.net.SocketOutputStream.socketWrite(Native Method)
    at java.net.SocketOutputStream.write(Unknown Source)
    . . .
    at examples.servlets.HelloWorldServlet.service
```

```
    (HelloWorldServlet.java:28)
    at javax.servlet.http.HttpServlet.service
    (HttpServlet.java:853)
    . . .
```

示例 13-6

说明：从线程快照中找到之前取得的 LWPID，检查是什么原因引起 CPU 的高利用率。

13.2.4 Linux 平台上探查

在 Linux 传统系统上，每个线程均作为独立的进程出现。

发生异常高 CPU 占用率时，请重复这些步骤来捕捉服务器活动的快照：

1）使用 top 确定哪些线程正在使用 CPU；

2）使用以下命令获得服务器 Thread Dump：kill - 3 <WLSpid>。

然后按下述方法利用收集到的输出：

1）在 top 输出中，寻找具有启动服务器的 userID 的进程线程；

2）获得 CPU 占用率最高的服务器线程的 PID；

3）将高占用率的 PID 转换为十六进制值；

4）在服务器 Thread Dump 中找到十六进制 PID；

5）确定该线程执行的哪一项任务导致了异常高 CPU 占用率。

Linux 上 top 输出示例：

```
PID   USER      PRI NI SIZE  RSS SHARE STAT  %CPU %MEM TIME COMMAND
...
22962 landingbj  9   0  86616 84M 26780 S     0.0  4.2  0:00 java
...
```

示例 13-7

说明：在传统 Linux 上每个线程被映射成为一个进程，这点和其他类 Unix 系统不同。上面给出是只是 top 命令输出的很小的一段内容。

Thread dump 输出示例：

```
    . . .
    "ExecuteThread: '0' for queue: 'default'" daemon prio=1 tid=0x83da550
nid=0x59b2 waiting on monitor [0x56138000..0x56138870]
    at java.lang.Object.wait(Native Method)
    at java.lang.Object.wait(Object.java:415)
    at weblogic.kernel.ExecuteThread.waitForRequest(ExecuteThread.java:146)
    at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:172)
    . . .
```

示例 13-8

说明：如果 PID 是 22962 转换成十六进制为 0X59B2。查询 thread dump 找到 nid=0X59B2 的线程。例如上面的示例 0X59B2 对应执行线程 0，在上面的例子中 CPU 利用率为 0%。

为快速、重复地捕捉数据，可编写一个脚本，如下是脚本示例：

```
for loopnum in 1 2 3
do
top -b -n1 >> dump_high_cpu.txt
kill -3 $1
echo "cpu snapshot and thread dump done. #" $loopnum
sleep 1
echo "Done sleeping."
done
```

示例 13-9

说明：携带参数（WLS 进程的 PID），重复执行三次。此脚本把 top 信息追加到名为 dump_high_cpu.txt 的文件。Thread Dump 信息会出现在将 stdout 重定向到的文件中或输出到屏幕上。

13.2.5 AIX 平台上探查

发生异常高 CPU 占用率时，请重复这些步骤来捕捉服务器活动的快照：

1) 找到正在使用 CPU 的线程 ID(thread ID, TID)：ps -mp <WLSpid> -o THREAD 即 CPU 值最高的线程；

2) 使用以下命令获得服务器 Thread Dump：kill -3 <WLSpid> 。

然后按下述步骤进行探查：

1) 使用以下命令在服务器进程上运行 dbx：dbx -a <WLSpid> ；

2) 在 dbx 中，使用以下命令获得所有线程的列表：thread；

3) 在输出线程列表中，找到高占用率 TID 及其号码（$t<num> 形式的号码）；

4) 在 dbx 中，使用以下命令获得详细的线程信息：th info <num> ；

5) 在“general”输出中，找到 pthread_t（十六进制）值；

6) 从进程中分离：detach(重要)；

7) 在服务器 Thread Dump 中找到像 “native ID:<hexNum>” 这样的十六进制 phtread_t 号码；

8) 确定该线程执行的哪一项任务导致了异常高 CPU 占用率。

AIX 上 ps 命令输出示例：

```
$ ps -mp 250076 -o THREAD
USER       PID    PPID  TID ST CP PRI SC WCHAN F     TT    BND COMMAND
landingbj 250076 217266 -   A  38 60  72  *    242011 pts/0 -
/wwsl/sharedInstalls/aix/jdk130/...
- - -     315593 Z 0 97  1 - c00007 - - -
- - -     344305 S 0 60  1 f1000089c020e200 400400 - - -
- - -     499769 S 0 60  1 f1000089c0213a00 400400 - - -
. . .
- - -     655429 S 0 60  1 f10000879000a040 8410400 - - -
- - -     659527 S 0 60  1 f10000879000a140 8410400 - - -
- - -     663625 S 0 60  1 f10000879000a240 8410400 - - -
- - -     667723 S 37 78 1 f1000089c020f150 400400 - - -
- - -     671821 S 0 60  1 f10000879000a440 8410400 - - -
- - -     675919 S 0 60  1 - 418400 - - -
```

```
. . .
```

示例 13-10

说明：执行命令 ps －mp <WLSpid> -o THREAD 查看相关内容。注意 TID“667723”在 CP 列数值达到 37 而其他线程接近 0。

AIX dbx thread 示例：

```
$dbx -a 250076
(dbx) thread
thread state-k wchan state-u k-tid mode held scope function
. . .
$t15 wait 0xf10000879000a140 blocked 659527 k no sys _event_sleep
$t16 wait 0xf10000879000a240 blocked 663625 k no sys _event_sleep
$t17 run running 667723 k no sys JVM_Send
$t18 wait 0xf10000879000a440 blocked 671821 k no sys _event_sleep
$t19 wait running 675919 k no sys poll
$t20 wait 0xf10000879000a640 blocked 680017 k no sys _event_sleep
. . .
```

示例 13-11

说明：执行命令“dbx －a 250076”，在 dbx 中列出所有本地线程执行命令“thread”，上面列出的只是有关线程的一小部分。

AIX dbx th info 示例：

```
(dbx) th info 17
thread state-k wchan state-u k-tid mode held scope function
$t17 run running 667723 k no sys JVM_Send
general:
   pthread addr = 0x3ea55c68 size = 0x244
   vp addr = 0x3e69e5e0 size = 0x2a8
   thread errno = 2
   start pc = 0x300408b0
   joinable = no
   pthread_t = 1011
scheduler:
   kernel =
   user = 1 (other)
event :
   event = 0x0  cancel = enabled, deferred, not pending
stack storage:
. . .
(dbx)detach
```

示例 13-12

说明：为了得到本地线程的信息，执行命令：th info 17

AIX Thread Dump 示例：

```
"ExecuteThread: '11' for queue: 'default'" (TID:0x31cf86d8,
sys_thread_t:0x3e5ea108, state:R, native ID:0x1011) prio=5
   at java.net.SocketOutputStream.socketWrite(Native Method)
```

```
    . . .
    at java.io.PrintWriter.println(PrintWriter.java(Compiled Code))
    at examples.servlets.HelloWorldServlet.service(HelloWorld
    Servlet.java(Compiled Code))
    at javax.servlet.http.HttpServlet.service
    (HttpServlet.java:853)
    at weblogic.servlet.internal.ServletStubImpl$Servlet
    InvocationAction.run(ServletStubImpl.java:1058)
    at weblogic.servlet.internal.ServletStubImpl.invokeServlet
(ServletStubImpl.java:401)
     at weblogic.servlet.internal.ServletStubImpl.invokeServlet
    (ServletStubImpl.java:306)
    . . .
```

示例 13-13

说明：拿到 pthread_t 号并用它在 thread dump 找到正确的线程，即从上步中取得的 pthread_t 所匹配的 native ID,在 thread dump 中找到相应线程，并查看其状态。

13.2.6 Windows 平台上探查

下载一个工具来分析线程的 CPU 的占用率：pslist 可提供进程的线程详细信息。另外，还有一个 Process Explorer 是可显示进程的动态性能统计信息的图形化工具。

发生异常高 CPU 占用率时，请重复这些步骤来确定服务器的线程活动：

1) 使用以下两种工具之一捕捉正在使用 CPU 的线程 (LWPID)：pslist -d <WLSpid> 或 Process Explorer;

注：下载地址: http://www.sysinternals.com/ntw2k/freeware/pslist.shtml

和 http://www.sysinternals.com/ntw2k/freeware/procexp.shtml

2) 使用以下命令获得服务器 Thread Dump：Ctrl+Break

然后按下述方法利用收集到的输出：

1) 找到“用户时间”和“内核时间”增加最快的线程;

2) 将线程编号转换为十六进制值，如<hexNum>;

3) 在服务器 Thread Dump 中找到像“nid=<hexNum>”这样的十六进制线程编号;

4) 确定该线程执行的哪一项任务导致了异常高 CPU 占用率。

pslist 输出示例:

```
$ pslist -d 1720
java 1720:
Tid  Pri Cswtch State        User Time   Kernel Time Elapsed Time
1520 8   9705   Wait:UserReq 0:00:23.734 0:00:01.772 0:08:14.511
1968 8   6527   Wait:UserReq 0:00:06.309 0:00:00.070 0:08:14.120
1748 15  157    Wait:UserReq 0:00:00.010 0:00:00.010 0:08:14.110
. . .
588 10   59123  Wait:UserReq 0:00:48.830 0:00:02.633 0:08:01.211
1784 8   150    Wait:UserReq 0:00:00.090 0:00:00.000 0:08:01.201
1756 8   251    Wait:UserReq 0:00:00.941 0:00:00.000 0:08:01.201
1716 8   6      Wait:Queue   0:00:00.000 0:00:00.000 0:08:01.191
```

```
    1800 8    1457    Wait:Queue    0:00:00.761 0:00:00.210 0:08:01.191
    . . .
```

示例 13-14

说明：在 WebLogic 进程上执行命令：“pslist -d 1720”，一段时间之后执行相同操作，对比一段时间之内 User Time/Kernel Time 增长最快的线程。在上面的例子中 ID 588 号线程出现问题。拿到线程号 588 转换成十六进制及 0X24C。在 thread dump 中查找“nid=0x24c”号线程。

Windows 平台下 Thread Dump 示例：

```
    "ExecuteThread: '10' for queue: 'default'" daemon prio=5 tid=0x20d75808
nid=0x24c runnable [219ff000..219ffd90]
    at java.net.SocketOutputStream.socketWrite0(Native Method)
    . . .
    at java.io.PrintWriter.println(PrintWriter.java:515)
    - locked <0x11d0d1c0>
    (a weblogic.servlet.internal.ChunkWriter)
    at examples.servlets.HelloWorld2.service(HelloWorld2.
    java:94)
    at javax.servlet.http.HttpServlet.service
    (HttpServlet.java:853)
    at weblogic.servlet.internal.ServletStubImpl$ServletInvocationAction.run
    (ServletStubImpl.java:1058)
    . . .
```

示例 13-15

13.3 异常高 CPU 占用率故障排除策略及相关资源

对这类问题的故障排除策略如下：

（1）通过重复收集信息确定占用率模式：

- 捕捉 CPU 占用率高的线程或用户时间快速增加的线程的快照；
- 同时获得 Thread Dump；
- 使用各平台提供的工具。

（2）在服务器的一个或多个 Thread Dump 中查找异常高 CPU 占用率线程；

（3）在 Thread Dump 中探查导致异常高 CPU 占用率的具体原因；

（4）继续监视，看是否还会发生故障。