tomcat&java异常排查思路
好的，在 Ubuntu 14.04 上排查 Tomcat
异常导致网页访问缓慢的问题，可以遵循以下步骤和命令。核心思路是：先定位是系统资源瓶颈还是应用内部
问题，再深入 Java 应用内部。

排查思路流程图

` ` ` mermaid
flowchart TD
A[Tomcat访问缓慢] --> B(系统级资源排查)
A --> C(应用级内部排查)

subgraph B[第一步：系统资源排查]
direction TB
B1[CPU
top, vmstat] --> B2{资源异常？ } B3[内存
free, vmstat] --> B2
B4[磁盘 I/O
iostat, iotop] --> B2
B5[网络
netstat, ss] --> B2
B2 -- 是 --> B6[针对性地解决资源瓶颈]
end

subgraph C[第二步： Java应用排查]
direction TB
C1[检查 Tomcat日志
tail -f catalina.out] --> C2{发现异常错误？ } C2 -- 是 --> C3[根据错误日志解决]
C2 -- 否 --> C4[检查线程状态
jstack]
C4 --> C5[分析线程栈
查找阻塞、死锁、等待]
C5 --> C6[发现可疑线程？]
C6 -- 是 --> C7[结合代码分析问题]
end

B2 -- 否 --> C
C6 -- 否 --> D[第三步：性能分析]
D --> D1[生成 Heap Dump
jmap] D --> D2[分析内存使用
MAT, jhat] D --> D3[找出内存泄漏或过度消耗]
` ` `

第一步：系统级资源排查 (快速确认瓶颈)

首先排除服务器本身的资源瓶颈（ CPU、内存、磁盘 I/O）。

1. **综合性能查看 (`top` / `htop`)**
` ` ` bash
top
` ``

* **看第一行 (load average):** 1分钟、 5分钟、 15分钟的系统平均负载。如果值 **超过 CPU核心数 ** （例如 4核机器负载长期 >4），说明系统负载过高。
* **看 %CPU:** 检查 `%Cpu(s)` 这一行的 `%id`（空闲）是否很低， `%wa`（I/O等待）是否很高。高
`%wa` 表示磁盘是瓶颈。
* **看 %MEM:** 向下看进程列表，按 `Shift + M` 按内存使用排序，找到 `java` 或 `tomcat`
进程，看其内存使用率是否异常高，接近分配的最大值。

2. **内存和 Swap使用情况 (`free`)**
` `` bash
free -h
` ``

* 重点关注 `available` 列（ Ubuntu 14.04 可能没有，就看 `free` 列）。如果 `free` 内存极少，并且
`swap` 的 `used` 在不断增加，说明物理内存不足，系统在使用交换分区，这会极大拖慢速度。

3. **磁盘 I/O状态 (`iostat`, `iotop`)**
` ` ` bash
需要安装 sysstat 包: sudo apt-get install sysstat
iostat -xm 2

需要安装 iotop 并需要 root权限: sudo apt-get install iotop && sudo iotop
` ` `

* 查看 `%util` 列，如果持续接近 100%，表示磁盘 I/O非常繁忙。
* 查看 `await` 列，表示 I/O操作的平均等待时间（毫秒），如果很高（例如 >100ms
），说明磁盘响应慢。

4. **网络连接检查 (`netstat`, `ss`)**
` ` ` bash
查看 Tomcat连接数（ ESTABLISHED状态）
netstat -natp | grep :8080 | grep ESTABLISHED | wc -l

或者使用更快的 ss 命令
ss -t sport = :8080 state established | wc -l
` ` `

* 连接数过多可能耗尽线程池资源。

第二步： Java应用内部排查 (核心步骤)

如果系统资源正常，问题很可能出在应用本身： GC、死锁、阻塞、内存泄漏等。

1. **查看 Tomcat日志 **
` `` bash
进入 Tomcat的 logs目录，查看最新的 catalina.out和localhost日志
cd /path/to/tomcat/logs
tail -f catalina.out
grep -i "exception" catalina.out
` ` `

* **首先检查有无明显的异常堆栈信息 **，如 `OutOfMemoryError`, `NullPointerException`, 数据库连接超时等。

2. **查找 Java进程 PID**
` `` bash
通常 Tomcat的进程名是 java
ps -ef | grep java
或
jps -l
` ``

* 找到对应的 Tomcat进程的 PID（例如 `12345`）。

3. **检查 GC情况 (`jstat`) - 非常有用 **
` `` bash
需要 JDK工具，假设 PID为 12345
jstat -gcutil 12345 2s
` ``

* **输出列解读 :** `S0`, `S1` (幸存区使用率), `E` (伊甸园区使用率), `O` (老年代使用率), `M` (
元空间使用率), `YGC` (年轻 GC次数), `YGCT` (年轻 GC时间), `FGC` (Full GC次数), `FGCT` (Full GC时间)
* **关键指标:**
* **`FGC` (Full GC) 次数频繁 ** 且 **`FGCT` (Full GC时间) 很长 **：这是导致应用 “卡顿 ”
的最常见原因。如果 `O` (老年代) 使用率一直很高（如 >95%），很可能存在 **内存泄漏 **。

4. **查看线程栈 (`jstack`) - 排查死锁和阻塞 **
` `` bash
生成当前线程的快照
jstack 12345 > jstack.log

或者连续抓取几次，对比分析
jstack 12345 > jstack_1.log
sleep 10
jstack 12345 > jstack_2.log
` ` `

* **分析 `jstack.log`:**
* 搜索 `deadlock` 看有无死锁信息。
* 搜索 `blocked` 看大量线程阻塞在哪个锁上。
* 看线程的 **状态 ** (`RUNNABLE`, `WAITING`, `BLOCKED`) 和**堆栈信息 **

，找出大多数线程在做什么。例如，如果很多线程卡在数据库连接 `getConnection()` 上，可能是连接池不够用。

5. **生成堆转储文件 (`jmap`) - 用于分析内存泄漏 **
   ```bash
   # 生成 Heap Dump文件（文件较大，注意磁盘空间）
   jmap -dump:live,format=b,file=heap_dump.hprof 12345
   ```
   * 这个命令会对生产环境有一定影响（可能触发 STW的 Full GC），谨慎使用。
   * 生成的文件 `heap_dump.hprof` 可以下载到本地，用 **Eclipse MAT (Memory Analyzer Tool)** 或 **jhat** 工具分析，查看是什么对象占用了大量内存。

第三步: Tomcat自身配置检查

1. **连接数配置 **
   检查 `$CATALINA_HOME/conf/server.xml` 中的连接器配置:
   ```xml
   <Connector port="8080" protocol="HTTP/1.1"
              connectionTimeout="20000"
              maxThreads="200" <!-- 最大线程数是否足够? -->
              acceptCount="100" <!-- 等待队列长度 -->
              ... />
   ```
   * 如果并发请求数超过 `maxThreads`，多余的请求会在队列中等待，导致响应变慢。

总结与行动指南

1. **`top`** -> 发现 CPU或内存高。
2. **`jstat -gcutil`** -> 确认是频繁 Full GC导致。
3. **`jstack`** -> 检查线程状态，看 GC时线程在做什么，有无死锁。
4. **`jmap` + MAT** -> 分析是什么对象占满了内存，定位内存泄漏的代码。
5. **查看日志 ** -> 始终是第一步，寻找任何明显的错误线索。
6. **检查配置 ** -> 确认 `maxThreads`、数据库连接池大小等配置是否合理。

根据上述步骤，你通常可以定位到是系统资源问题、 GC问题、代码阻塞问题还是配置问题。

好的 我帮你写一个 适合 **Ubuntu14.04 + Tomcat** 的监控采集脚本 。
脚本会每天定期执行，调用 `jstack` 或 `jstat` 收集 Tomcat 的 JVM 信息，并写入日志文件，每条日志前加上时间戳。

示例脚本 `collect_tomcat_jvm.sh`

```
#!/bin/bash
# □□ Tomcat JVM □ jstack/jstat □□
# □□□□□□□□/var/log/tomcat_jvm_monitor.log

# □□□
LOG_DIR="/var/log"
LOG_FILE="$LOG_DIR/tomcat_jvm_monitor.log"
JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64"   # □□□□□□□□ JDK □□
TOMCAT_USER="tomcat"  # □□□□□ tomcat □□□
DATE=$(date +"%Y-%m-%d %H:%M:%S")

# □□ Tomcat □ Java □□ PID
PID=$(ps -u $TOMCAT_USER -o pid,comm | grep java | awk '{print $1}')

if [ -z "$PID" ]; then
    echo "[$DATE] □□□ Tomcat □□" >> $LOG_FILE
    exit 1
fi

# □□□□□
echo "==================== [$DATE] ====================" >> $LOG_FILE

# □□ jstat GC □□
$JAVA_HOME/bin/jstat -gc $PID >> $LOG_FILE 2>&1

# □□□□□□
$JAVA_HOME/bin/jstack -l $PID >> $LOG_FILE 2>&1

echo -e "\n" >> $LOG_FILE
```

□□□

1. □□□□
   □□□ `/usr/local/bin/collect_tomcat_jvm.sh` □□□□□□□□□

   ```
   sudo chmod +x /usr/local/bin/collect_tomcat_jvm.sh
   ```

2. □□□□
   □□ `/var/log/` □□□□□□□□□□□□□□□□□

3. 定时任务

使用 `cron` 每天定时执行，比如每天凌晨 2 点收集一次：

```
crontab -e
```

添加：

```
0 2 * * * /usr/local/bin/collect_tomcat_jvm.sh
```

如果你想更频繁（比如每 30 分钟收集一次），可以写：

```
*/30 * * * * /usr/local/bin/collect_tomcat_jvm.sh
```

📦 建议：

- `jstat` 适合收集 JVM GC/内存指标
- `jstack` 适合排查线程阻塞、死锁问题
- 日志文件会越来越大，可以结合 `logrotate` 自动切割。

要不要我帮你把这个脚本扩展成 **logrotate** 配置，避免日志无限增长？

好的 📦 我帮你把 `collect_tomcat_jvm.sh` 脚本对应的 **logrotate** 配置文件 写出来，避免日志无限增长。

你现在的日志文件路径是：

```
/home/shuncom/tomcat_jvm_monitor.log
```

---

logrotate 配置示例

新建文件 `/etc/logrotate.d/tomcat_jvm_monitor`：

```
/home/shuncom/tomcat_jvm_monitor.log {
    daily
    rotate 7
    compress
    delaycompress
    missingok
    notifempty
    create 0644 root root
    sharedscripts
    postrotate
```

```
        true
    endscript
}
```

以下对各行参数进行注释，但不能把行内注释写入脚本

```
/home/shuncom/tomcat_jvm_monitor.log {
    daily               # 每天轮转一次
    rotate 7            # 保留 7 天的日志，超过删除
    compress            # 启用压缩，老日志会变成 .gz
    delaycompress       # 延迟一天再压缩，避免影响最新日志查看
    missingok           # 如果日志不存在，不报错
    notifempty          # 如果日志为空，不进行轮转
    create 0644 root root  # 新日志的权限和属主（根据实际情况修改）
    sharedscripts
    postrotate
        # logrotate 每次切割日志后执行的操作
        # 这里不需要额外操作，因为脚本本身会继续写新日志
        true
    endscript
}
```

检查 logrotate 是否正常工作

手动执行一次（模拟日志轮转）：

```
sudo logrotate -f /etc/logrotate.d/tomcat_jvm_monitor
```

执行后，你应该会看到：

```
/home/shuncom/tomcat_jvm_monitor.log  →  /home/shuncom/tomcat_jvm_monitor.log.1
```

老日志会被压缩（如 `.1.gz`），新日志文件会重新创建。

⚠ 注意事项：

1. 如果 `collect_tomcat_jvm.sh` 是用 `root` 运行的，`create 0644 root root` 就对了；
   如果是 Tomcat 用户运行，改成 `create 0644 tomcat tomcat`。
2. 默认 logrotate 会每天运行一次，定时任务在 `/etc/cron.daily/logrotate`。
3. 如果日志量特别大，可以改成 `size 100M`（按大小轮转）。

要不要我帮你改成 “按大小 + 按天混合轮转”，比如每天轮转一次，或者日志超过 200MB 时立即切割？

明白了，你要 按天 + 按大小混合轮转 ：

- 每天轮转一次；
- 如果日志超过 **100MB**，即使没到一天也立即切割。

logrotate 支持这种组合配置。

配置文件

`/etc/logrotate.d/tomcat_jvm_monitor`

```
/home/shuncom/tomcat_jvm_monitor.log {
    daily               # 每天轮转一次
    size 100M           # 日志超过 100MB 时立即切割
    rotate 7            # 保留 7 个历史文件
    compress            # 压缩历史日志
    delaycompress       # 延迟一天再压缩，避免影响最新日志查看
    missingok           # 如果日志不存在，不报错
    notifempty          # 如果日志为空，不轮转
    create 0644 root root   # 新日志文件权限和属主
    sharedscripts
    postrotate
        true
    endscript
}
```

验证

手动测试（强制执行一次）：

```
sudo logrotate -f /etc/logrotate.d/tomcat_jvm_monitor
```

日志会被切割为：

```
tomcat_jvm_monitor.log
tomcat_jvm_monitor.log.1
```

```
tomcat_jvm_monitor.log.2.gz
...
```

⚠ 补充说明:

- `daily + size` 会实现 二选一触发 ，即每天轮转一次，或者超过 100MB 时立即切割。
- `rotate 7` 表示最多保留 7 个旧文件，超过会自动删除。
- 如果你想确保 每天只切一次 （即使超过 100MB 也只在第二天切割），那要用 `maxsize` 而不是 `size` 。

要不要我帮你写个对比表，解释 `size` 和 `maxsize` 的区别，方便你选择?

Revision #4
Created 2025-09-04 02:00:10 UTC by Admin
Updated 2025-09-04 02:29:47 UTC by Admin