iFeve.com

并发编程

研究 · 传播 · 分享

自我介绍

- 并发编程网（ifeve.com）站长
- 支付宝开发工程师，花名清英

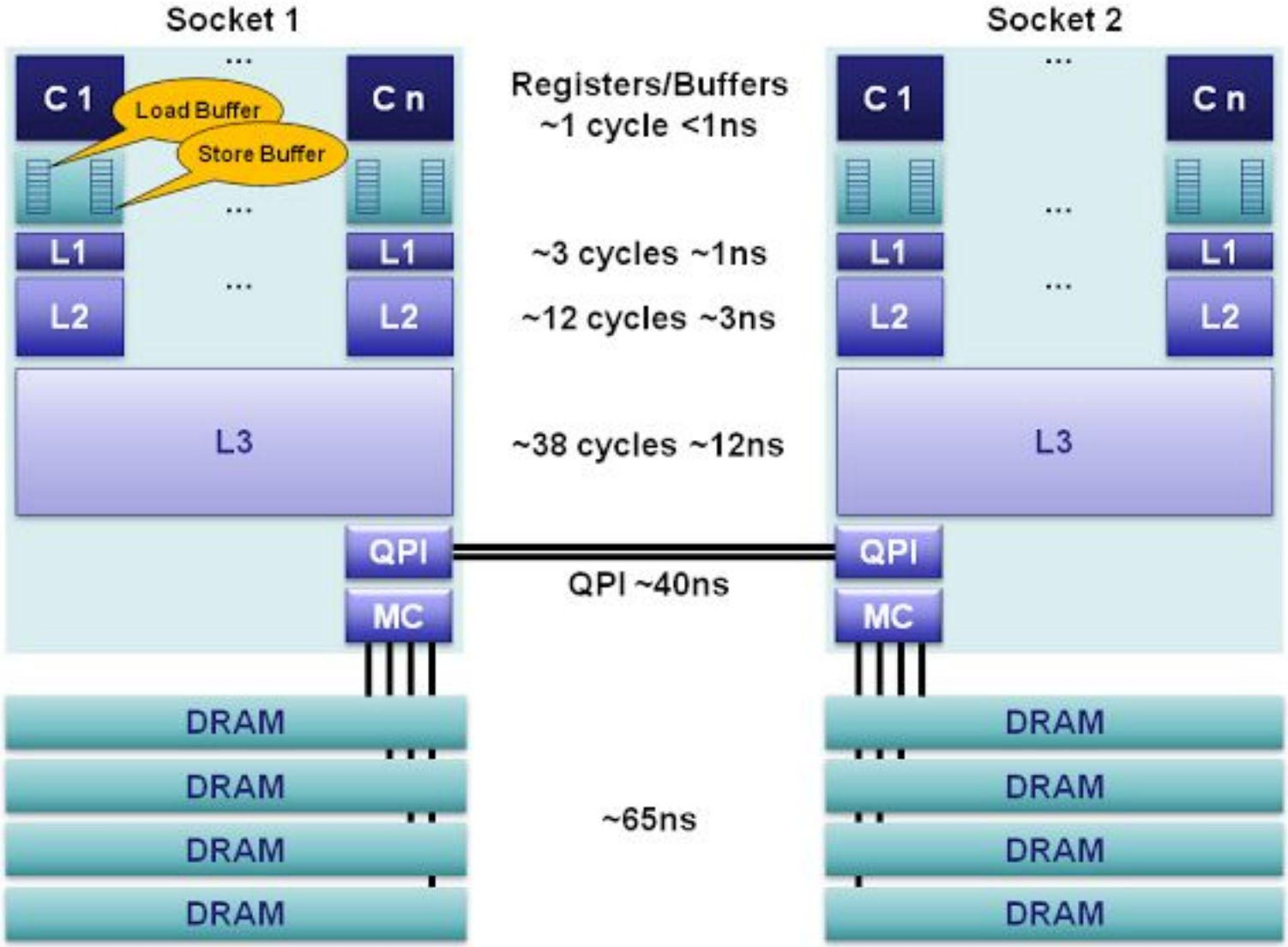
并发编程实战

方腾飞@ifeve

议题

- 并发与CPU的关系
- 让代码并发执行
- 生产者消费者模式介绍
- Java线程池的使用和原理
- 线上并发问题排查

并发与CPU的关系


Socket 1
Socket 2
C 1
C n
Load Buffer
Store Buffer
Registers/Buffers
~1 cycle <1ns
L1
L2
~3 cycles ~1ns
~12 cycles ~3ns
L3
~38 cycles ~12ns
QPI
MC
QPI ~40ns
DRAM
~65ns

让代码并发执行

- 并发执行一定比串行执行快吗？
- 如何让代码最佳并发执行？

串行执行

```
private static void serial() {
    long start = System.currentTimeMillis();
    int a = 0;
    for (long i = 0; i < count; i++) {
        a += 5;
    }
    int b = 0;
    for (long i = 0; i < count; i++) {
        b--;
    }
    long time = System.currentTimeMillis() - start;

    System.out.println("serial:" + time);
```

并发执行

```
private static void concurrency() throws InterruptedException {
    long start = System.currentTimeMillis();
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            int a = 0;
            for (long i = 0; i < count; i++) {
                a += 5;
            }
        }
    });
    thread.start();
    int b = 0;
    for (long i = 0; i < count; i++) {
        b--;
    }
    long time = System.currentTimeMillis() - start;
    thread.join();
    System.out.println("concurrency :" + time);
}
```

执行结果

循环次数	串行执行耗时（单位ms）	并发执行耗时	快多少
一亿	130	77	约一倍
一千万	18	9	约一倍
一百万	5	5	差不多
十万	4	3	慢
一万	0	1	慢

并发执行总结

- 并发执行不一定快，因为有线程创建和调度的时间。
- 大于10ms的任务才考虑并发执行。
- 可以考虑使用线程池，减少线程创建和上下文切换的消耗。
- 无锁编程

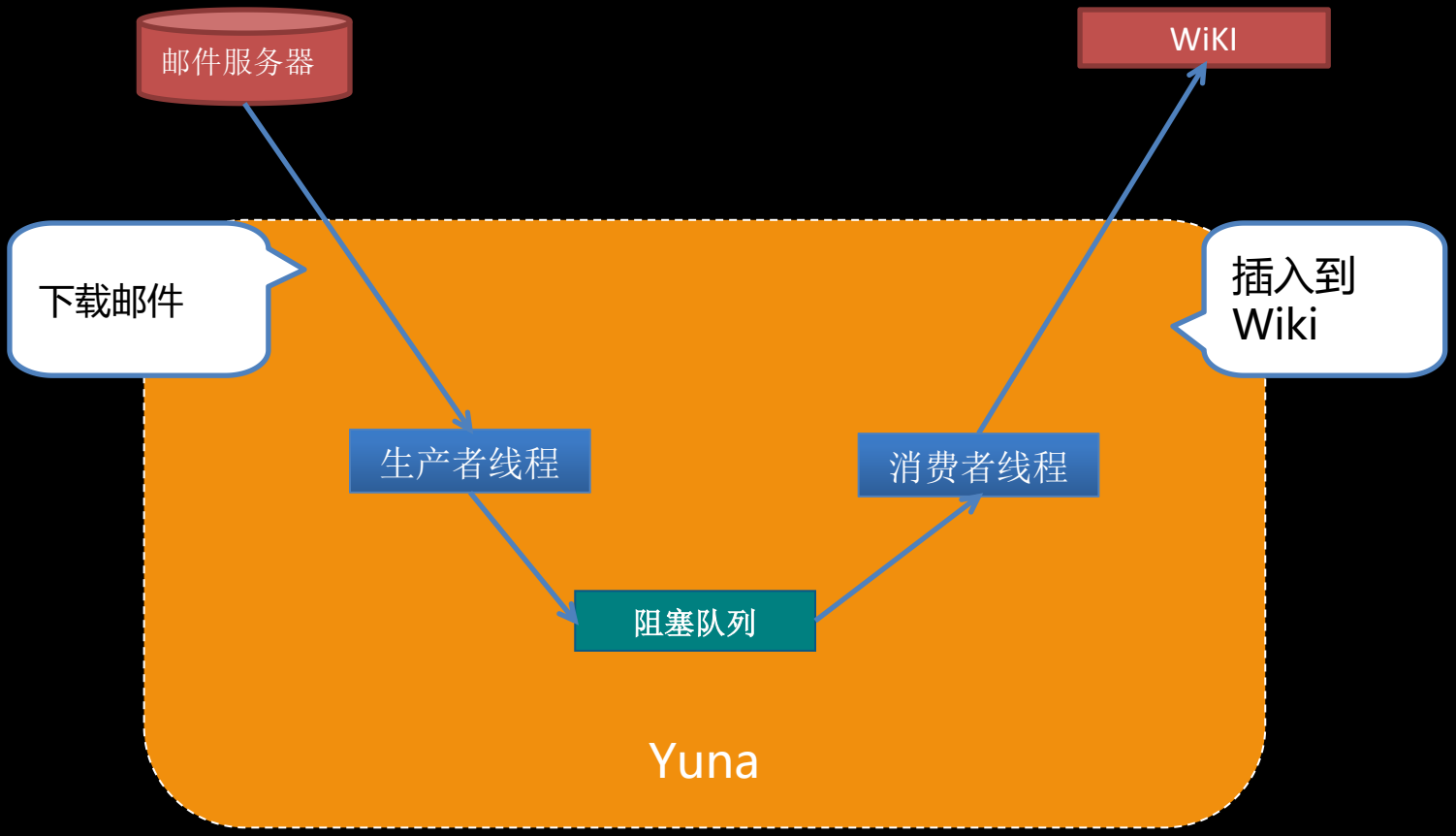
Java包里的并发技巧

- CopyOnWriteArrayList：写时复制
- ConcurrencyHashMap：分段锁，无锁检查
- ThreadPoolExecutor：生产者消费者模式

生产者消费者模式

- 用于平衡生产和消费能力
- 生产者，消费者和阻塞队列
- 经典实现：Java的线程池

生产消费者应用场景—Yuna工具



线程池

- 一组执行任务的线程组成的池
- 降低资源消耗
- 提高响应速度
- 提高线程的可管理性

线程池-实现原理


调用方
创建线程执行任务
存储任务
创建线程执行任务
抛弃任务或其他
基本线程池
阻塞队列
最大线程池
饱和处理者
Java里的线程池

线程池-创建

- new ThreadPoolExecutor(corePoolSize, maximumPoolSize,keepAliveTime, milliseconds,runnableTaskQueue, threadFactory,handler);

线程池-参数说明

- corePoolSize（线程池的基本大小）
- maximumPoolSize（线程池最大大小）
- keepAliveTime（线程的最大存活时间）
- runnableTaskQueue（任务队列）
- threadFactory（线程工厂）
- RejectedExecutionHandler（饱和策略）

线程池参数—阻塞队列（1）

- ArrayBlockingQueue：一个由数组结构组成的有界阻塞队列。
- LinkedBlockingQueue：一个由链表结构组成的有界阻塞队列。
- PriorityBlockingQueue：一个支持优先级排序的无界阻塞队列。
- DelayQueue：一个支持延时获取元素的无界阻塞队列。
- SynchronousQueue：一个不存储元素的阻塞队列。
- LinkedTransferQueue：一个由链表结构组成的无界阻塞队列。
- LinkedBlockingDeque：一个由链表结构组成的双向阻塞队列。

线程池参数—阻塞队列（2）

- 分有界队列和无界队列
- 有界队列用数组实现
- 无界队列用链表实现
- 每个阻塞队列用于不同的应用场景。

线程池参数—饱和策略处理者

- AbortPolicy：直接抛出异常。
- CallerRunsPolicy：直接用调用者线程来运行任务。
- DiscardOldestPolicy：丢弃队列里最近的一个任务，并执行当前任务。
- DiscardPolicy：不处理，直接丢弃掉。
- 自定义：实现RejectedExecutionHandler接口，如记录日志或持久化不能处理的任务。

线程池-关闭

- Shutdown：将线程池的状态设置成SHUTDOWN状态，然后中断所有没有正在执行任务的线程。
- shutdownNow：遍历线程池中的工作线程，然后逐个调用线程的interrupt方法来中断线程，所以无法响应中断的任务可能永远无法终止。

合理的配置线程池

- 任务的性质：CPU密集型任务，IO密集型任务和混合型任务。
- 任务的优先级：高，中和低。
- 任务的执行时间：长，中和短。
- 任务的依赖性：是否依赖其他系统资源，如数据库连接。
- 选择有界阻塞队列。

线程池的统一管理

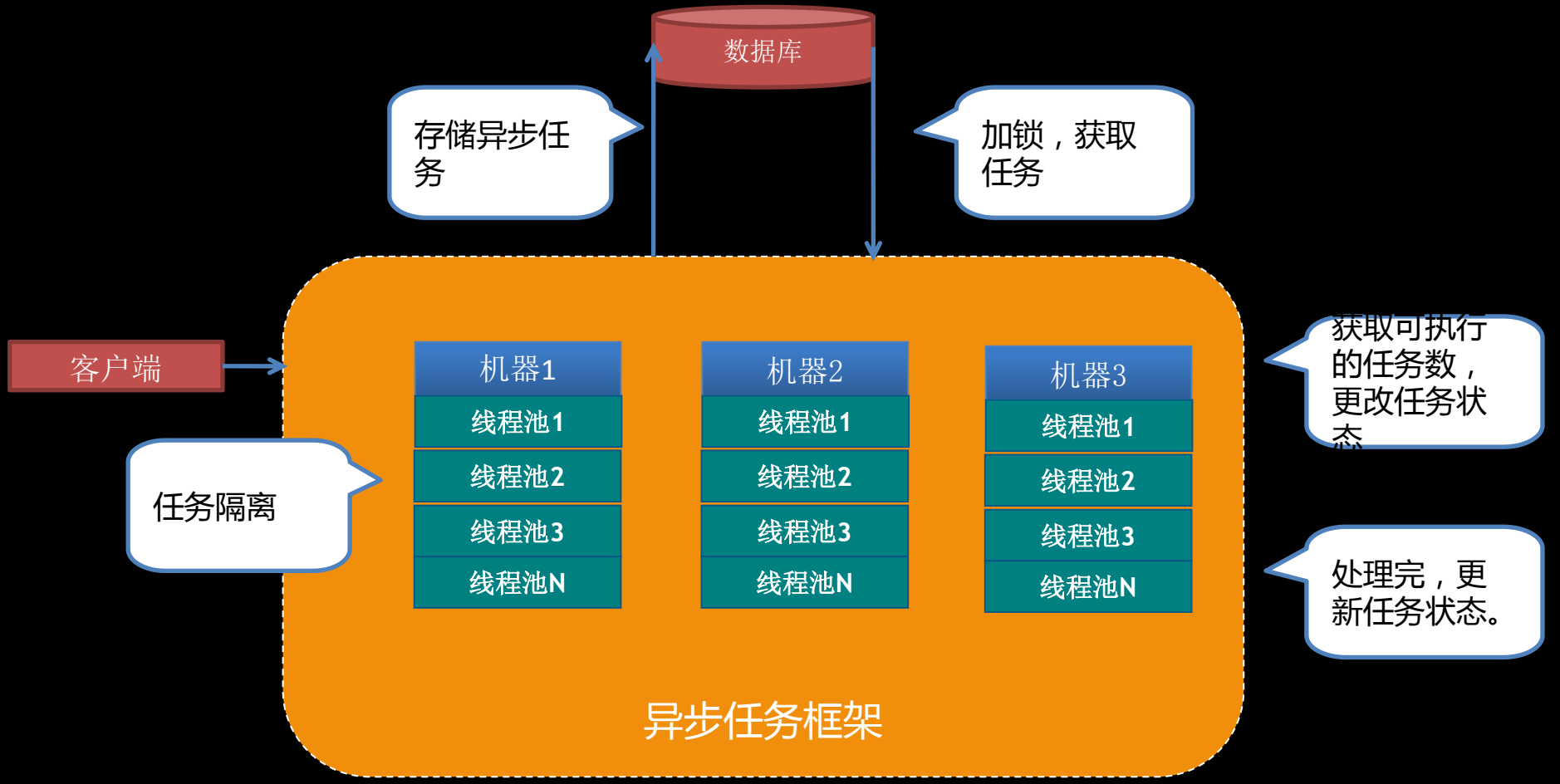
把系统所有的线程池配置放在一起，统一管理。

```
<!-- 精准营销自动生成询盘线程池 -->
<bean id="threadPoolService" class=".eq.sys.threadpool.ThreadPoolServiceImpl" init-method="initThreadPool">
        <property name="corePoolSize" value="50"/>
        <property name="maximumPoolSize" value="100"/>
        <property name="keepAliveTime" value="10"/>
        <property name="queueSize" value="600"/>
</bean>
<!-- 旺旺消息接收服务线程池 -->
<bean id="aliwwThreadPoolService" class="eq.sys.threadpool.ThreadPoolServiceImpl" init-method="initThreadPool">
        <property name="corePoolSize" value="50"/>
        <property name="maximumPoolSize" value="200"/>
        <property name="keepAliveTime" value="5"/>
        <property name="queueSize" value="6000"/>
</bean>

<!-- 反哺精准营销线程池 -->
<bean id="feedPMThreadPoolService" class="eq.sys.threadpool.ThreadPoolServiceImpl" init-method="initThreadPool">
        <property name="corePoolSize" value="50"/>
        <property name="maximumPoolSize" value="100"/>
        <property name="keepAliveTime" value="10"/>
        <property name="queueSize" value="600"/>
</bean>

<!– 通用销线程池 -->
<bean id="feedPMThreadPoolService" class="eq.sys.threadpool.ThreadPoolServiceImpl" init-method="initThreadPool">
        <property name="corePoolSize" value="50"/>
        <property name="maximumPoolSize" value="100"/>
        <property name="keepAliveTime" value="10"/>
        <property name="queueSize" value="600"/>
</bean>
```

线程池的统一监控

线程池名:	旺旺消息接收服务		
核心线程数:	50	当前线程数:	0
任务队列数:	0	活动线程数:	0
已完成执行的任务总数:	0	计划执行的任务总数:	0
允许的最大线程数:	200	曾经同时位于池中的最大线程数:	0
线程池名:	精准营销自动生成询盘		
核心线程数:	50	当前线程数:	0
任务队列数:	0	活动线程数:	0
已完成执行的任务总数:	0	计划执行的任务总数:	0
允许的最大线程数:	100	曾经同时位于池中的最大线程数:	0
线程池名:	反哺精准营销		
核心线程数:	50	当前线程数:	0
任务队列数:	0	活动线程数:	0
已完成执行的任务总数:	0	计划执行的任务总数:	0
允许的最大线程数:	100	曾经同时位于池中的最大线程数:	0

应用场景—异步任务框架


数据库
存储异步任务
加锁，获取任务
客户端
任务隔离
机器1
线程池1
线程池2
线程池3
线程池N
机器2
线程池1
线程池2
线程池3
线程池N
机器3
线程池1
线程池2
线程池3
线程池N
获取可执行的任务数，更改任务状态
处理完，更新任务状态。
异步任务框架

线上问题排查—看CPU性能

```
[tengfei.fangtf@wunion173184.cm4 ~]$ top

top - 22:27:25 up 463 days, 12:46,  1 user,  load average: 11.80, 12.19, 11.79
Tasks: 113 total,   5 running, 108 sleeping,   0 stopped,   0 zombie
Cpu(s): 62.0%us,  2.8%sy,  0.0%ni, 34.3%id,  0.0%wa,  0.0%hi,  0.7%si,  0.2%st
Mem:   7680000k total,  7665504k used,    14496k free,    97268k buffers
Swap:  2096472k total,    14904k used,  2081568k free,  3033060k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
31177 admin     18   0 5351m 4.0g  49m S 301.4 54.0 935:02.08 java
31738 admin     15   0 36432  12m 1052 S  8.7  0.2  11:21.05 nginx-proxy
```

线上问题排查—看CPU性能

```
top - 22:24:50 up 463 days, 12:43,  1 user,  load average: 12.55, 12.27, 11.73
Tasks: 110 total,   3 running, 107 sleeping,   0 stopped,   0 zombie
Cpu0  : 72.4%us,  3.6%sy,  0.0%ni, 22.7%id,  0.0%wa,  0.0%hi,  0.7%si,  0.7%st
Cpu1  : 58.7%us,  4.3%sy,  0.0%ni, 34.3%id,  0.0%wa,  0.0%hi,  2.3%si,  0.3%st
Cpu2  : 53.3%us,  2.6%sy,  0.0%ni, 34.1%id,  0.0%wa,  0.0%hi,  9.6%si,  0.3%st
Cpu3  : 52.7%us,  2.7%sy,  0.0%ni, 25.2%id,  0.0%wa,  0.0%hi, 19.5%si,  0.0%st
Cpu4  : 59.5%us,  2.7%sy,  0.0%ni, 31.2%id,  0.0%wa,  0.0%hi,  6.6%si,  0.0%st
Mem:   7680000k total,  7663152k used,    16848k free,    98068k buffers
Swap:  2096472k total,    14904k used,  2081568k free,  3032636k cached
```

线上问题排查—Top

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
31558	admin	15	0	5351m	4.0g	49m	S	12.2	54.0	10:08.31	java
31561	admin	15	0	5351m	4.0g	49m	R	12.2	54.0	9:45.43	java
31626	admin	15	0	5351m	4.0g	49m	S	11.9	54.0	13:50.21	java
31559	admin	15	0	5351m	4.0g	49m	S	10.9	54.0	5:34.67	java
31612	admin	15	0	5351m	4.0g	49m	S	10.6	54.0	8:42.77	java
31555	admin	15	0	5351m	4.0g	49m	S	10.3	54.0	13:00.55	java
31630	admin	15	0	5351m	4.0g	49m	R	10.3	54.0	4:00.75	java
31646	admin	15	0	5351m	4.0g	49m	S	10.3	54.0	3:19.92	java
31653	admin	15	0	5351m	4.0g	49m	S	10.3	54.0	8:52.90	java
31607	admin	15	0	5351m	4.0g	49m	S	9.9	54.0	14:37.82	java

Jstat分析JVM内存

```
[tengfei.fangtf@wunion173184.cm4 ~]$ sudo /opt/taobao/java/bin/jstat -gcutil 31177 1000 5
  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT
  0.00   1.27  61.30  55.57  59.98  16040  143.775   30   77.692  221.467
  0.00   1.27  95.77  55.57  59.98  16040  143.775   30   77.692  221.467
  1.37   0.00  33.21  55.57  59.98  16041  143.781   30   77.692  221.474
  1.37   0.00  74.96  55.57  59.98  16041  143.781   30   77.692  221.474
  0.00   1.59  22.14  55.57  59.98  16042  143.789   30   77.692  221.481
```

Dump线程

- 线程ID进制转换：

printf "%x\n" 31558

- Dump线程

sudo -u admin /opt/java/bin/jstack 31177 > /home/tengfei.fangtf/dump16

统计线程状态

```
[tengfei.fangtf@wunion173184.cm4 ~]$ grep java.lang.Thread.State dump16 | awk '{print $2$3$4$5}' | sort |
uniq -c
     39 RUNNABLE
     21 TIMED_WAITING(onobjectmonitor)
      6 TIMED_WAITING(parking)
     51 TIMED_WAITING(sleeping)
    305 WAITING(onobjectmonitor)
      3 WAITING(parking)
```

查找问题线程

```
"http-0.0.0.0-7001-97" daemon prio=10 tid=0x000000004f6a8000 nid=0x555e in Object.wait()
[0x0000000052423000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x00000007969b2280> (a org.apache.tomcat.util.net.AprEndpoint$Worker)
        at java.lang.Object.wait(Object.java:485)
        at org.apache.tomcat.util.net.AprEndpoint$Worker.await(AprEndpoint.java:1464)
        - locked <0x00000007969b2280> (a org.apache.tomcat.util.net.AprEndpoint$Worker)
        at org.apache.tomcat.util.net.AprEndpoint$Worker.run(AprEndpoint.java:1489)
        at java.lang.Thread.run(Thread.java:662)
```

谢谢大家

- 提问和交流