## 1.用途

生成虚拟机当前时刻的线程快照，查看各个线程的虚拟机栈。可用于定位线程出现长时间停顿的原因，如死锁、死循环和请求外部资源导致的长时间等待等。

## 2.主要参数

| 参数 | 说明 |
| --- | --- |
| **-l** | 除了堆栈信息外，额外输出关于**锁**的信息 |
| **-F** | 强制输出线程堆栈，比如输出请求不被响应时 |
| **-m** | 调用到本地方法时，可使用-m显示c/c++的堆栈 |

## 3.其他

### 3.1 堆栈信息示例：

所谓堆栈信息类似与跑异常时e.printstack()输出的信息，即虚拟机栈的调用链。示例如下：

```
//表示从Threads开始，直到TcpSocketSender的176行调用                Thread.sleep(5)   开始休眠
"cat-TcpSocketSender" #84 daemon prio=5 os_prio=31 tid=0x00007fd44a532000 nid=0x14603 sleeping[0x0000700014581000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
        at com.dianping.cat.message.io.TcpSocketSender.processMessage(TcpSocketSender.java:176)
        at com.dianping.cat.message.io.TcpSocketSender.run(TcpSocketSender.java:225)
        at java.lang.Thread.run(Thread.java:748)
        at com.dianping.cat.util.Threads$RunnableThread.run(Threads.java:287)

   Locked ownable synchronizers:
        - None
在controller中如果有个地方卡住了了，因为调用了Thread.sleep(xx)方法，如下：
"qtp1758132212-114" #114 prio=5 os_prio=31 tid=0x00007fe1dd04e800 nid=0x12d03 waiting on condition
[0x0000700007a84000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
......//方法名和所在类类名
        at com.sankuai.meituan.waimai.algorithm.controller.UACAuthController.getInfo(UACAuthController.java:64)
......
        at
com.sankuai.meituan.waimai.algorithm.controller.UACAuthController$$EnhancerBySpringCGLIB$$64058469.getInfo(<generated>)

......
        at java.lang.reflect.Method.invoke(Method.java:498)
......
  at org.springframework.web.filter.DelegatingFilterProxy.doFilter(DelegatingFilterProxy.java:263)
......
        at org.eclipse.jetty.util.thread.QueuedThreadPool$3.run(QueuedThreadPool.java:555)
        at java.lang.Thread.run(Thread.java:748)
......
   Locked ownable synchronizers:
        - None
```

### 3.2 死锁

例如在两个线程中分别对a、b加锁，但是顺序不一致，可能导致死锁。死锁堆栈日志和相应代码如下：

```
jstack -l 15940
output:
```

```
Found one Java-level deadlock:
=============================
  //Thread-1在等待Thread-0持有的资源，资源类型（被谁持有的什么类型）
"Thread-1":
  waiting to lock monitor 0x00007fa7de8338a8 (object 0x000000076ace6788, a java.lang.Object),
  which is held by "Thread-0"
  //同上
"Thread-0":
  waiting to lock monitor 0x00007fa7de831018 (object 0x000000076ace6798, a java.lang.Object),
  which is held by "Thread-1"

Java stack information for the threads listed above:
===================================================
"Thread-1":
//发生死锁的位置
        at com.sankuai.meituan.waimai.algorithm.util.Test$Task2.run(Test.java:33)
        - waiting to lock <0x000000076ace6788> (a java.lang.Object)
        - locked <0x000000076ace6798> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:748)
"Thread-0":
        at com.sankuai.meituan.waimai.algorithm.util.Test$Task1.run(Test.java:21)
        - waiting to lock <0x000000076ace6798> (a java.lang.Object)
        - locked <0x000000076ace6788> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:748)

Found 1 deadlock.


//代码
public class Test {
    static Object obj1=new Object();
    static Object obj2=new Object();

    static class Task1 implements Runnable{
        @Override
        public void run() {
            try{
                synchronized (obj1){
                    Thread.sleep(1000);
                    synchronized (obj2){ }
                }
            }catch(Exception e){}
        }
    }

    static class Task2 implements Runnable{
        @Override
        public void run() {
            try{
                synchronized (obj2){
```

```
                Thread.sleep(1000);
                synchronized (obj1){ }
            }
        }catch(Exception e){}
    }
}

public static void main(String[] args) throws InterruptedException {
    new Thread(new Task1()).start();
    new Thread(new Task2()).start();
}
}
```