# 概述

## 1.1什么是arthas

arthas是阿里巴巴开源的一款java诊断工具，它可以帮助我们解决

1. 查找类是从哪个jar包中加载的
2. 有部分问题是在开发和测试环境中难以复现的，但实际生产中不可能加日志后再发布到生产环境
3. 查看系统的运行状态
4. 监控JVM运行状态
5. 如何快速定位应用的热点在哪，绘制火焰图

## 1.2运行要求

arthas必须运行在JDK6+以上的版本，采用java语言编写，因此支持多系统跨平台使用 ，采用命令行交互

## 1.3安装

arthas的安装主要分为在线安装和离线安装

arthas在启动前会检测本地是否有运行的java进程，如果没有java进程，arthas是无法启动的

### 1.3.1windows和Linux在线安装

mkdir 指定安装目录

cd 指定安装目录

curl -o <https://alibaba.github.io/arthas/arthas-boot.jar>

java -jar arthas-boot.jar 执行这条命令之前必须保证有java进程正在运行

如果本地没有安装arthas，接下按数字键来选则一个java进程，就会自动去下载arthas

安装完毕，自动进入arthas命令行

### 1.3.2离线安装

从maven仓库下载全量安装包:

https://maven.aliyun.com/repository/public/com/taobao/arthas/arthas-packaging/3.1.7/arthas-packaging-3.1.7-bin.zip

解压到指定目录

## 1.4卸载

linux下：

rm -rf ~/.arthas/

rm -rf ~/logs/arthas

windows下：

直接删除C:/User/Administrator/.arthas

和C:/User/Administrator/logs/arthas

# 使用

## 2.1启动和attach

在启动的时候，按数字键选择要attach的java进程，可以通过命令行去直接操作，也可以打开浏览器访问ip:3658来访问

## 2.2基础命令

### 2.2.1 help帮助命令

help显示命令的帮助信息

### 2.2.2 pwd查看当前路径

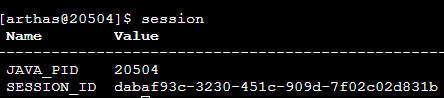
### 2.2.3 cat查看文件内容

### 2.2.4 grep过滤查找

|  |  |  |
| --- | --- | --- |
| **参数** | **作用** | **示例** |
| -n | 显示行号 | help |grep mjad -n |
| -i | 忽略大小写查找 | help |grep MJad -n -i |
| -m 行数 | 最大显示行数， | help |grep sys -m 10 |
| -e 正则表达式 | - | - |

### 2.2.5 cls清除屏幕内容

### 2.2.6 session查看当前会话信息



### 2.2.7 reset重置增强类

arthas允许我们在运行过程中，对某个类进行增强，额外增加某些功能，这就是增强类，其常用reset命令如下：

reset <类名> 还原指定增强类

reset \*<后缀名> 还原以后缀名为结尾的增强类

reset 还原所有增强类

### 2.2.8 version查看arthas版本号

### 2.2.9 history打印历史命令

### 2.2.9 quit退出当前会话(不影响其他客户端)

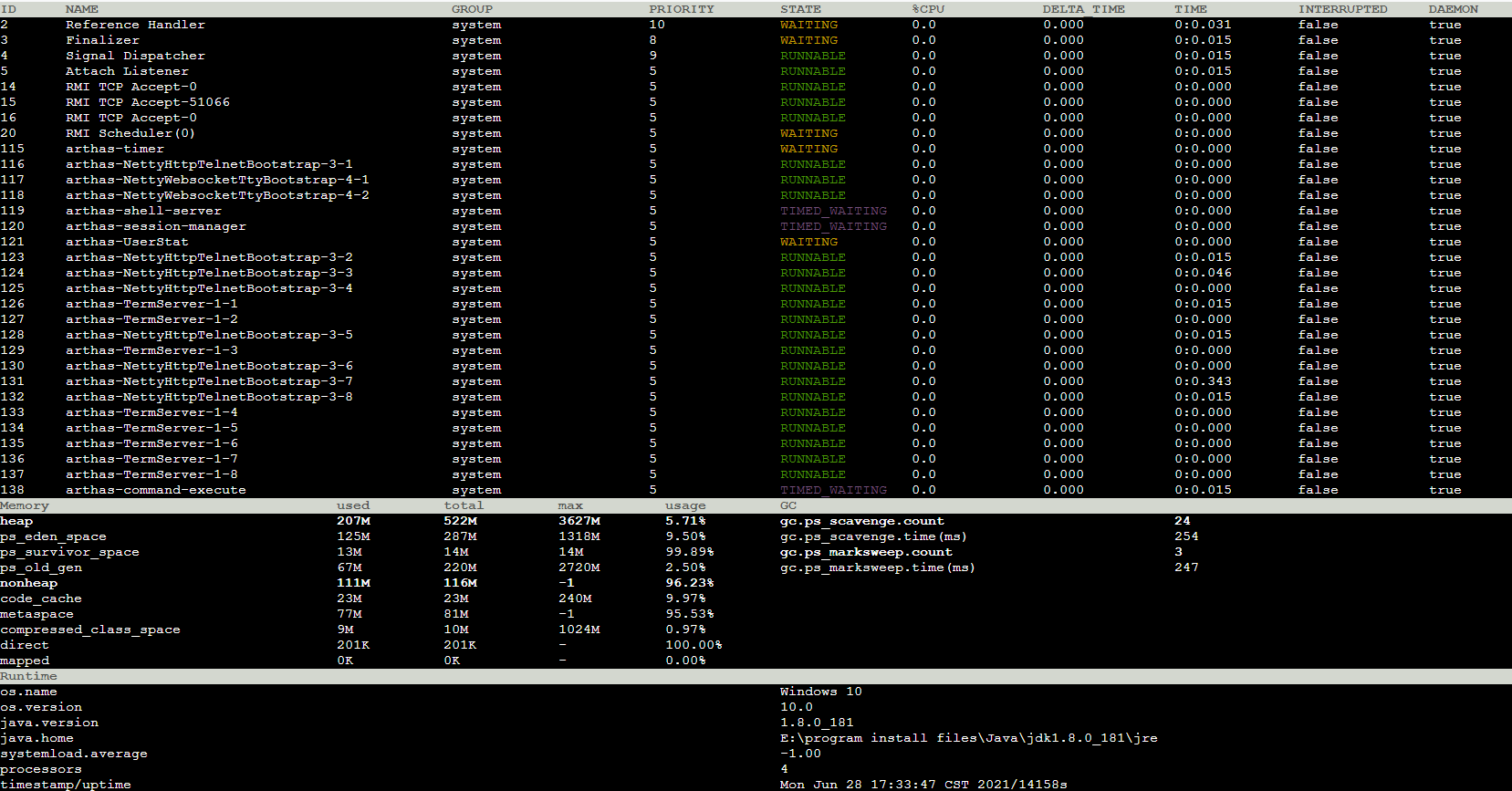
### 2.2.10 stop停止arthas服务(其他客户端也关闭)

### 2.2.11 keymap查看快捷键列表，以及自定义快捷键

## 2.3 jvm相关命令

### 2.3.1dashboard仪表板

输入dashboard，按回车键，查看仪表盘，仪表盘是不断刷新的，可以按q或者ctrl+c中断仪表盘

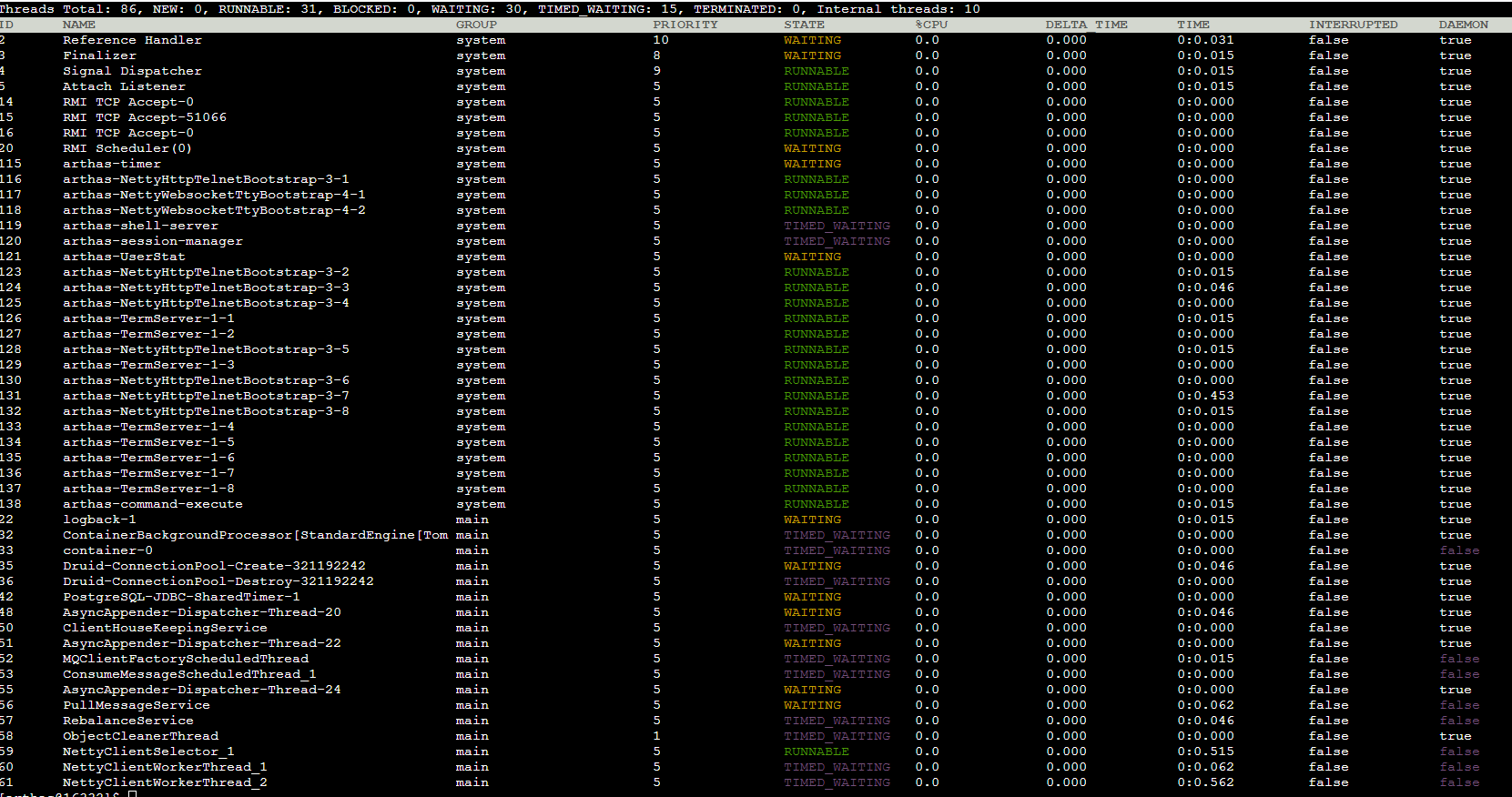


我们可以看到仪表盘主要分为三部分：

1. JVM线程信息：展示JVM线程的id，name，线程组名，线程优先级，状态，CPU占比，上次采样后线程运行的增量CPU时间，线程运行的总CPU时间，线程中断位状态，是否守护线程等信息
2. JVM内存使用情况：主要分为了heap堆内存和nonheap非堆内存来展示
3. 系统环境信息

### 2.3.2thread查看线程情况

输入thread查看线程情况

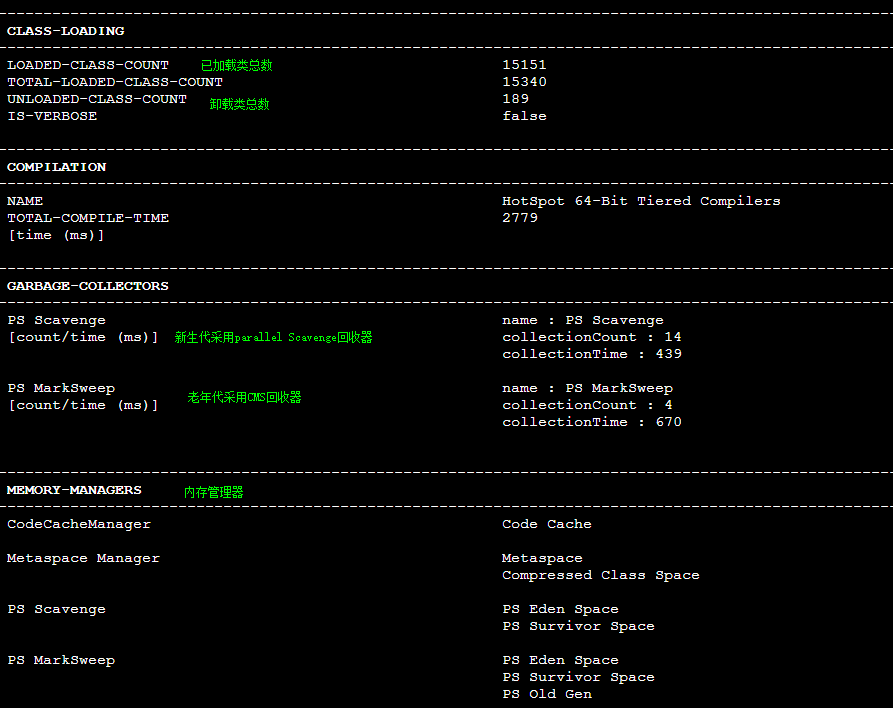


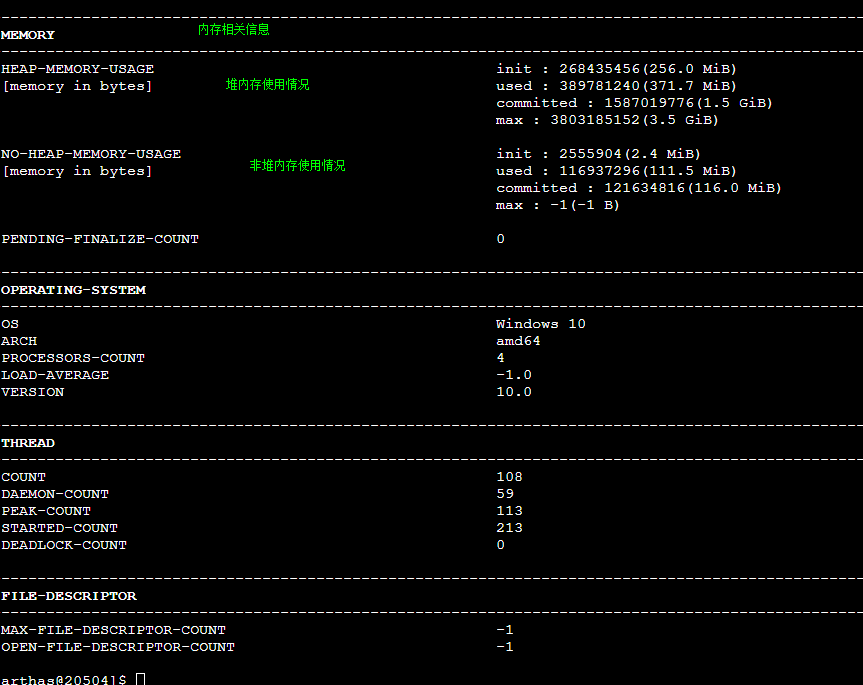
如果想要详细查看某个线程信息，可以输入thread <ID>，如想看arthas-timer线程详情，那么执行thread 115即可

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| thread -n <数字> | 打印最忙的几个线程 | thread -n 6 |
| thread -b | 打印当前阻塞其他线程的线程**（可以用来检查线程死锁）** | thread -b |
| thread --state <状态> | 查看某种状态的线程 | thread --state WAITING |
| thread -i <毫秒数> | 指定cpu采样间隔，单位毫秒 | thread -i 1000 |

### 2.3.3 jvm命令

查看当前JVM信息，这个命令会打印很多信息，这里只挑选一部分说





### 2.3.4 sysprop查看修改jvm的系统属性

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| sysprop | 查看所有jvm属性 | sysprop |
| sysprop <属性名> | 查看指定jvm属性 | sysprop sun.desktop |
| sysprop <属性名> <值> | 修改指定属性 | sysprop user.country US |

### 2.3.5 sysenv查看当前jvm环境变量

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| sysenv | 查看所有jvm环境变量 | sysenv |
| sysenv <属性名> | 查看指定jvm环境变量 | - |

### 2.3.6 vmoption查看、更新jvm参数

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| vmoption | 查看所有jvm属性 | vmoption |
| vmoption <属性名> | 查看指定jvm属性 | vmoption HeapDumpBeforeFullGC |
| vmoption <属性名> <值> | 修改指定属性 | vmoption HeapDumpBeforeFullGC true |

### 2.3.7 getstatic查看类的静态属性

通过getstatic命令可以直接查看类在运行过程中的静态属性

getstatic <类名> <属性名>，eg：

getstatic com.cetc52.ycsx.constants.VideoSystemApiConstants LOGIN

## 2.4 class/classLoader相关命令

### 2.4.1 sc查看已加载类信息

查看jvm中已加载的类信息。是search-class的简写。sc默认开启了子类匹配功能，即不仅能看到某个类，也会显示其子类，如果不希望看到子类，可以打开optionsdisable-sub-class true开关

参数：

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| sc <类名表达式> | 类名表达式匹配，支持全限定名格式如com.taobao.Test，也支持com/tobao/test格式，还支持\* | sc com.cetc52.ycsx.controller.OrganizationController |
| sc -d | 输出当前类的详细信息。如果一个类被多个ClassLoader加载，则会出现多次 | sc com.cetc52.ycsx.controller.OrganizationController -d |
| sc -E | 开启正则表达式。默认状态下是通配符表达式 | - |
| sc -f | 输出当前类的成员变量信息（需要配合-d一起使用） | sc com.cetc52.ycsx.controller.OrganizationController -d -f |

### 2.4.2 sm查看已加载类信息

查看已加载类的方法信息

参数：

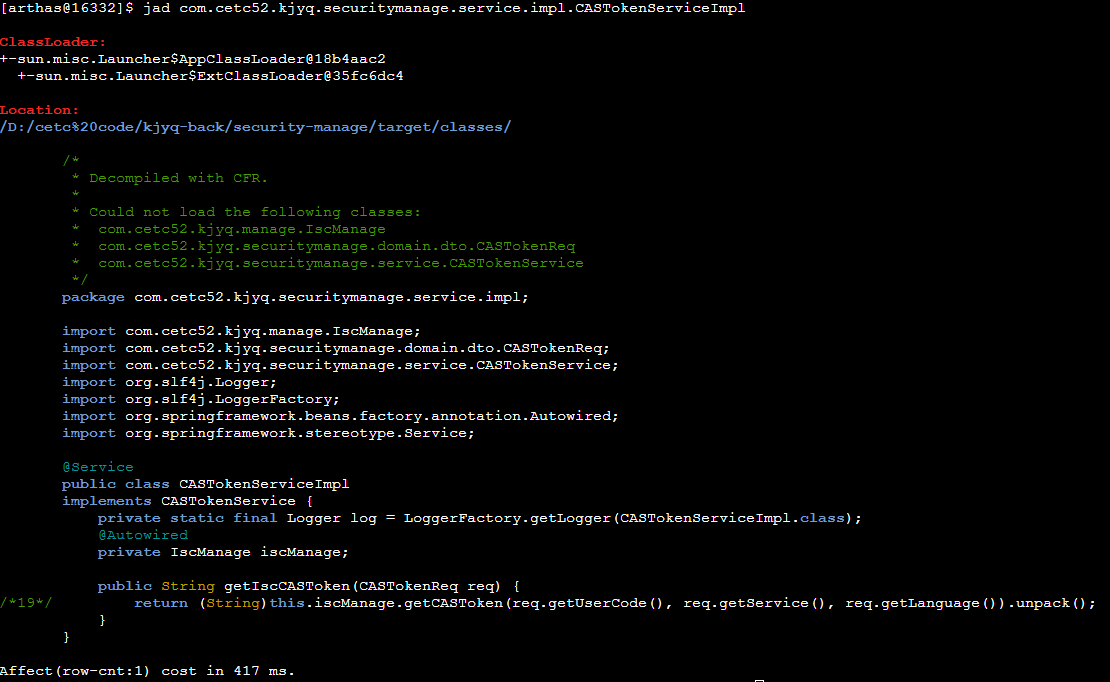
|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| sm <类名表达式> | 类名表达式匹配，支持全限定名格式如com.taobao.Test，也支持com/tobao/test格式，还支持\* | sm com.cetc52.ycsx.controller.OrganizationController |
| sm -d | 输出当前类方法的详细信息。如果一个类被多个ClassLoader加载，则会出现多次 | sm com.cetc52.ycsx.controller.OrganizationController -d |
| sm -E | 开启正则表达式。默认状态下是通配符表达式 | - |

### 2.4.3通过jad反编译class文件

jad com.cetc52.kjyq.securitymanage.service.impl

将jvm中实际运行的class的字节码反编译成java代码，也可以在类的全限定名后面指定某个方法，那么只会显示该方法的反编译结果，如：

jad com.cetc52.kjyq.securitymanage.service.impl getIscCASToken



### 2.3.4 mc编译命令

类似javac命令，在内存中将.java文件编译成.class文件，eg：

mc /root/kjyq/src/main/java/com/cetc52/kjyq/securitymanage/service/impl/CASTokenServiceImpl.java

加上-d后，可以指定.class文件输出目录

mc /root/kjyq/src/main/java/com/cetc52/kjyq/securitymanage/service/impl/CASTokenServiceImpl.java -d /root

### 2.3.5 redefine加载class文件

加载外部的.class文件，redefine到jvm中去。

1. **redefine后原来的类无法恢复，reset命令也无效**
2. **redefine可能失败，比如增加了新的field属性或者新增了新的方法，redefine只能修改已有的属性或方法**
3. **redefine和jad/watch/trace/monitor/tt命令冲突。如果执行完redefine后，再执行上述命令，则会将redefine的字节码重置，即修改后的字节码无效了**
4. **正在执行的函数，没有退出前不能生效（比如线程，一般就不会生效，因为线程一般是死循环）**

### 2.3.6 dump保存已加载的字节码

dump的作用是将已加载的字节码文件保存到指定目录下，即logs/arthas/classdump/目录下

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| dump <类的全限定名> | 保存已加载字节码到指定目录 | dump java.lang.String |
| dump -c <hashcode> | 类所属ClassLoader的hashcode | dump -c 3d4ec69 java.lang.String |
| dump -E | 开启正则表达式 | vmoption HeapDumpBeforeFullGC true |

### 2.3.7 classloader获取类加载器信息

classloader主要有两个作用：

1：classloader命令可以统计jvm中所有类加载器的统计信息，并可以展示继承树，urls等

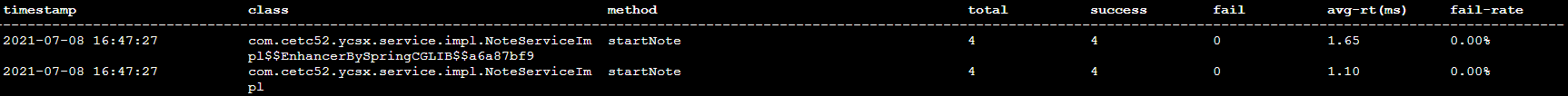
2：可以让指定的classloader去getResources，打印出所有查找到的resource的url。可以用于解决ResourceNotFoundException

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| classloader -l（小写L） | 按类加载实例进行统计 | classloader -l |
| classloader -t | 打印所有classloader的继承树（类加载器的继承关系） | classloader -t |
| classloader -a | 列出所有被classloader加载的类（谨慎使用） | classloader -a |
| classloader -c <hashcode> | 指定某个类加载器的hashcode，以此查看该加载器加载的类在哪些jar包里 | classloader -c 18b4aac2 |
| classloader -c <hashcode> -r <资源路径> | 在-c的前提下再查找-r列出的资源 | classloader -c 18b4aac2 -r java/lang/String.class |
| classloader -c <hashcode> -load | 使用指定类加载器加载某个指定类 | classloader -c 18b4aac2 -load java.lang.String |

## 2.5 monitor命令统计指定方法执行情况

监控指定类中方法的执行情况

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| monitor <类名> <方法名> | 按类加载实例进行统计 | monitor com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| monitor -c <时间：秒> <类名> <方法名> | 设置统计周期，默认120s | monitor -c 5 com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |



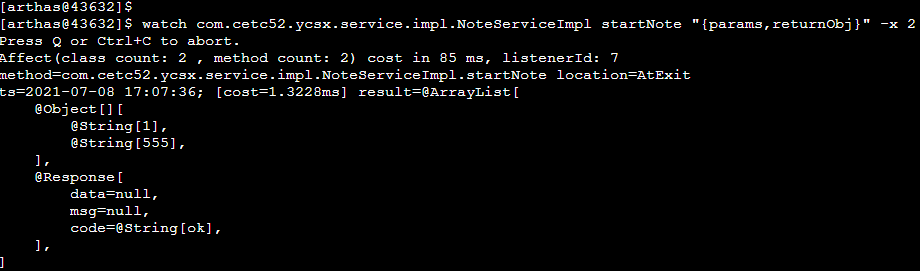
## 2.6 watch命令监控方法参数返回值

它可以监视程序在执行过程中，输入参数、返回值、抛出的异常是什么

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| watch <类名> <方法名> | 按类加载实例进行统计，相当于使用了-f | watch com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| watch <类名> <方法名> <观察表达式> | 观察表达式使用了OGNL表达式，params代表参数列表，returnObj代表返回值 | watch com.cetc52.ycsx.service.impl.NoteServiceImpl startNote “{params,returnObj}” |
| watch -b <类名> <方法名> | 在方法调用之前观察，这种方式获取不到方法的返回值，因此返回值都为null。默认关闭 | watch -b com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| watch -e <类名> <方法名> | 在方法异常之后观察。默认关闭 | watch -e com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| watch -s <类名> <方法名> | 在方法返回之后观察。默认关闭 | watch -s com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| watch -f <类名> <方法名> | 在方法结束之后观察。默认打开 | watch -f com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| watch <类名> <方法名> -x <属性深度> | 用于指定要打印出来的参数/返回值中属性展示的深度，默认为1 | watch com.cetc52.ycsx.service.impl.NoteServiceImpl startNote “{params,returnObj}” -x 2 |



因为这个方法使用了动态代理，所以会有两个method，上面展示了方法执行时间等信息

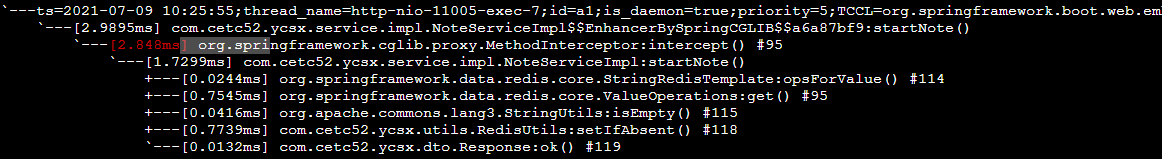


上图设置了属性深度为2（-x 2）

## 2.7 trace命令打印方法内部调用路径及耗时

trace命令可以打印方法内部的调用路径，并输出路径上每个节点的耗时，并且用红色标出最耗时的部分

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| trace <类名> <方法名> | 打印调用栈及其每一步耗时 | trace com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| trace <类名> <方法名> <条件表达式> | 条件表达式使用了OGNL表达式 |  |
| trace <类名> <方法名> -n <次数> | 设置命令执行次数，只打印n次后，退出trace命令 | trace com.cetc52.ycsx.service.impl.NoteServiceImpl startNote -n 2 |
| trace <类名> <方法名> #cost | 方法执行耗时，可以用来过来符合表达式的栈内容 | trace com.cetc52.ycsx.service.impl.NoteServiceImpl startNote ‘#cost > 1’-n 2如果执行过程中有大于1毫秒的，就打印出来 |



## 2.8 stack命令打印当前方法被调用的路径

stack命令可以打印方法被调用的调用栈

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| stack <类名> <方法名> | 打印方法被调用的调用栈 | stack com.cetc52.ycsx.service.impl.NoteServiceImpl startNote |
| stack <类名> <方法名> <条件表达式> | 条件表达式使用了OGNL表达式 |  |
| stack <类名> <方法名> -n <次数> | 设置命令执行次数，只打印n次后，退出stack 命令 | stack com.cetc52.ycsx.service.impl.NoteServiceImpl startNote -n 2 |

## 2. profiler命令生成火焰图

生成应用热点的火焰图，本质上是通过不断采样，将采样后的结果生成火焰图。默认情况下是生成cpu火焰图

profiler 命令 参数

|  |  |  |
| --- | --- | --- |
| **命令** | **含义** | **示例** |
| profiler start | 启动火焰图采样 | profiler start |
| profiler list | 列出profiler支持的事件 | profiler list |
| profiler getSamples | 获取当前已采样的样本总数，随着时间推移，这个采样总数会越来越大 | profiler getSamples |
| profiler status | 显示当前已经运行时间 | profiler status |
| profiler stop | 停止记录，生成svg格式的火焰图 | profiler stop |
| profiler stop --format html | 停止记录，生成html格式的火焰图 | profiler stop --format html |