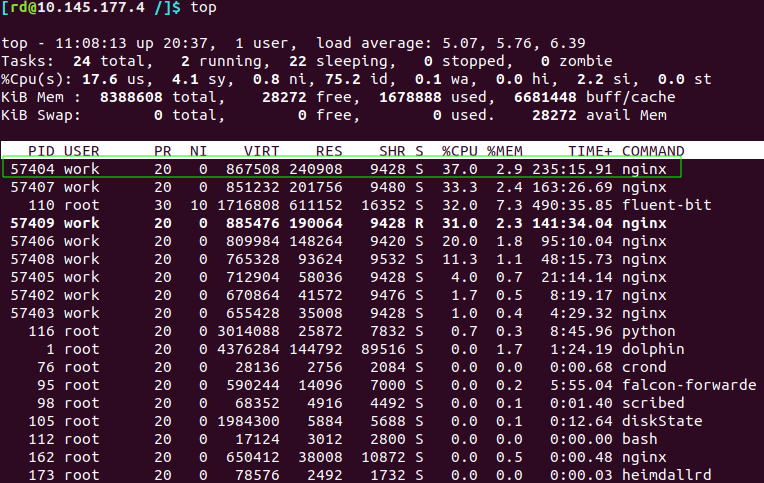
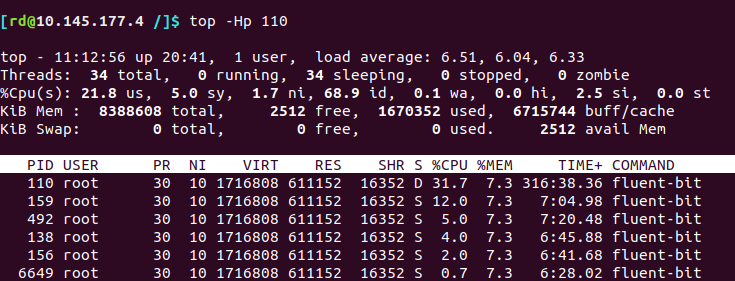
# 1.Top



# 2.Top -HP 110



# 3. ps aux| grep java| grep user-data

查看puri-user-data的Java服务的进程号

# 4.jstack 21118（进程号） | grep 'puri'

查看所有线程名为puri线程（线程名与项目全路径相关。我们的项目都是以puri开头，因此grep ‘puri’）的执行状态

图形用户界面, 文本, 应用程序

描述已自动生成

# 4文件重定向 >

jstack 21118 | grep 'puri' > jstack.log

# 5.远程文件拷贝（从本机到远端10.232.66.95）

scp puri-user-data.log [doqinvera@10.232.66.95:/Users/doqinvera/Desktop](mailto:doqinvera@10.232.66.95:/Users/doqinvera/Desktop)

# 6.OOM分析

如果发现正常服务无法提供服务时，问题排查步骤：

1. **查看最后一条业务日志：**

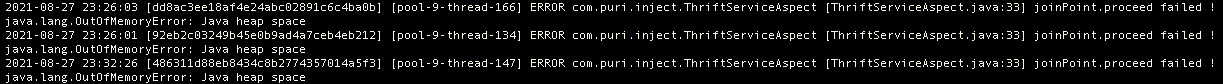
**tail -n 10 mt-application.log**

发现最后一条业务日志的**时间**是昨晚10点左右。这意味着自昨晚10点以后，mt项目就无法提供服务了。

1. **查看最后一条错误日志：**

**tail -n 10 mt-application.err.log**

发现最后一条错误日志是：java.lang.OutOfMemoryError:Java heap space



由于内存溢出，导致服务无法正常提供

1. 分析JVM中的大对象(由于服务当时已经重启，当时的内存状态丢失。因此此处仅以JVM排查命令演示)

**jps (查看服务进程号)**

**jmap -histo 11784|head -n 50**

文本

描述已自动生成

1. 查看内存JC的情况

**jstat -gcutil 11784 1000电脑萤幕

中度可信度描述已自动生成**

1. 查看调用栈

**jstack 11749**

# 7. jmap -histo 11784|head -n 50

<https://www.jianshu.com/p/a4ad53179df3>

文本

描述已自动生成文本

描述已自动生成

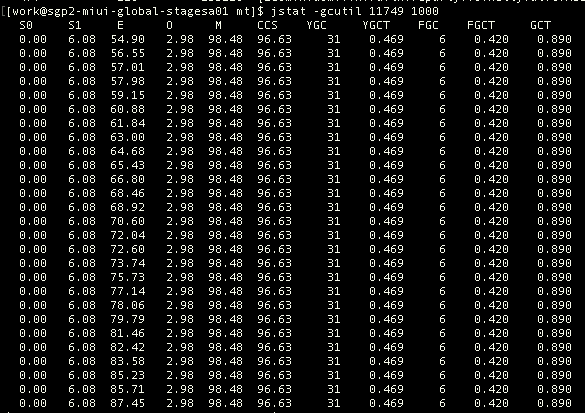
|  |  |
| --- | --- |
| num | 内存占用排名，值越小，表示总占用内存越大 |
| instances | 存活实例的个数 |
| bytes | 占用的内存大小(B) |
| class name | 占用的类 |

# 8. jstat -gcutil 11749 1000(每隔1s=1000 ms统计一次进程11749 的GC情况)

jstat -options [pid] [统计频率]

<https://blog.csdn.net/zhaozheng7758/article/details/8623549>

|  |  |
| --- | --- |
| class | (类加载器) |
| compiler | JIT |
| gc | GC堆状态 |
| gccapacity | 各区大小 |
| gccause | 最近一次GC统计和原因 |
| gcnew | 新区统计 |
| gcnewcapacity | 新区大小 |
| gcold | 老区统计 |
| gcoldcapacity | 老区大小 |
| gcpermcapacity | 永久区大小 |
| **gcutil** | **GC统计汇总** |
| printcompilation | HotSpot编译统计 |



|  |  |
| --- | --- |
| S0 | 年轻代中第一个survivor（幸存区）已使用的占当前容量百分比 |
| S1 | 年轻代中第二个survivor（幸存区）已使用的占当前容量百分比 |
| E | 年轻代中Eden已使用的占当前容量百分比 |
| O | old代已使用的占当前容量百分比 |
| P | perm代已使用的占当前容量百分比 |
| YGC | 从应用程序启动到采样时年轻代中gc次数 |
| YGCT | 从应用程序启动到采样时年轻代中gc所用时间(s) |
| FGC | 从应用程序启动到采样时old代(全gc)gc次数 |
| FGCT | 从应用程序启动到采样时old代(全gc)gc所用时间(s) |
| GCT | 从应用程序启动到采样时gc用的总时间(s) |

# 9. jvm内存模型

https://blog.csdn.net/wwt1996/article/details/105762516

日程表

描述已自动生成

|  |  |  |  |
| --- | --- | --- | --- |
| Young(年轻代) | Eden | From Survivor | to Survivor |
|  | 新的对象实例总是首先放在Eden区 | Eden区和 Tenure(终生代)的缓冲，可以向Tenure(终生代)转移活动的对象实例。 | |
| Old(老年代) | Tenure |  | |
|  | 存放生命周期长久的实例对象 |  | |
| Perm(永久代) | 非堆内存的组成部分。主要存放加载的Class类级对象如class本身，method，field等等 |  | |

|  |  |  |
| --- | --- | --- |
| Young GC | 绝大部分对象在Eden区生成，当Eden区装填满的时候，会触发Young GC | 在Eden区实现清除策略。   * 没有被引用的对象则直接回收。 * 存活的对象会被移送到Survivor区。Survivor 区分为S0和S1两块内存空间，送到哪块空间呢?每次Young GC的时候，将存活的对象复制到未使用的那块空间，然后将当前正在使用的空间完全清除，交换两块空间的使用状态。 * 如果Young GC要移送的对象大于Survivor区容量上限，则直接移交给老年代。   每个对象都有一个计数器，每次YGC都会加1。参数能配置计数器能控制移交次数，默认值是15，意思就是在Survivor 区交换14次之后，晋升至老年代。 |
| Full GC |  | * Survivor 区处理超过上限，老年代也承受时，则会触发Full Garbage Collection(Full GC) * 如果依然无法放下，则抛OutOfMemoryError，堆出现OOM的概率是所有内存耗尽异常中最高的。 |

-Xms设置堆的最小空间大小。

-Xmx设置堆的最大空间大小。

-XX:NewSize设置新生代最小空间大小。

-XX:MaxNewSize设置新生代最大空间大小。

-XX:PermSize设置永久代最小空间大小。

-XX:MaxPermSize设置永久代最大空间大小。

-Xss设置每个线程的堆栈大小