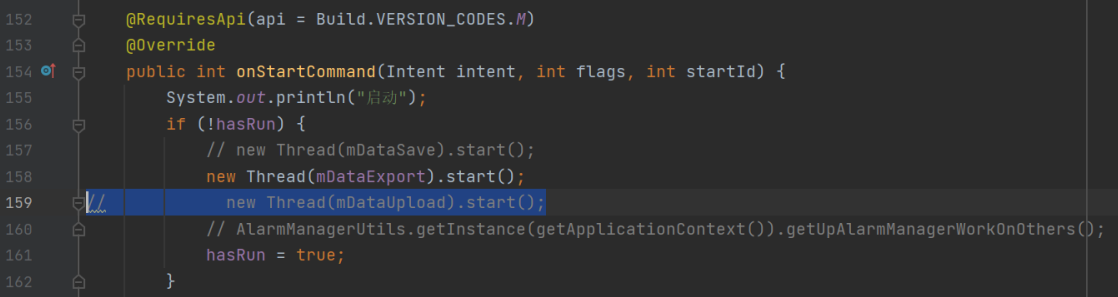
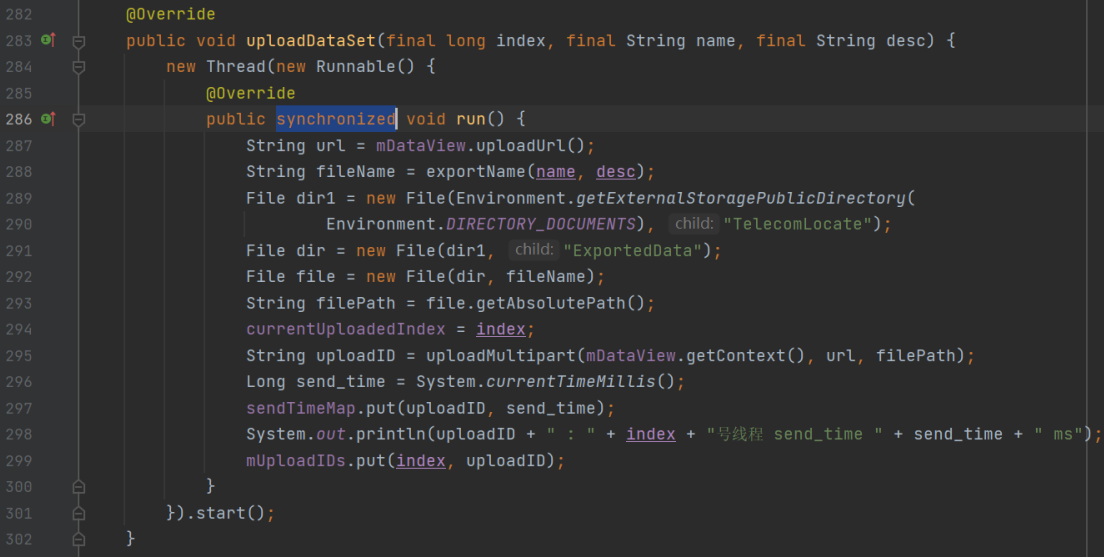
1. **实现客户端的并发请求**
2. 注释掉原先单个请求自动upload的代码语句。



1. 在uploadDataSet方法中进行修改，记录每一个线程的send\_time，并添加输出提示信息。

对uploadDataSet方法添加synchronized关键字，用来控制线程同步，保证线程在多线程环境下，不被多个线程同时执行，确保数据的完整性。



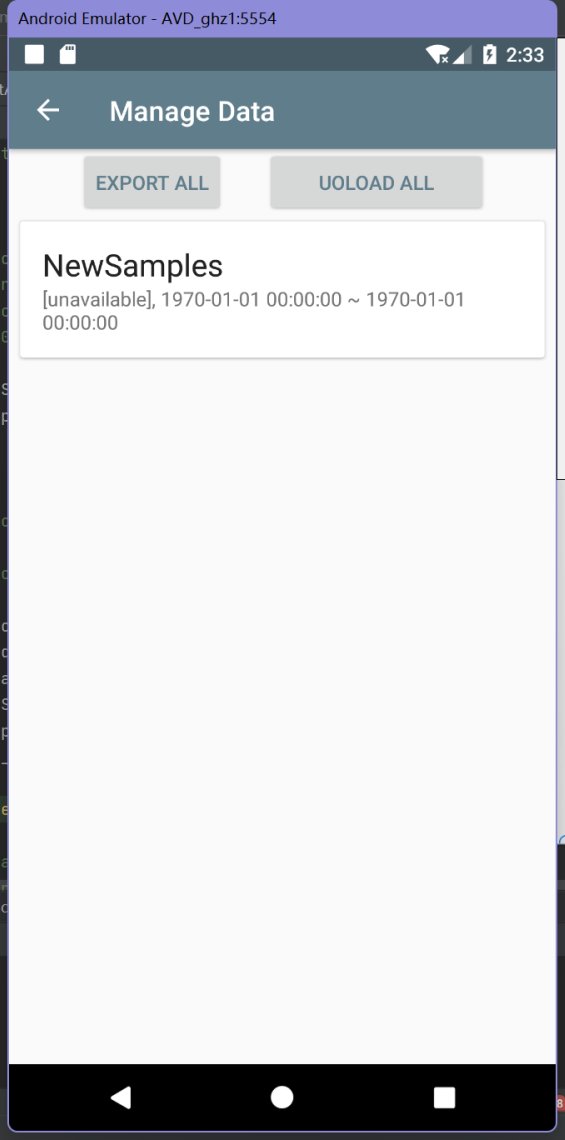
1. 添加exportAllDatasets和uploadAllDatasets两个方法接口，分别用来实现当前所有数据的export和upload操作。



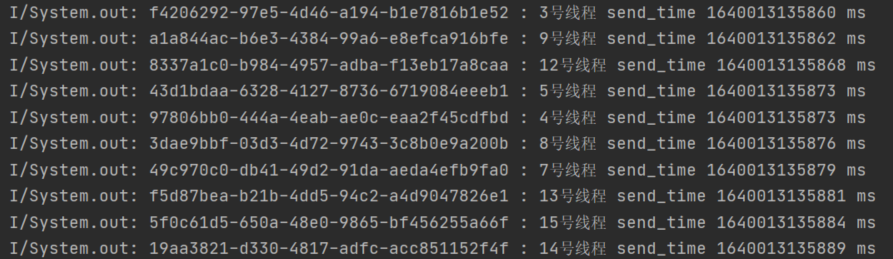
1. 在视图层添加export all和upload all两个按钮。



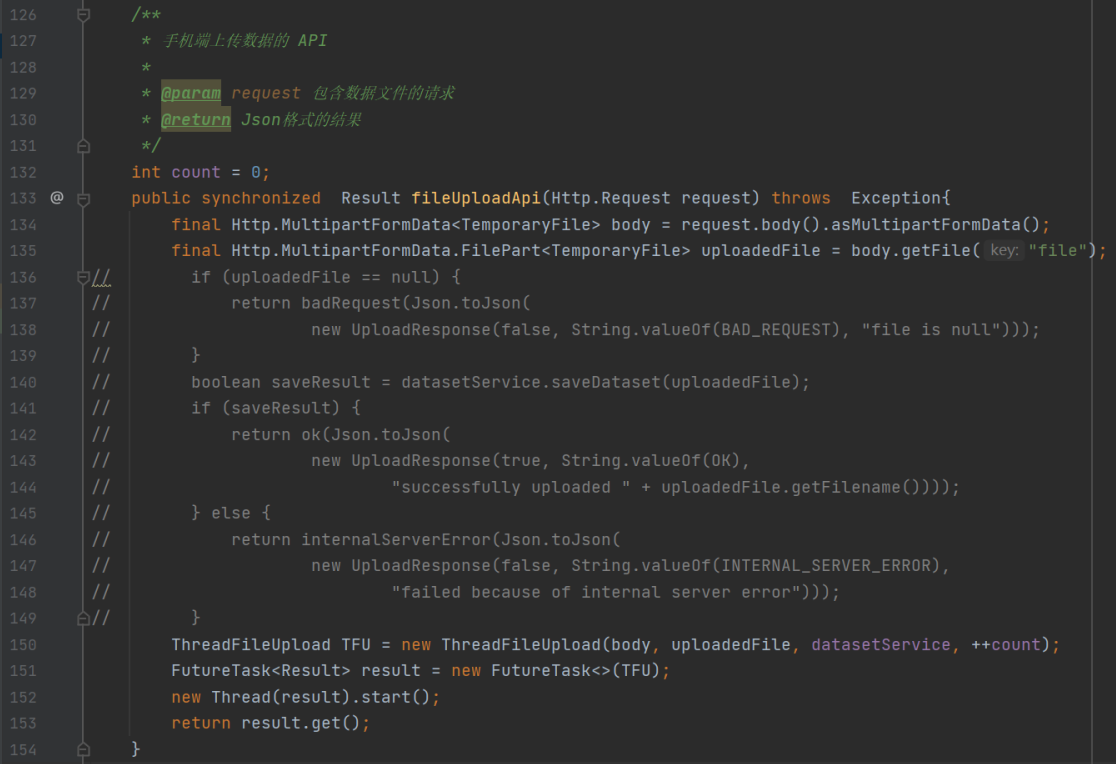
1. export all和upload all按钮分别绑定exportAllDatasets和uploadAllDatasets事件。这样之后，依次点击如下界面上的两个按钮，便可触发当前所有数据的export和upload，实现多线程的创建。



1. 测试结果：客户端成功实现并发请求，在打印信息中可以看到各线程的sent\_time（以ms为单位），而线程的序号index（取决于创建时间）的排列是无序的，从而可发现多线程执行的确实是无序的。



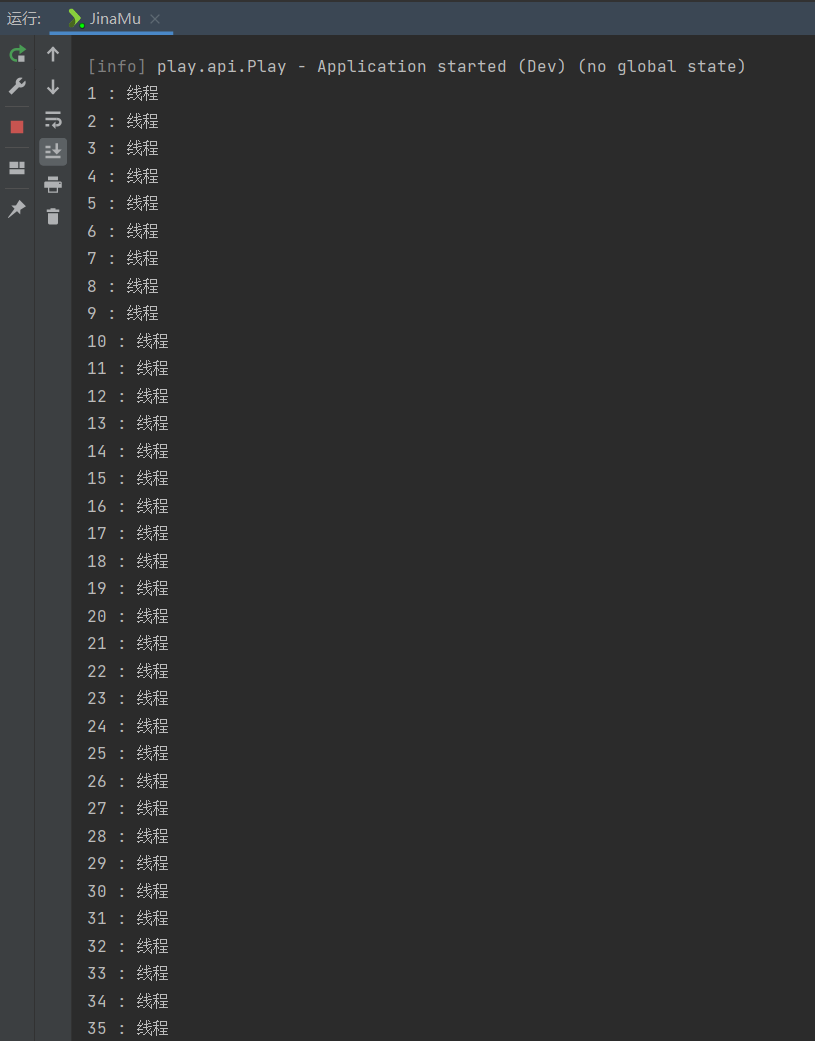
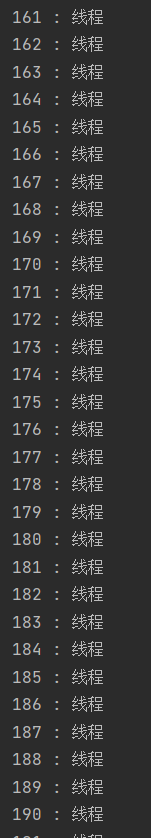
1. **实现服务端支持客户端的并发请求**
2. 在fileUploadApi方法中实现服务端接受多线程请求的api，并与uploadDataSet同样地，为保证线程安全，添加上synchronized关键字。



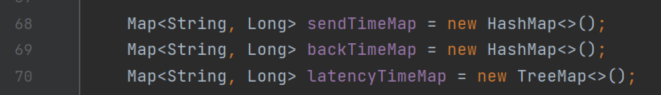
1. 服务端处理每个请求的响应逻辑。



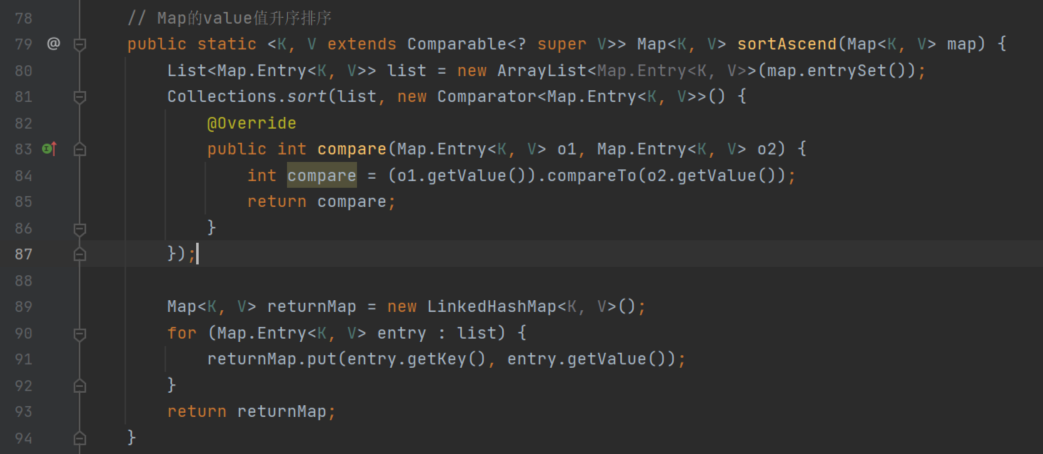
1. 测试结果：服务端成功处理客户端的并发请求。输出信息中每一行首部的数字仅代表服务端接收到请求的顺序，并不等同于上述步骤中客户端里的线程序号index。

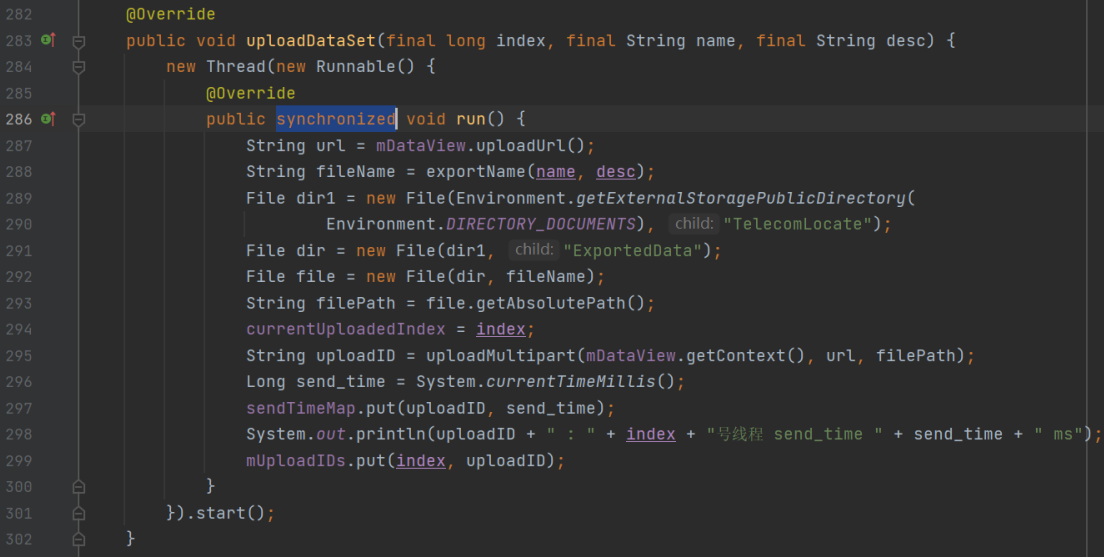
1. **客户端统计发送的请求的返回延迟**
2. 在DataPresenter类中声明三个Map容器,key均是线程的uploadId,value分别存储线程的sent\_time,back\_time,latencyTime。其中延迟时间latencyTime = back\_time - sent\_time。



1. 定义一个sortAscend方法实现对Map容器按value升序排序。



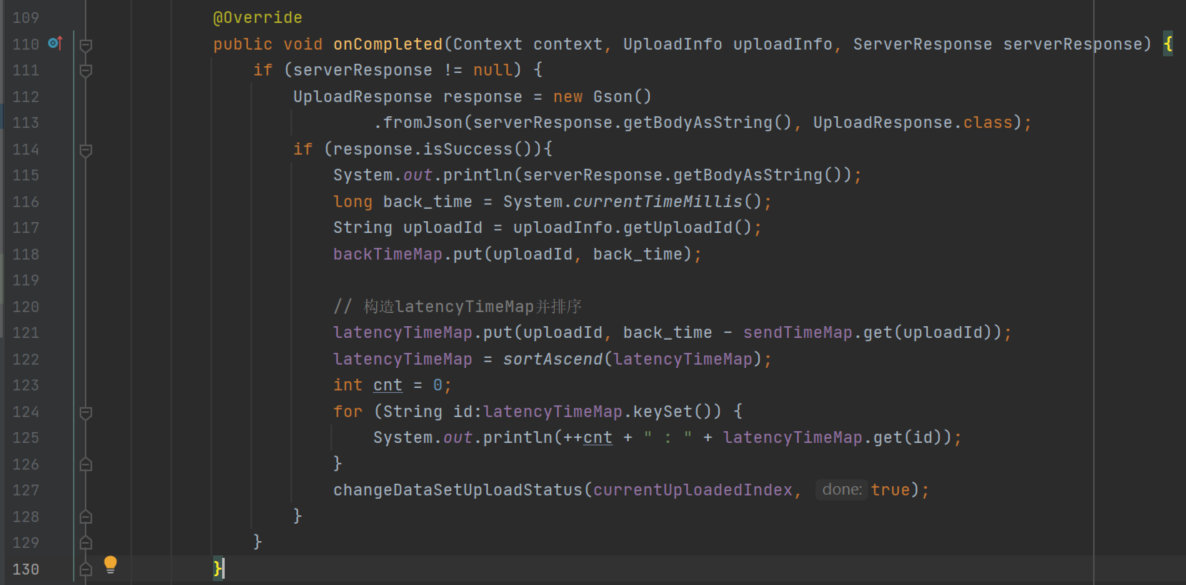
1. 在uploadDataSet方法中记录每个线程的send\_time,并与uploadId组成键值对存入sendTimeMap。



1. 在onCompleted方法中记录每个线程返回的back\_time,与uploadId组成键值对存入backTimeMap。

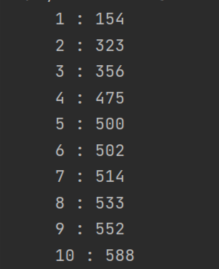
同时back\_time减去send\_time得到的latencyTime与uploadId组成键值对存入latencyTimeMap。

对latencyTimeMap采用sortAscend方法实现按照延迟从低到高排序,并在排序后输出延迟时间。

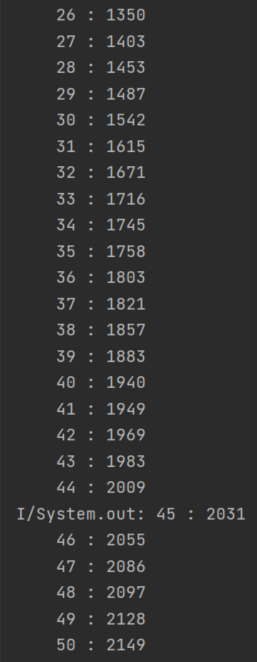
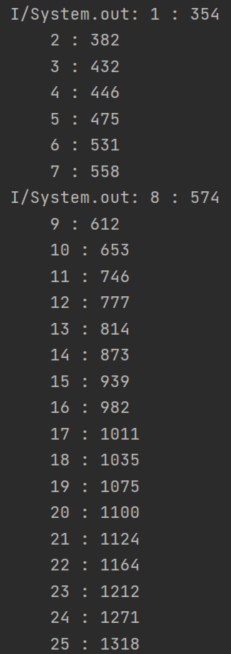


1. 统计结果:

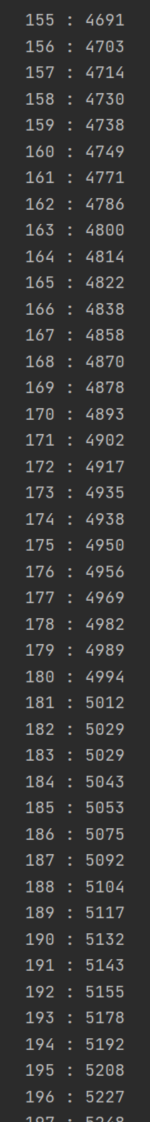
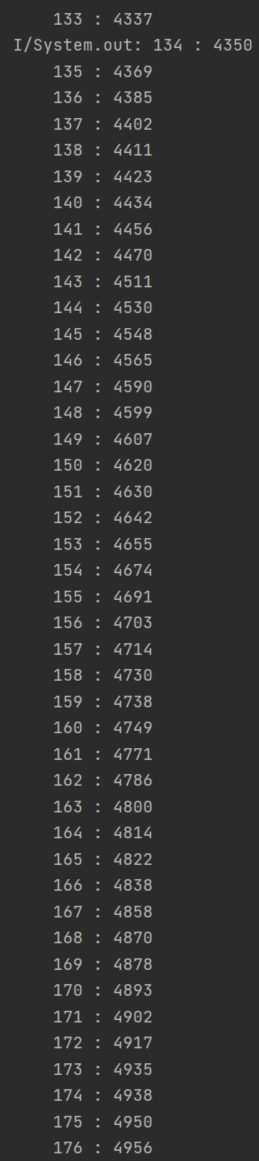
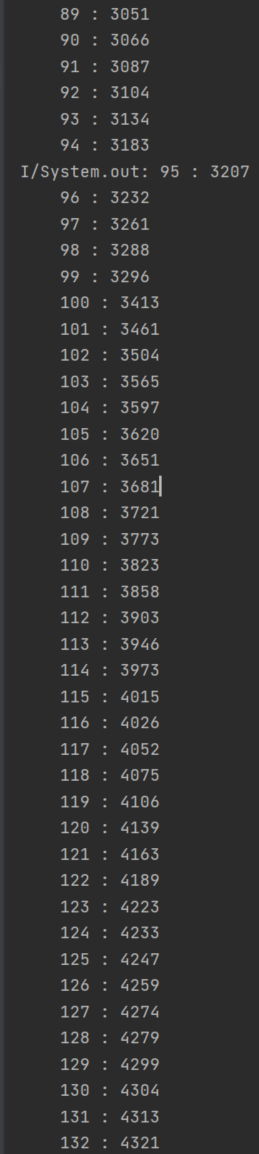
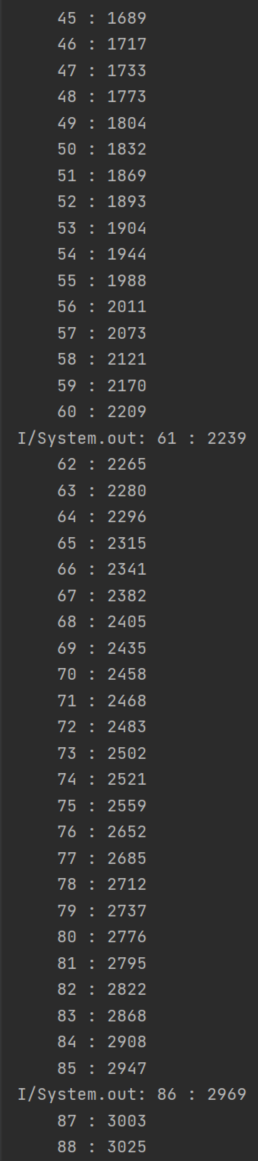
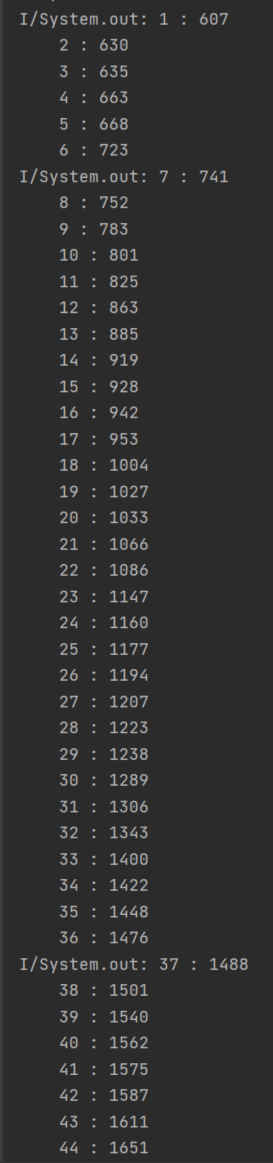
* 并发数 = 10



* 并发数 = 50



* 并发数 = 200



|  |  |  |  |
| --- | --- | --- | --- |
| 并发数\latency | Top 50% | Top 90% | Top 99% |
| 10 | 500 ms | 552 ms | 588 ms |
| 50 | 1318 ms | 2031 ms | 2149 ms |
| 200 | 3413 ms | 4994 ms | 5258 ms |