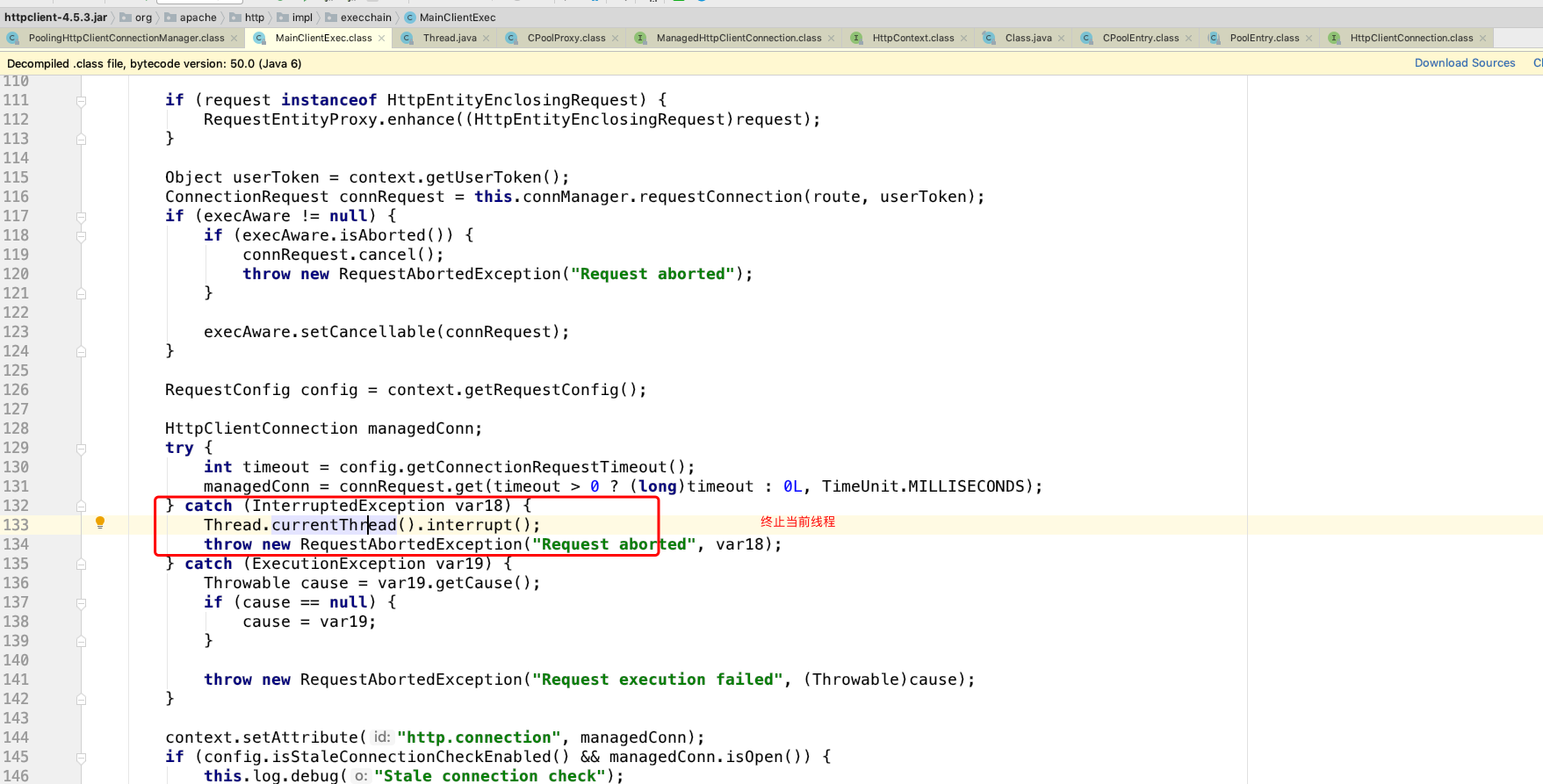
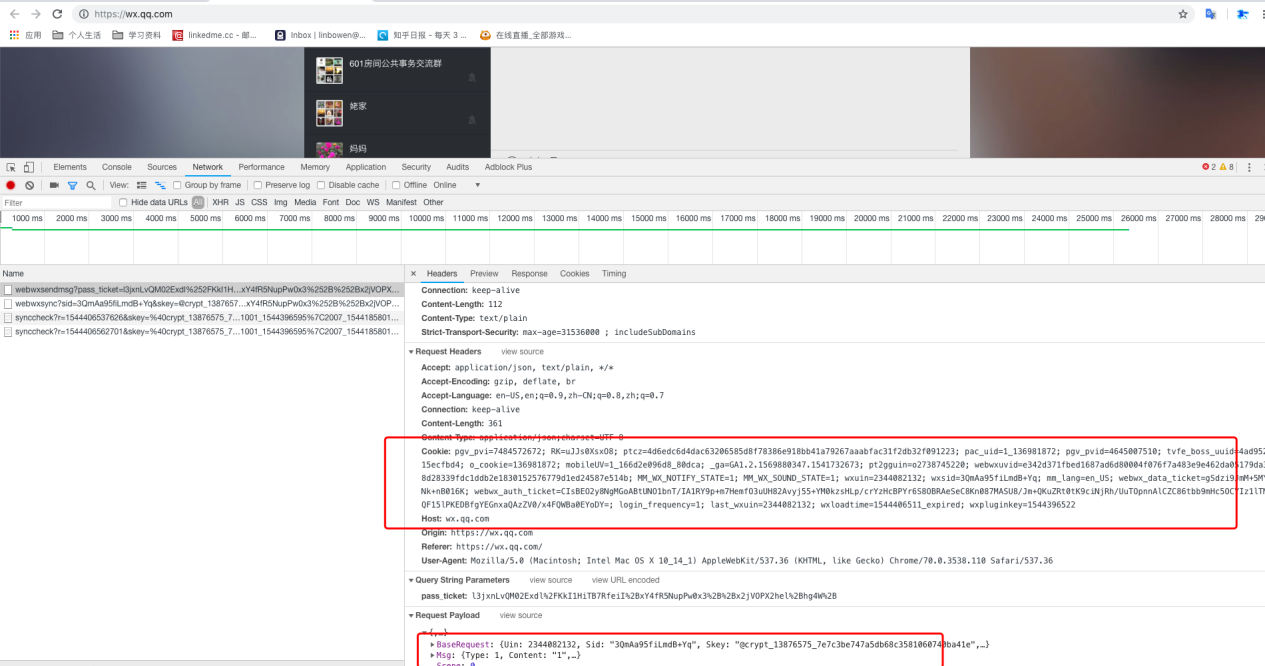
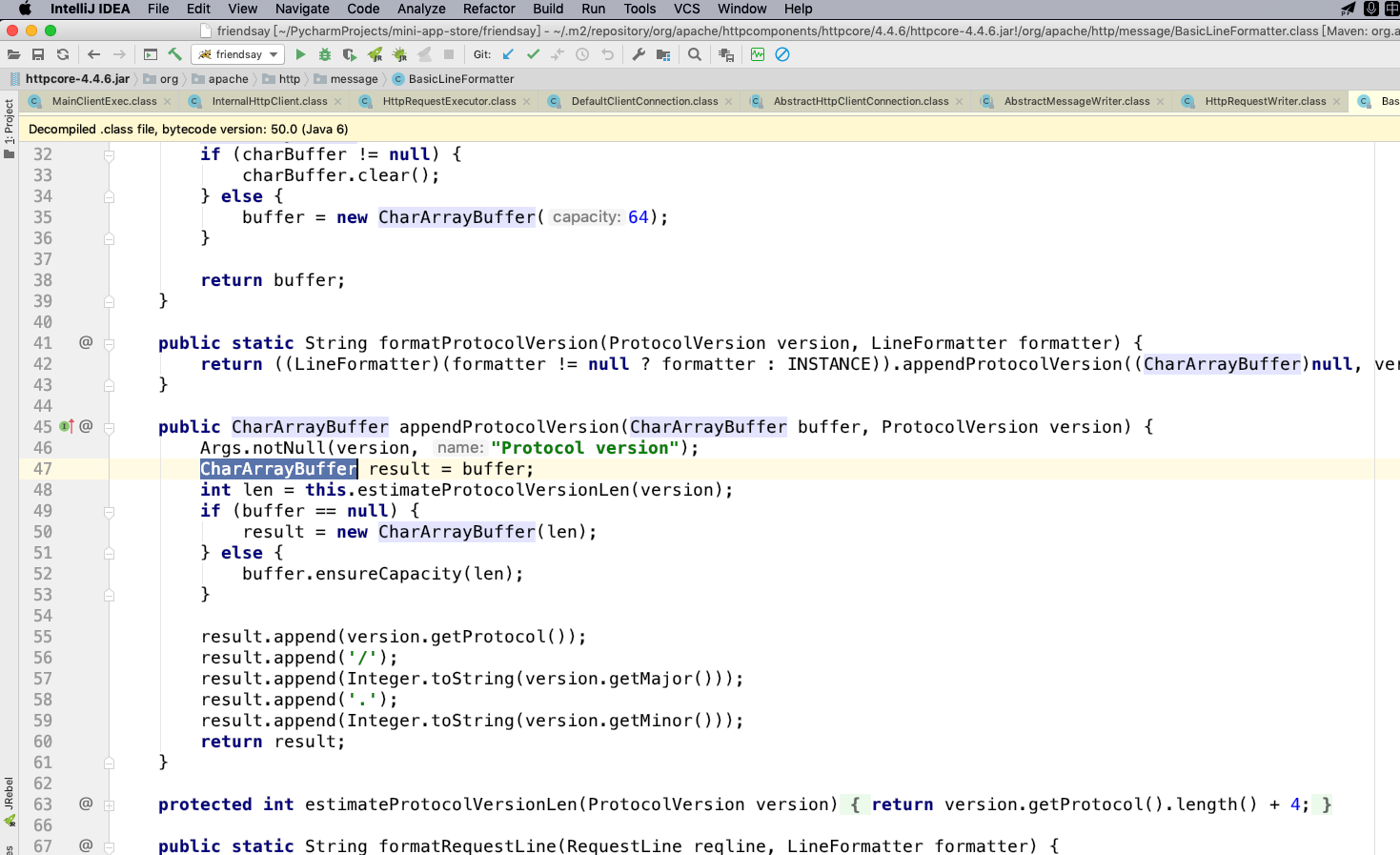
## 多线程的使用总结(线程池,锁,httpclient 线程池)

1. 线程池中 newFixedThreadPool 和 newSinglePool 最终都会调用 new ThreadPoolExecutor 进行创建线程池,当线程池进行高并发访问的时候,不执行的线程会进入 LinkedBlockingQueue 队列中进行等待

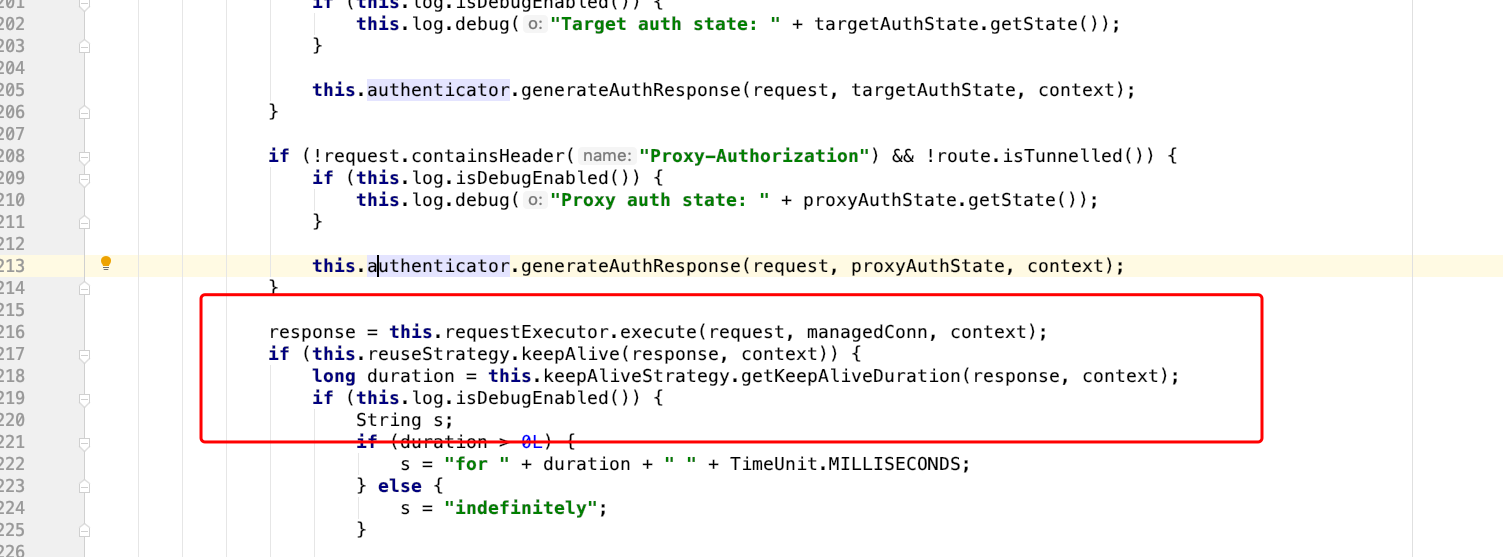


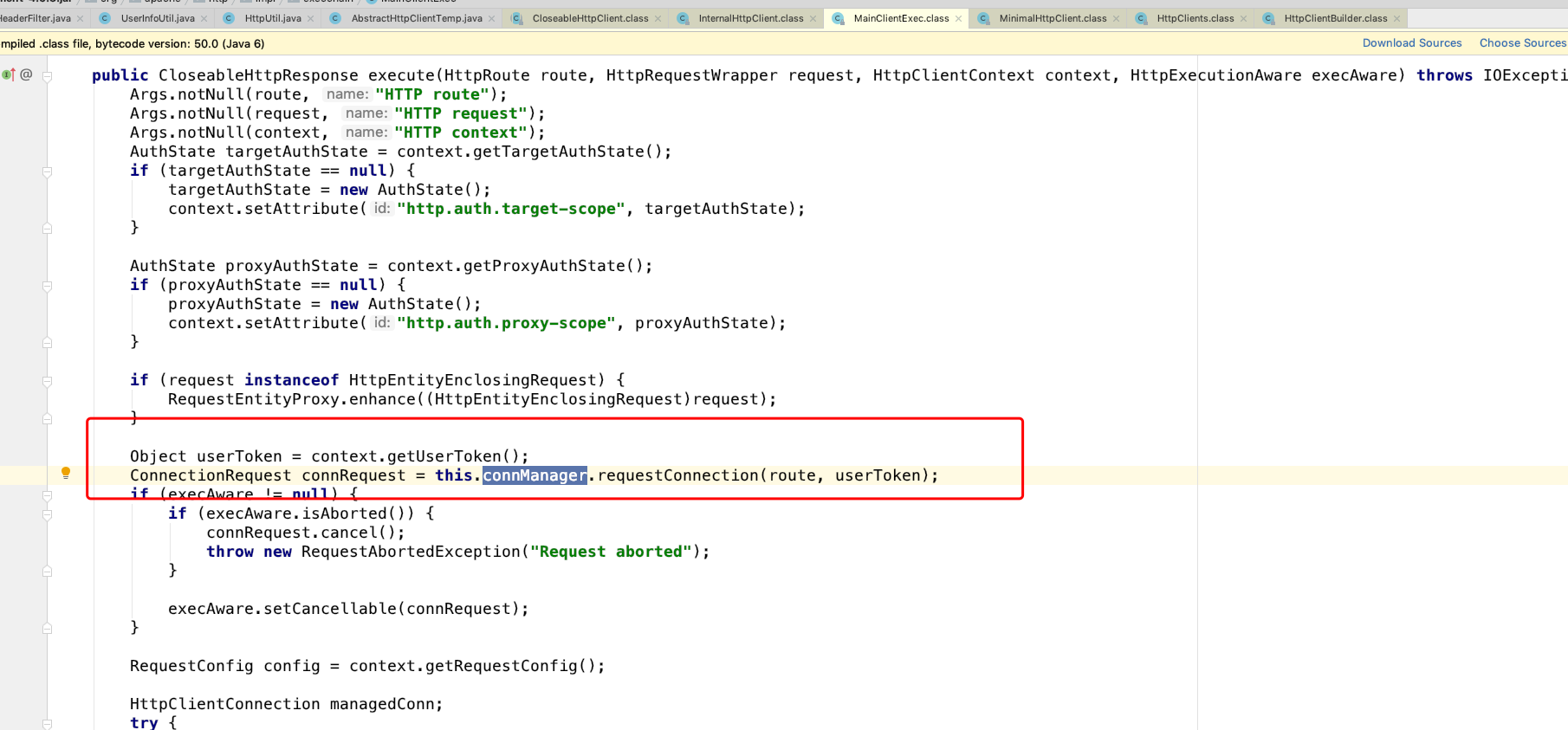
1. newFixedThreadPool 会自动回收不在使用的线程
2. httpclient 线程池的使用,通过创建PoolingHttpClientConnectionManager对象进行设置线程池,
   1. cm.setMaxTotal(200); 用来设置请求的最大并发量
   2. cm.setDefaultMaxPerRoute(100); 用来设置每一个 ip 的最大并发量
3. 在使用 httpclient 的线程池进行操作时,不能调用 httpclent.close() 方法,因为这个方法会关闭 请求线程池.这样在下次使用线程池进行访问的时候,就会出现 connection shutdown 的异常
   1. 调用 httpget.releaseConnection () 不会关闭连接,而是会重置链接的状态
   2. httpclient如果没有设置 cookiesstore 那么httpclient 会自动创建一个内置的 cookiestore 进行 cookie 存储
   3. 
   4. cookie 的访问方式,存在 https 和 http 的形式, https 的 cookie 只能在发送 https 请求的时候被发送,而进行 http 请求的时候不会发送
4. ReentrantLock 的优点是可以使用 trylock 进行尝试,此外提供了内置的 newcondition 的方法进行线程的 await 和 signal 控制
   1. 在线程 await 休眠的时候会释放,获取到的锁
   2. 在进行 signal 的时候需要获取到ReentrantLock 对象的锁,否则会出现 ilegalMonitorException 异常
   3. ReentrantLock默认是非公平锁,需要在构造方法上指定为 true
   4. ReentrantLock 的内部实现是 Sync

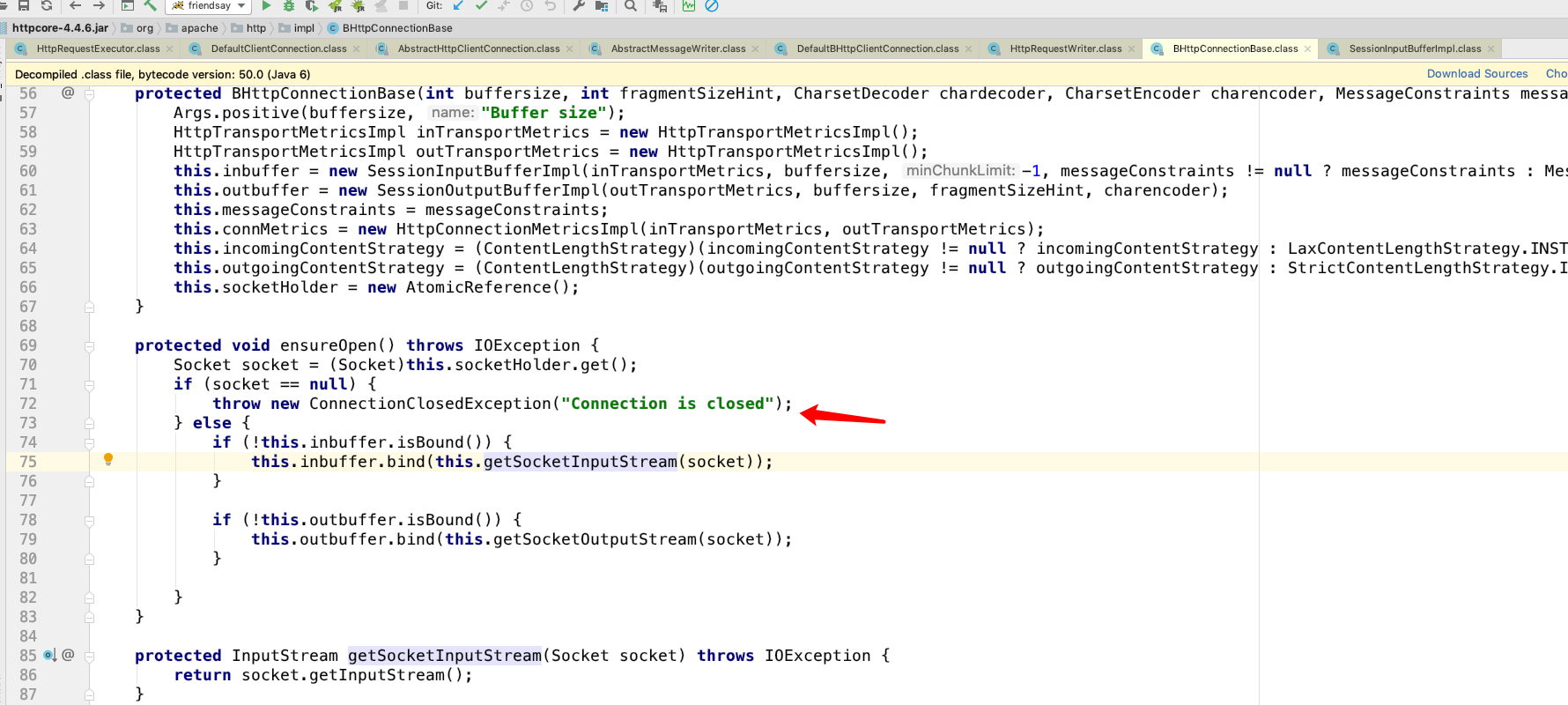
## httpclient 源码分析总结



1. httpclient 发送 http 请求的本质是建立 socket 连接,通过传输 http 协议的报文实现通讯
2. Httpclient 请求对应的是一个 future 对象,通过线程池执行这个 future 对象







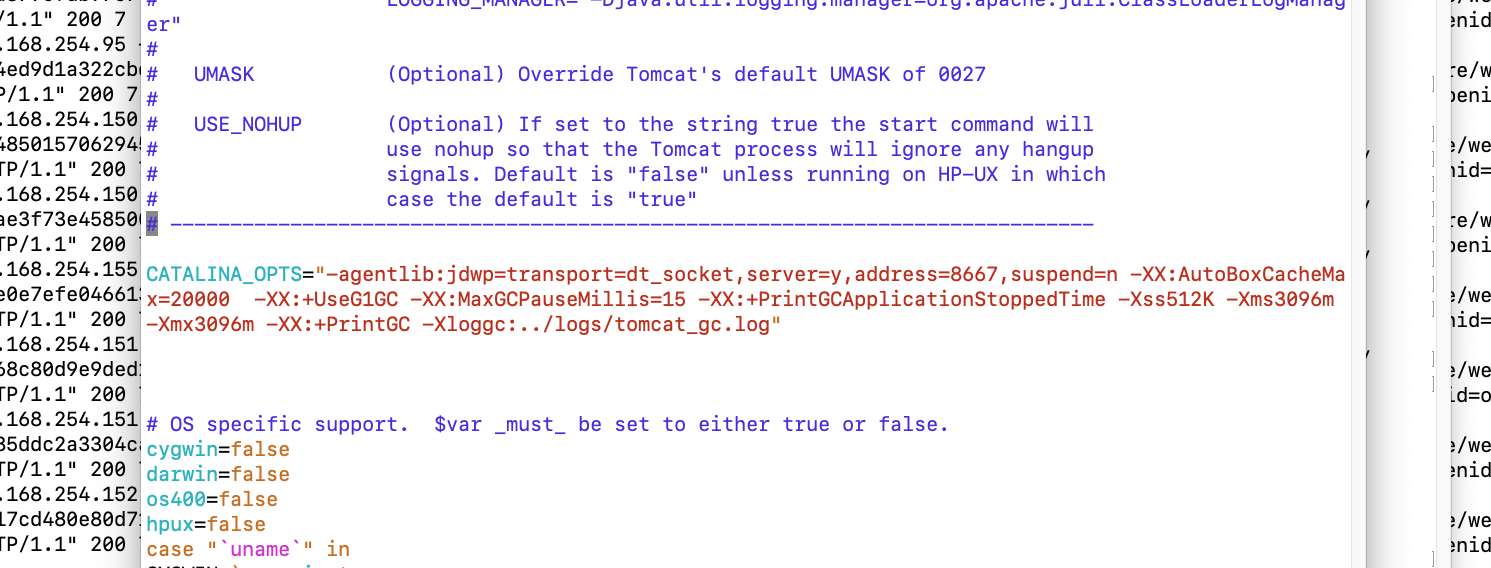
1. 在 execute 的时候会向 poolConnectionManager 访问进行对象创建.
2. CharArrayBuffer StringBufferBuilder

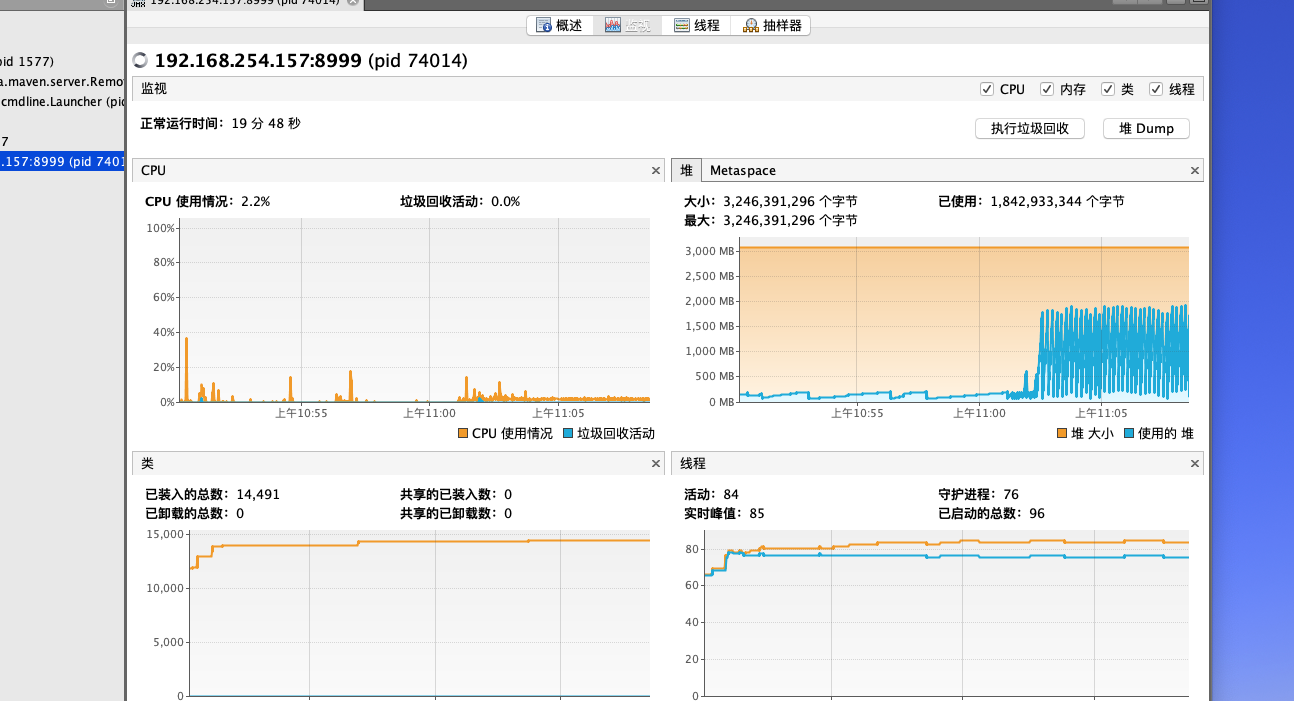
## java stream总结

1. forkPool 进行并发执行
2. Stream 和 pattralStream 用来区分串行流和并行流
3. 使用 lamada 表达式进行流操作, lamda 表达式中不支持参数类型, lamada 表达式会自动识别参数类型
   1. 例如 (user)->{ System.out.println(user.getName();)}

4.在使用 runnable ,等支持匿名类的地方都可以使用 lamada 表达式进行代替

## jvm 异常处理(堆,栈,线程管理)的总结





1. 使用线程池的时候,如果在线程中有死循环使用 jmap 观察堆的 gc 曲线即可发现,当线程中有死循环的时候,堆中 gc 频繁发生,而且曲线是锯齿状.

解决方案

1. 使用 jvisalvm 观察 jvm 虚拟机内存
2. 使用ps -ef | grep tomcat 找出 tomcat 对应的 pid
3. 使用 top -H -p pid 找出这个 jvm 中频繁活动的线程
4. 使用 jstack <pid> | grep <十六进制子进程号> 观察对应的线程栈
5. tomcat 异常退出的情况分析
   1. linux 内存满了
   2. 代码中存在 System.exit



