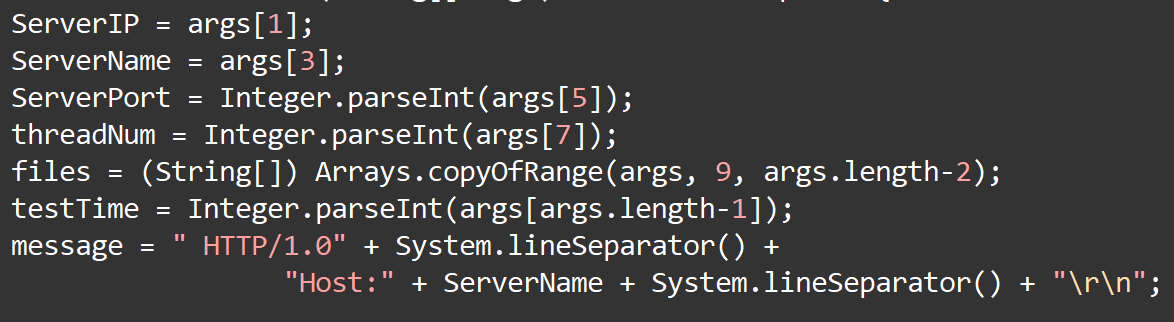
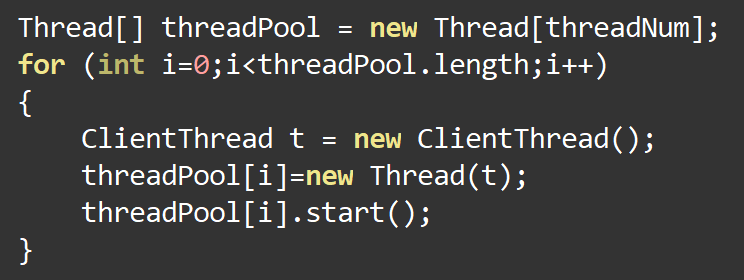
HTTP协议与Web服务器

### 1a：简单的客户端

得到参数设置的服务器IP、名称、端口，以及客户端的线程数、请求文件、测试时间

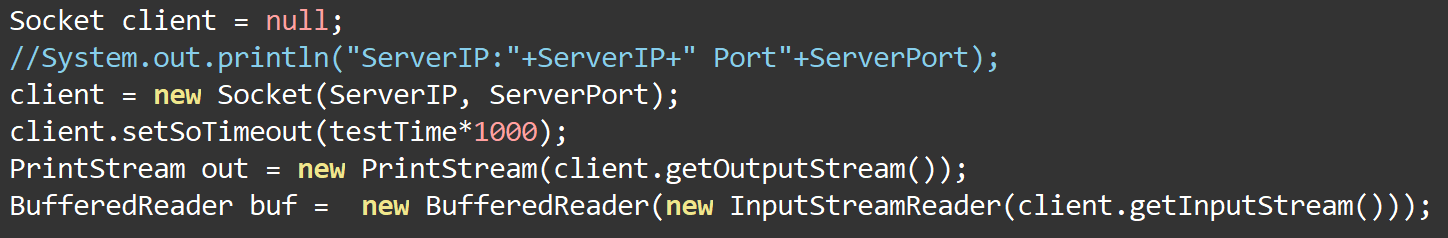


开启多线程

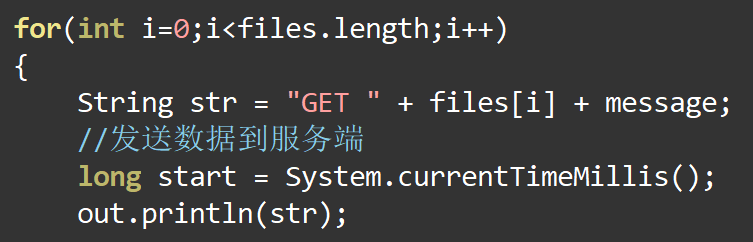


对于每个线程：

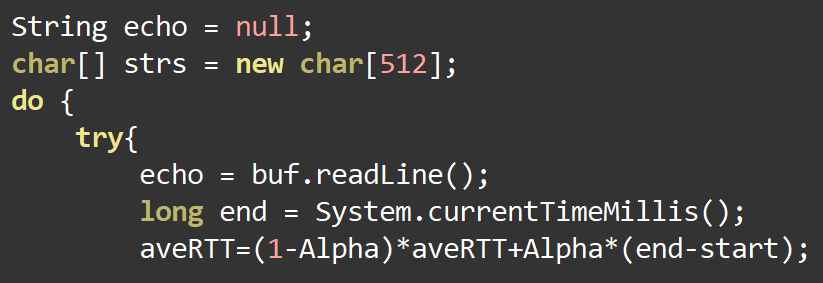
建立连接



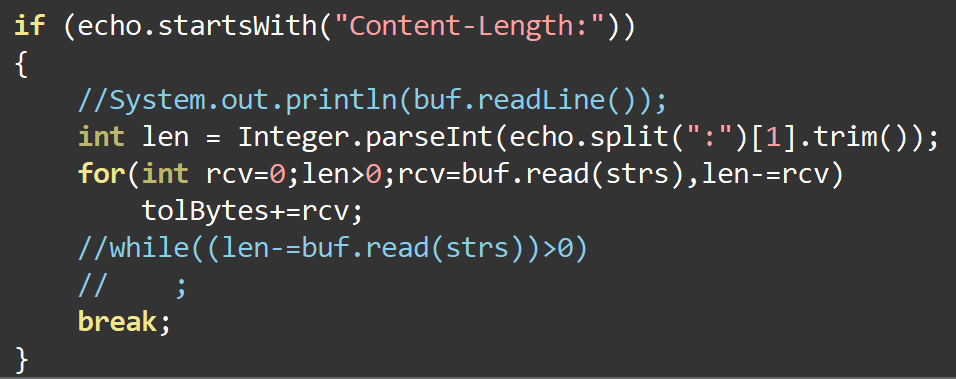
顺序发送各文件请求，记录下时间戳以测试性能



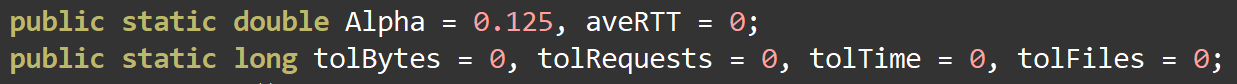
循环接收消息，并统计RTT时间，根据EWMA方法统计



读取文件数据，统计字节数

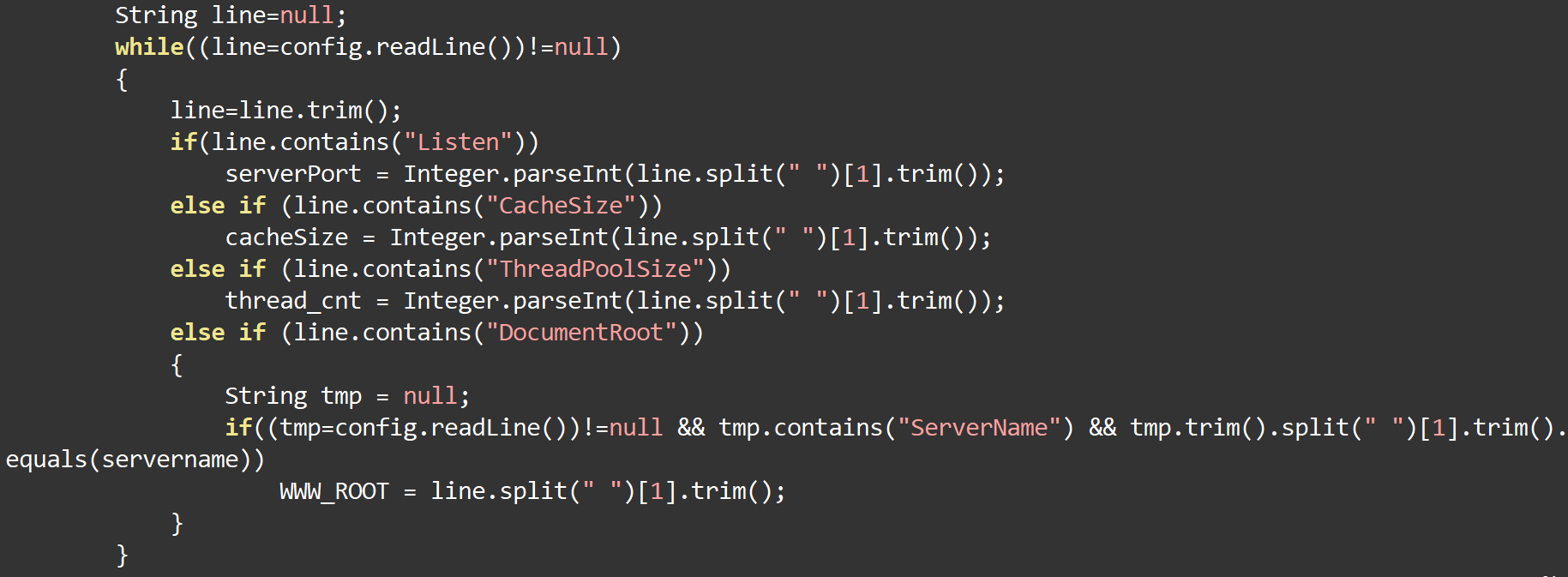


通过public static类型收集多个线程的统计信息

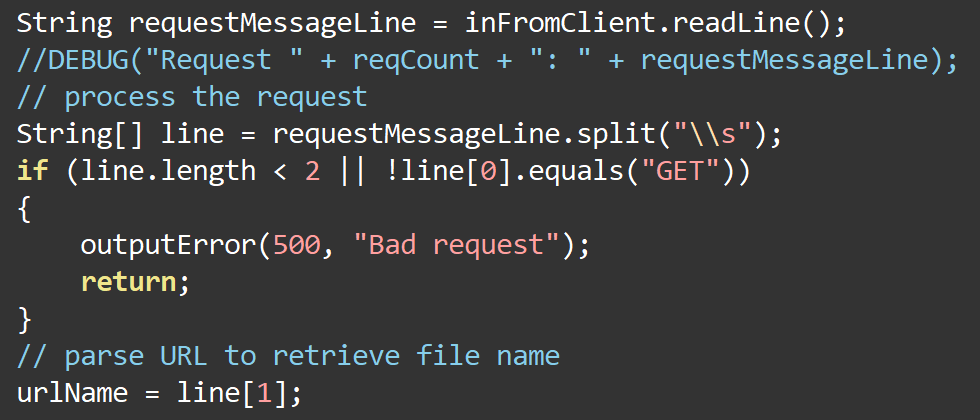


### 1b：顺序和每线程HTTP服务器

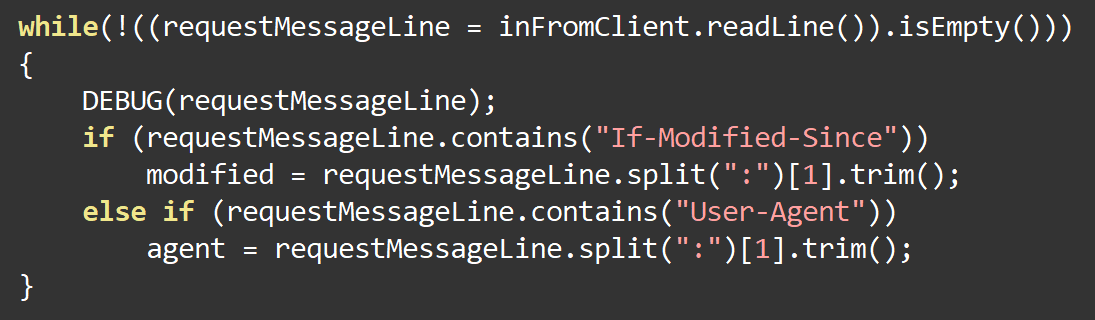
从配置文件读取配置，通过java字符串操作，包括服务器端口、缓存大小、线程池大小、服务器根目录



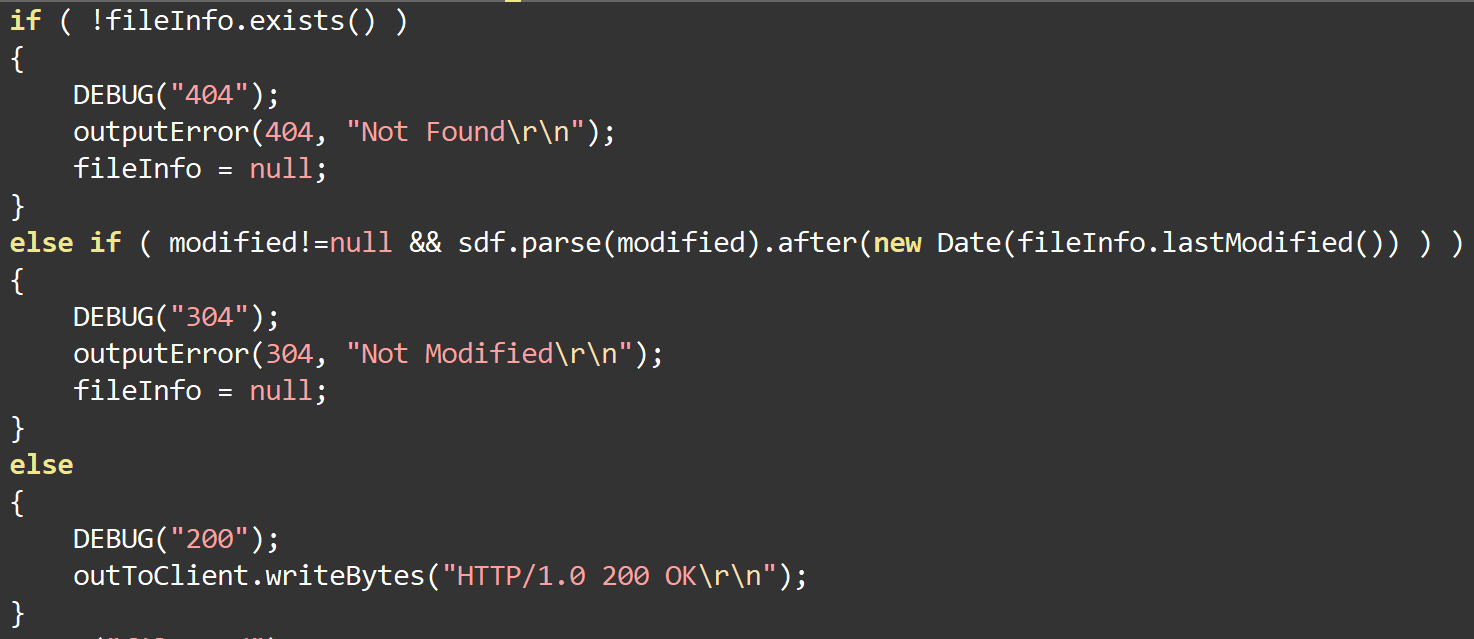
获取客户端发送的数据流，判断HTTP方法，并得到url

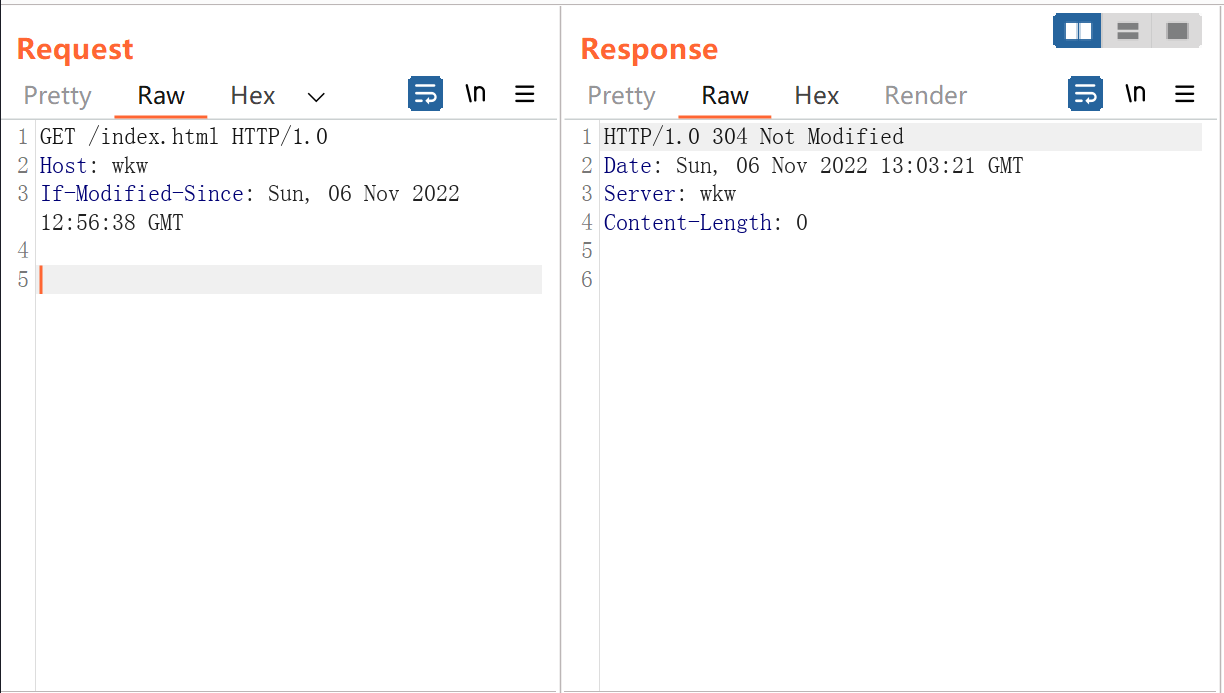


分析If-Modified-Since和User-Agent标头，用于后续响应处理



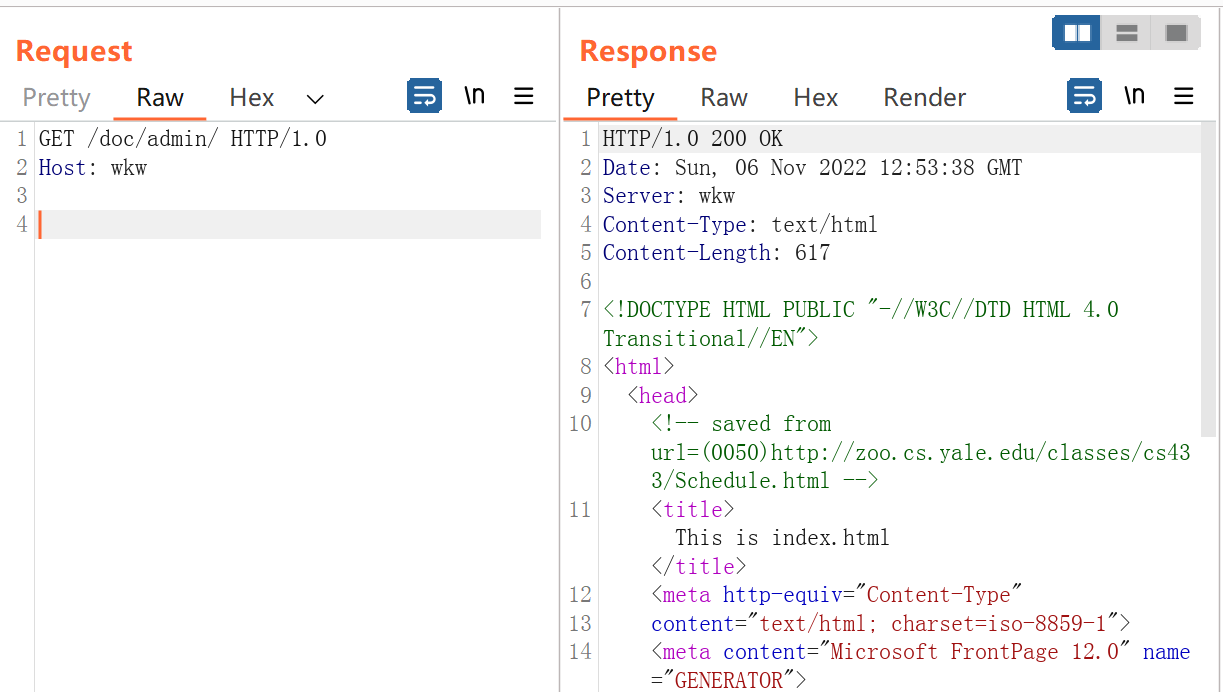
响应请求头

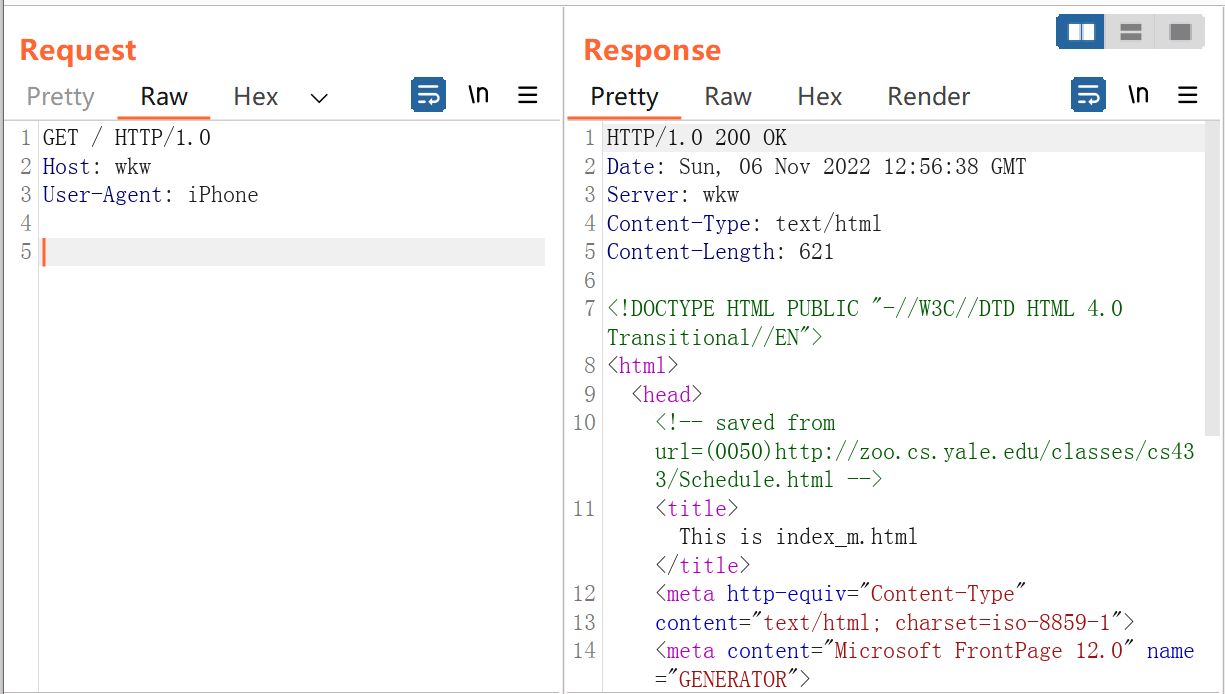




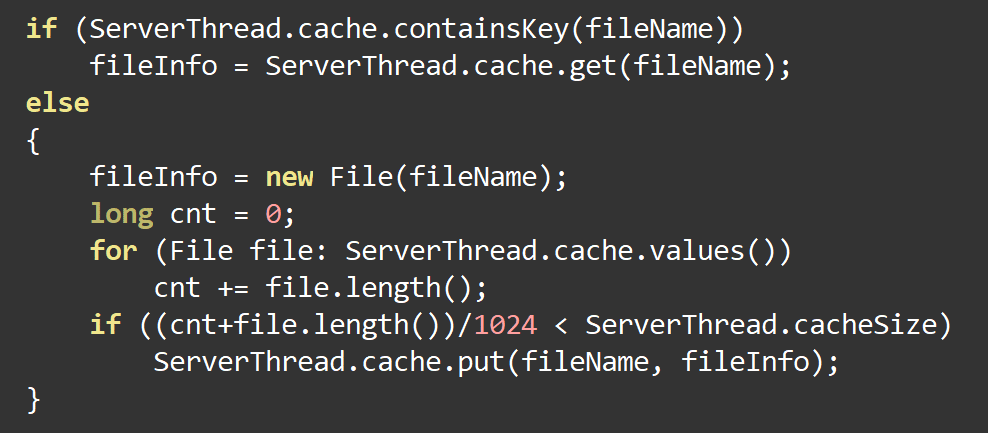
URL映射，根据要求，设置默认URL





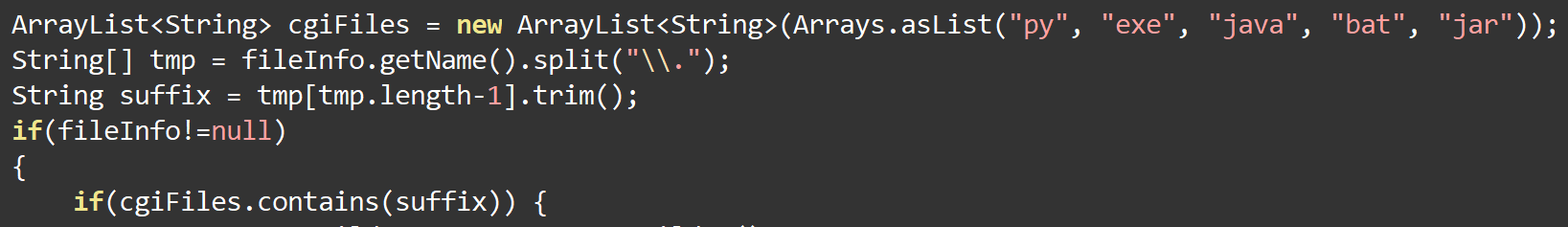


缓存，Map存储，读取文件时先在缓存中查找，找到即可直接读取，否则判断缓存空间是否足够

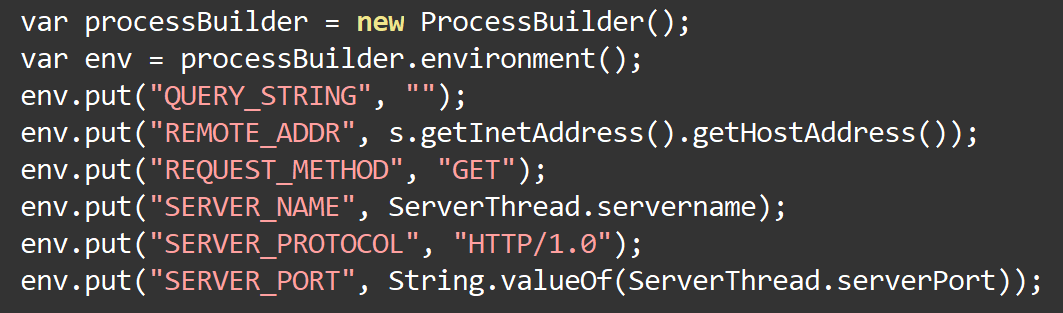


获取CGI文件动态内容

首先判断文件名后缀是否属于可执行文件

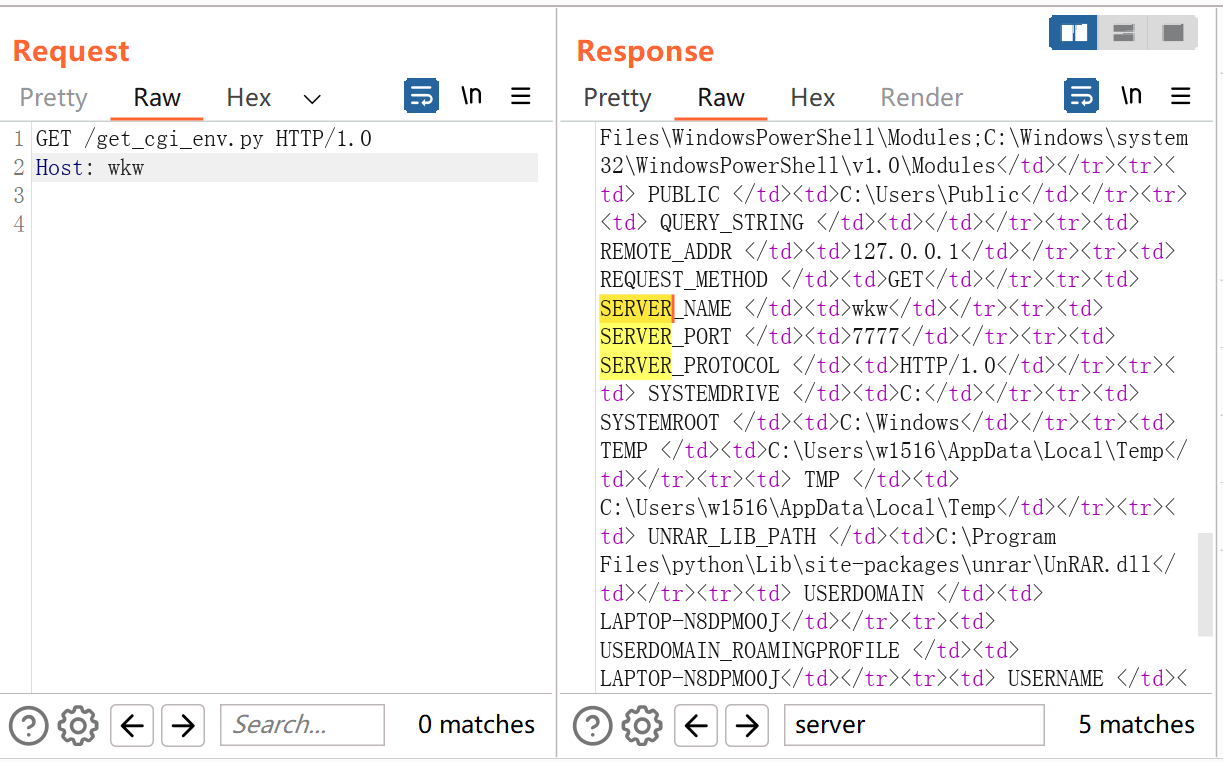


然后创建ProcessBuilder类，设置环境变量



根据文件后缀，执行相应命令以启动CGI程序，最后得到程序输出并发送给客户端

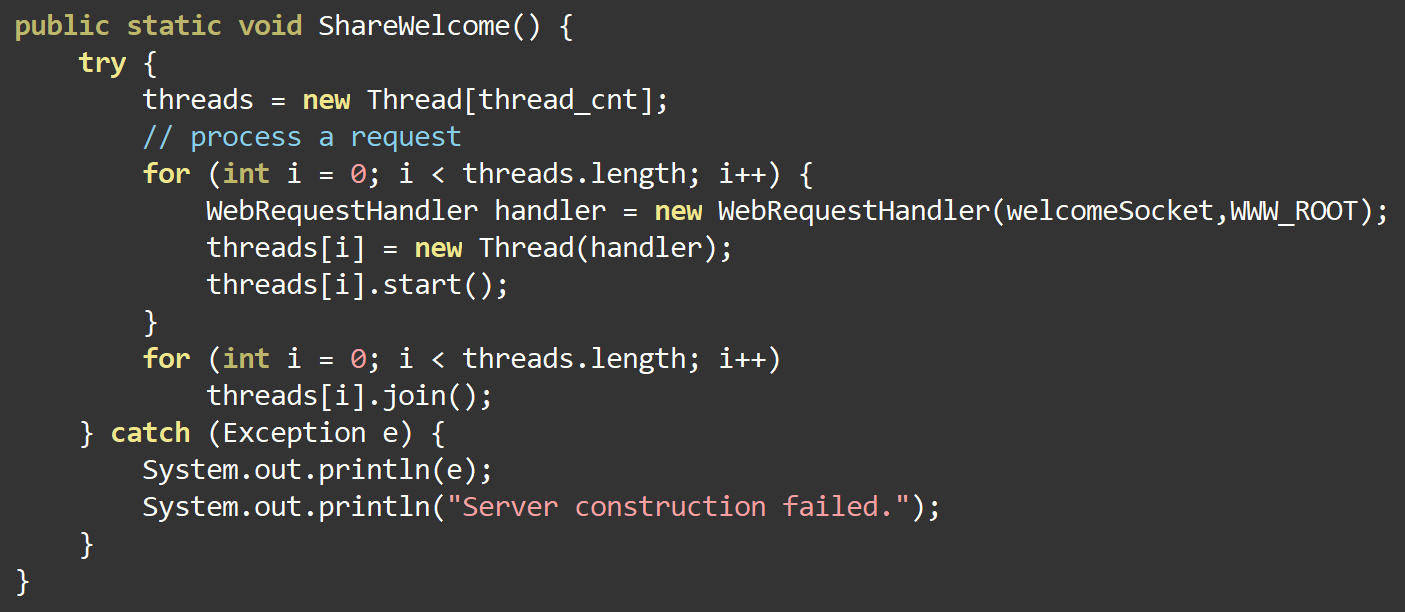




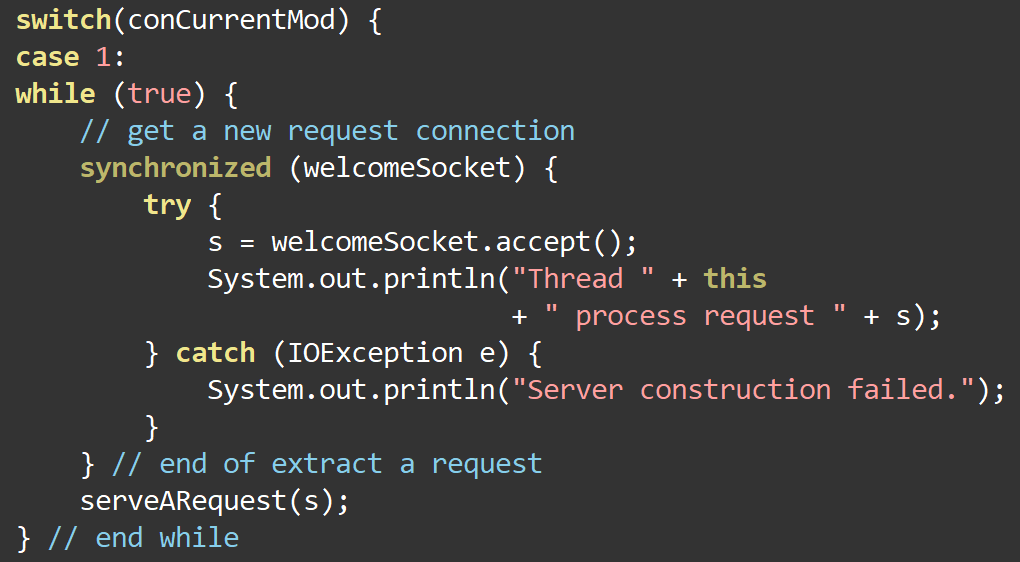
### 2a：使用线程的并发HTTP服务器

1. 服务线程在Welcome套接字上竞争的线程池

根据配置文件中的线程数，创建线程数组，开启多线程

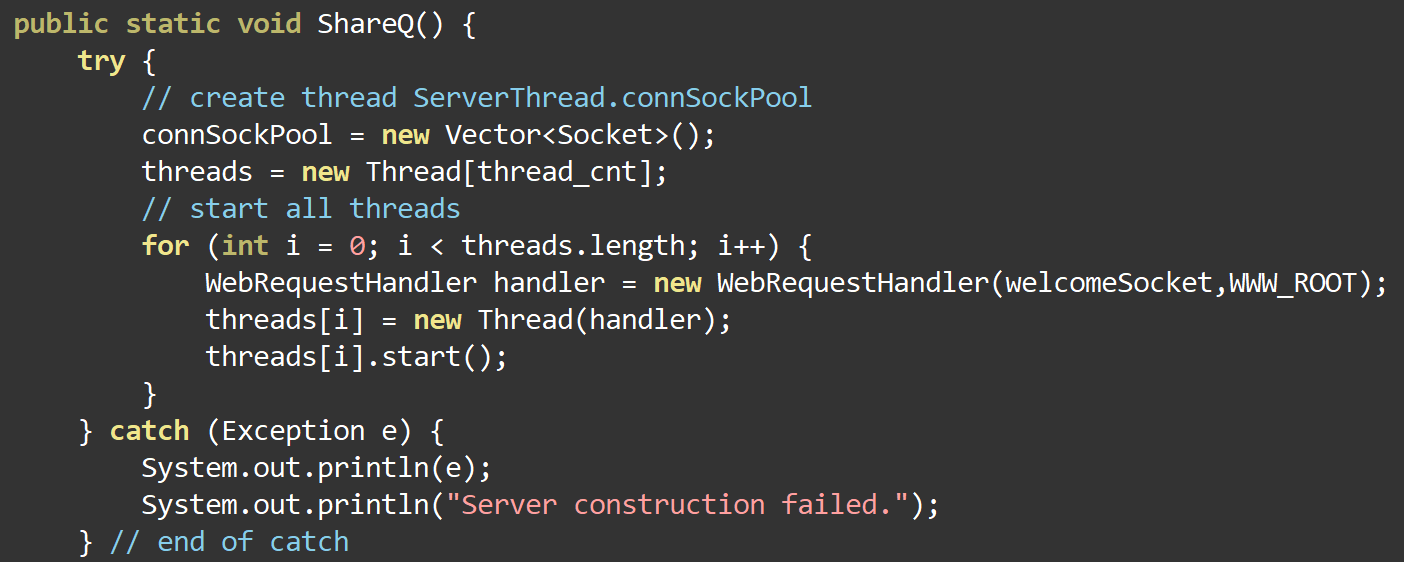


对于每一个线程，在Welcome套接字竞争获取客户端请求

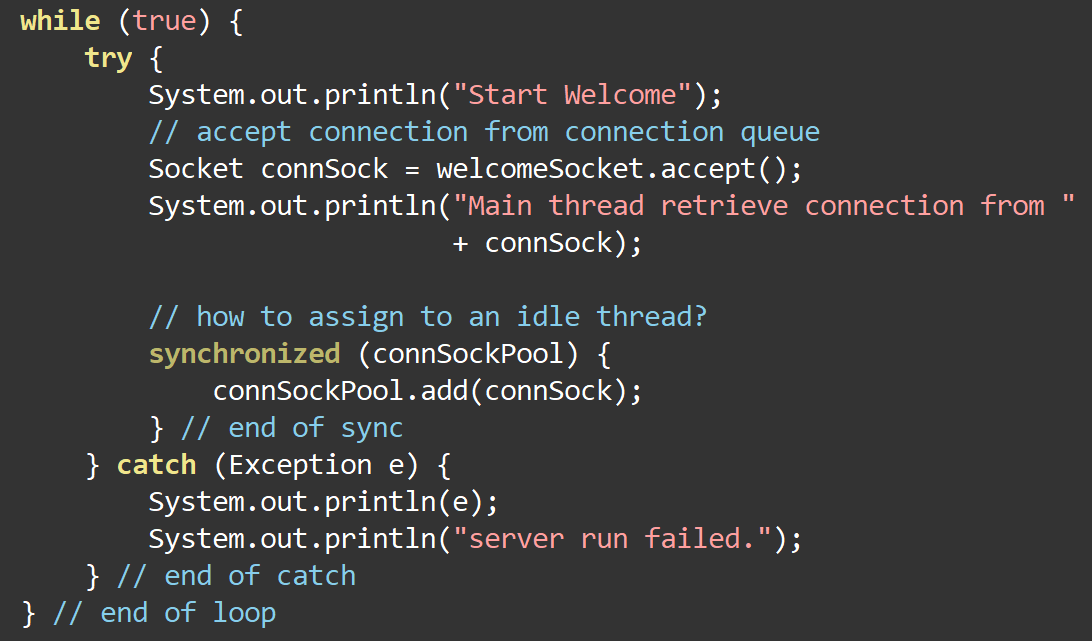


1. 具有共享队列和忙等待的线程池

同上，根据配置文件中的线程数，创建线程数组，开启多线程



将Welcome套接字上的客户端请求存放在队列connSockPool中



对于每一个线程，从connSockPool队列中获取请求，共享一个请求队列



### 2c：设计比较

* Netty provides multiple event loop implementations. In a typical server channel setting, [two event loop groups are created](http://netty.io/5.0/api/io/netty/bootstrap/ServerBootstrap.html#group(io.netty.channel.EventLoopGroup,%20io.netty.channel.EventLoopGroup)), with one typically called the boss group and the second worker group. What are they? How does Netty achieve synchronization among them?

boss用来接收进来的连接，worker用来处理已经被接收的连接，一旦‘boss’接收到连接，就会把连接信息注册到‘worker’上

* Method calls such as bind return ChannelFuture. Please describe how one may implement the sync method of a future

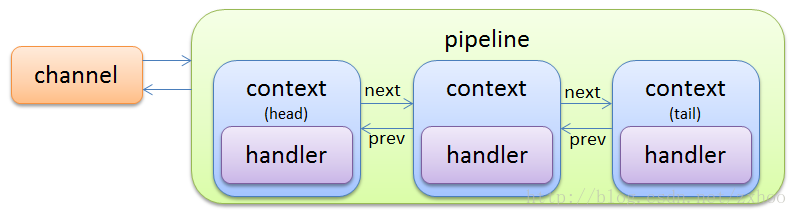
让需要未来结果的Future先睡眠，其他线程异步执行，当未来的结果产生后，再回调唤醒Future

* Instead of using ByteBuffer, Netty introduces a data structure called ByteBuf. Please give one key difference between ByteBuffer and ByteBuf.

ByteBuffer只有一个位置指针，需要有position、limit、flip等属性和操作来控制数据读写，Bytebuf通过两个位置指针来协助缓冲区的读写操作，更加方便，功能更强大

* A major novel, interesting feature of Netty which we did not cover in class is ChannelPipeline. A pipeline may consist of a list of ChannelHander. Compare [HTTP Hello World Server](http://netty.io/4.0/xref/io/netty/example/http/helloworld/package-summary.html) and [HTTP Snoop Server](http://netty.io/4.0/xref/io/netty/example/http/snoop/package-summary.html), what are the handlers that each includes?
* Please scan Netty implementation and give a high-level description of how ChannelPipeline is implemented

ChannelPipeline本质上是一个双向链表，它采用了责任链模式。在Netty中每个Channel都有且仅有一个ChannelPipeline与之对应。一个Channel包含了一个ChannelPipeline，而ChannelPipeline中又维护了一个由ChannelHandlerContext组成的双向链表。这个链表的头叫HeadContext，链表的尾叫TailContext，并且每个ChannelHandlerContext中又关联着一个ChannelHandler



### 2d：性能基准测试

服务器分别测试一小一大两种线程池，对于每种线程池，客户端测试从小到大（1、2、3、4、5、10、15、20、25、30、35、40、45、50、60、70…）线程数的请求，每个请求的请求文件序列存放在requests.txt文件，通过帕累托分布生成

