一、answer all questions specified above

Part 1a: Simple Client

Think about how to collect statistics from multiple threads.

> 在管理多个 thread 的类（即主类）中建立 HashMap 用于存储，在进程开始时将 HashMap 作为形式参数传入在线程内修改来收集多线程的数据。

Part 1b: Sequential and Per-thread HTTP Servers

how to handle multiple threads reading and adding to the Map.

> 在管理多个 thread 的进程中建立 HashMap 用于存储，层层传递，在创建 request 对象时初始化类中的 HashMap 进行管理。

```
WebRequestHandler wrh =
                new WebRequestHandler( connectionSocket, cfgMap, fileCache );

public WebRequestHandler(Socket connectionSocket,
                 Map<String, String> cfgMap, Map<String, String> fileCache)
throws Exception
    {
        reqCount ++;
        this.fileCache = fileCache;
        ...
```

- Heartbeat monitoring: Please describe a particular design and implement it.

  > 此处设计了一个 monitor，监视器的效果为每隔一段时间进行一次请求数目的测量，测量时间为 1s。当测量得到的请求数大于某个阈值时，关闭这一线程从而拒绝新的连接。线程的管理采用 ThreadPool 实现。

  > 目前仅在 part 1b 实现此功能，具体代码参见 ./SequentialAndPer-threadHTTPServers/HeartbeatMonitor.java

Part 2b (Option 1): Async Server: Multiplexed, Nonblocking Server (Reactive Server)

- **Design question:** a production-level server should have a timeout thread. Upon accepting a new connection, the accept handler should register a timeout event with the timeout thread with a callback function. The timeout value is specified by IncompleteTimeout . The default timeout value is 3 seconds. If the connection does not give a complete request to the server approximately within timeout from the time of being accepted, the server should disconnect the connection. Note that the timeout monitoring thread should not directly close a channel that the dispatcher thread is still monitoring (why?). You need to think very carefully about the exact details of the

interaction between these two threads, and describe your software design in your final report. You DO not need to implement it, but if you do, please let us know in your report, and you will receive a 5% bonus points if your implementation pass the functionality test.

设计：

在 Acceptor 中的 Server 接收套接字连接的同时使用 currentTimeMillis() 方法设计一个定时器，并将其放进一个线程（thread）当中，使他在 accept() 阻塞时保持运行。当时间超过 3s 时，定时器返回一个超时信号，使handleAccept(SelectionKey key) 这一方法返回一个失败的信号给 Dispatcher，Dispatcher 如果收到这一信号，则不进行 R/W 操作，反而将 channel 关闭；或者重新建立一个接受新连接的 key。

Acceptor：

```
// 放入一个 TimeOut Thread
// extract the ready connection
SocketChannel client = server.accept();
Debug.DEBUG("handleAccept: Accepted connection from " + client);
```

TimeOut Thread：

```
// currentTimeMillis()
// if
// return
```

Note that the timeout monitoring thread should not directly close a channel that the dispatcher thread is still monitoring (why?).

如果直接关闭了 channel，则在 Dispacher 中判断是否可读或者可写时会出现找不到 key 的问题，导致程序运行出错。