多客户端查询

题目要求

1. 实现客户端发送数据，服务端查询并返回数据；
2. 在上述要求下，实现客户端服务端多线程查找，重点在于负载均衡以及服务端多线程并行加速查找。

解题思路

服务端

1. 用线程池（Executors.newFixedThreadPool())作为服务端线程池，每提交一个任务就创建一个线程，直到达到线程池的最大数量，这时线程池的规模将不再变化，多余的任务将进入队列等待，若某个线程结束，那么线程池将从队列中补充一个新的线程。
2. 创建一个ServerSocket类，在指定端口建立一个监听服务。为了随时捕捉到一个来自Client端的请求，在无限循环中调用ServerSocket.accept()方法，一旦有client端连入，便用线程池启动一个任务，执行查询。

客户端

1. 指定线程数量，循环创建多个客户端线程，每个客户端线程建立新的Socket，端口号与serverSocket相同，如此便可与服务端线程建立连接。在主线程建立Vector存储每个子线程，方便后续数据调用；
2. 主线程获取test.txt中的单词，构建**二维数组[线程数][列数]，其中列数为 单词总数/线程数 上取整**，单词数可由LineNumberReader.getLineNumber()获得；
3. 主线程在获取所需查询的单词后，将单词均匀分配到每个线程，具体实现如循环队列一般，每为一个线程分配一个单词就将下一个单词分配至下一个线程；
4. 每个客户端线程在收到任务分配后，将所需查询的单词补"\0"后转化成字节数组转发至服务端线程（**重要的事情说三遍！慎用BufferReader！慎用BufferReader！！慎用BufferReader！！！传String一时爽，readLine()阻塞要人命！**），服务端线程处理完数据之后返回给客户端线程；
5. 待所有客户端线程执行完毕后（**thread.join()**），按照步骤2的顺序从每个线程存储结果的数组中取出单词，转化成字符串后作为结果输出到out.log文件中。

关键代码实现

读写文件、查询相关代码与第一题大致相同

服务端

```
public class Server {
    private static final int SERVER_PORT = 8899;
    private static final int ARRAY_LENGTH = 1024;
    private ServerSocket serverSocket;
    private ExecutorService serverPoolExecutorService;
    private byte[] all = new byte[ARRAY_LENGTH];
    private String[] words;

    private Server(int serverThreadCount) throws IOException {
        serverSocket = new ServerSocket(SERVER_PORT);
        serverPoolExecutorService = Executors.newFixedThreadPool(serverThreadCount);
        System.out.println("等待连接...");
    }
     /*
     *省略
     */
    private void service(String filePath) {
        try {
            while (true) {
                Socket socket = serverSocket.accept();
                serverPoolExecutorService.execute(new ServerThread(socket, filePath));
                System.out.println("已启动" + Thread.currentThread() + "服务端线程");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /*
     *省略
     */
}

public class ServerThread implements Runnable {
    private String filePath;
    private Socket server;
    private int id;
    private String[] words;// 存放单词，接受任务分配

    ServerThread(Socket server,String filePath) {
        this.server = server;
        this.filePath = filePath;
    }

    /*
     *省略
     */

    @Override
    public void run() {
        System.out.println("Server[port:" + server.getInetAddress() + ",id:" + id + "]已
运行");
        InputStream is = null;
        OutputStream os = null;
```

```
        try {
            byte[] raf = new byte[1024];
            is = server.getInputStream();
            is.read(raf);
            String str = new String(raf);
            this.words = str.split("\\s+|\n|\0");
            System.out.println("Processing Words:" + Arrays.toString(words));
            byte[] results;
            String result = "";
            for (int i = 0; i < words.length; i++) {
                if (!words[i].equals("null")) {//不能用!=比较字符串，!=比较的是引用
                    String feedback = search(words[i]);
                    System.out.println(feedback + "--" + Thread.currentThread());
                    result += feedback + "\0";
                }
            }
            results = result.getBytes();
            os = server.getOutputStream();
            os.write(results);
            System.out.println("Mission Completed");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
//              server.shutdownInput();
                if (os != null)
                    os.close();
                if (is != null)
                    is.close();
//              if (server != null)
//                  server.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
}
```

客户端

```
public class Client {
    private String[][] lists;
    private String[][] results;//存放结果
    private Vector<ClientThread> clientThreads;//存放线程
    private int clientThreadCount;
    private int wordsCount = 0;
    private int maxRounds = 0;
    private String inputFilePath;
    private String outputFilePath;

    private Client(Vector<ClientThread> clientThreads, int clientThreadCount, String
inputFilePath, String outputFilePath) {
```

```
        this.clientThreads = clientThreads;
        this.clientThreadCount = clientThreadCount;
        this.inputFilePath = inputFilePath;
        this.outputFilePath = outputFilePath;
    }

    private void setLists() {
        this.lists = new String[clientThreadCount][maxRounds];
        this.results = new String[clientThreadCount][maxRounds];
    }

    private void setMaxRounds() throws IOException {
        File file = new File(inputFilePath);
        FileReader fr = new FileReader(file);
        LineNumberReader lnr = new LineNumberReader(fr);
        lnr.skip(Long.MAX_VALUE);
        wordsCount = lnr.getLineNumber() + 1;//获取test.txt中单词个数
        this.maxRounds = (int) Math.ceil((double) wordsCount / clientThreadCount);
        System.out.println("test.txt含有单词数：" + wordsCount);
        System.out.println("所需申请二维数组中的二维长度是：" + maxRounds);

        //动态指定二维数组中的二维长度
        if (this.maxRounds > 0)
            setLists();
    }

    /*
     * 获取需要查询的单词
     */
    private void getWords() {
        try {
            //设置二维数组的二维长度
            setMaxRounds();

            File file = new File(inputFilePath);
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);
            String strs = "";
            int pos = 0;
            int round = 0;
            while ((strs = br.readLine()) != null) {
                lists[pos][round] = strs;
                pos = (pos + 1) % clientThreadCount;
                if (pos == 0)
                    round++;
            }
            br.close();
            fr.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void getResults() {
        int pos = 0;
        while (pos < clientThreadCount) {
            byte[] result = clientThreads.get(pos).getByte();
```

```
            results[pos] = byte2string(result);
            pos++;
        }
    }

    private String[] byte2string(byte[] bytes) {
        String string = new String(bytes);
        String[] result = string.split("\\s+|\n|\0");
        return result;
    }

    /*
     *省略
     */
    private void request() throws IOException {
        int i = 0;
        while (i < clientThreadCount) {
            ClientThread clientThread = new ClientThread(i, lists[i]);
            clientThread.start();
            //添加至数组，便于取用
            clientThreads.add(clientThread);
            i++;
            System.out.println("已启动" + i + "个客户端线程");
        }
    }

    /*
     *省略主函数
     */
}


public class ClientThread extends Thread {
    private static final int CLIENT_PORT = 8899;
    private Socket client;
    private int id;
    private String[] list;
    private byte[] resultWords;

    public byte[] getByte() {
        return resultWords;
    }

    ClientThread(int id, String[] list) throws IOException {
        this.client = new Socket("localhost", CLIENT_PORT);//需创建新的Socket，若在主线程拷贝赋值，则是同一个socket
        this.id = id;
        this.list = list;
        this.resultWords = new byte[1024];
    }

    private byte[] str2byte() {
        String consult = "";
        for (String s : list) {
            consult += s + "\0";
        }
        return consult.getBytes();
```

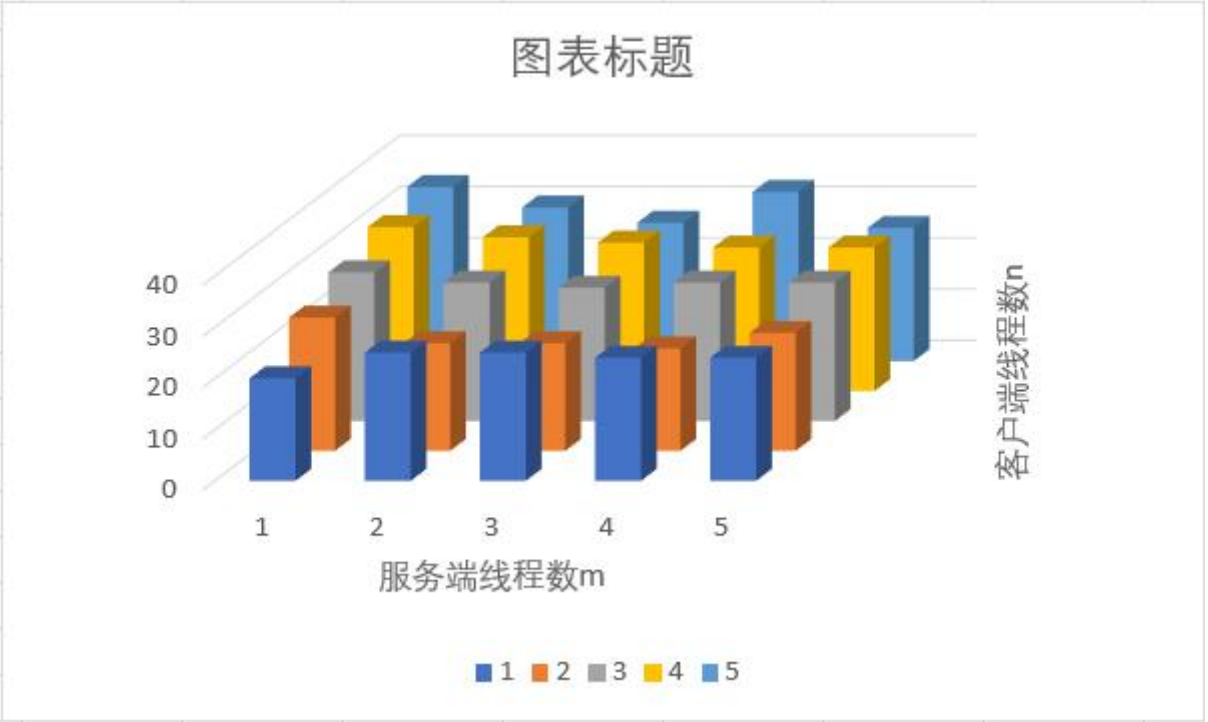
```
    }

    /*
     * 将需要查询的单词发送给服务端
     */
    private void sendWords() {
        try {
            OutputStream os = client.getOutputStream();
            byte[] binaryWords = str2byte();
            os.write(binaryWords);
            System.out.println("send");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /*
     * 接收服务器的查询结果,放入List
     */
    private void getBack() {
        try {
            InputStream is = client.getInputStream();
            is.read(resultWords);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        try {
            System.out.println("Client[port:" + client.getInetAddress() + ",id:" + id +
"]成功连接服务端");
            sendWords();
            getBack();
            if (client != null)
                client.close();
            assert client != null;
            System.out.println("Client[port:" + client.getInetAddress() + ",id:" + id +
"]已退出");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

结果图



经验总结

1. 在参考网络上代码的时候一定要弄明白为什么是这么写，比如此题的bufferreader.readline()，网上的多线程Socket多为类似聊天室的例子，而readline在此时使用是没问题的，因为聊天室在建立连接以后，本就是在while循环内，靠输入"\n"进行信息交互。而此题中，不能这么做是因为客户端服务端线程会始终在等待对方发送数据，造成线程阻塞，从而无法执行下面的代码。所以此题采用字节流一次发送所有单词会更好，至于单词分割，只要在每个单词后面加上标志位（如"\0"）即可。（这个bug找了我将近一个星期。。。）
2. 多多学习多看书。