

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

47 工作实战：Socket 结合线程池的使用

更新时间：2019-12-03 09:54:41



“立志是事业的大门，工作是登堂入室的旅程。——巴斯德”

引导语

Socket 面试最终题一般都是让你写一个简单的客户端和服务端通信的例子，本文就带大家一起来写这个 demo。

1 要求

- 1. 可以使用 Socket 和 ServiceSocket 以及其它 API；
- 2. 写一个客户端和服务端之间 TCP 通信的例子；
- 3. 服务端处理任务需要异步处理；
- 4. 因为服务端处理能力很弱，只能同时处理 5 个请求，当第六个请求到达服务器时，需要服务器返回明确的错误信息：服务器太忙了，请稍后重试~。

需求比较简单，唯一复杂的地方在于第四点，我们需要对客户端的请求量进行控制，首先我们需要确认的是，我们是无法控制客户端发送的请求数的，所以我们只能从服务端进行改造，比如从服务端进行限流。

有的同学可能很快想到，我们应该使用 ServerSocket 的 backlog 的属性，把其设置成 5，但我们在上一章中说到 backlog 并不能准确代表限制的客户端连接数，而且我们还要求服务端返回具体的错误信息，即使 backlog 生效，也只会返回固定的错误信息，不是我们定制的错误信息。

我们好好想想，线程池似乎可以做这个事情，我们可以把线程池的 coreSize 和 maxSize 都设置成 4，把队列大小设置成 1，这样服务端每次收到请求后，会先判断一下线程池中的队列有没

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

正好线程池的加入也可以满足第三点，服务端的任务可以异步执行。

2 客户端代码

客户端的代码比较简单，直接向服务器请求数据即可，代码如下：

```
public class SocketClient {
    private static final Integer SIZE = 1024;
    private static final ThreadPoolExecutor socketPoll = new ThreadPoolExecutor(50, 50,
                                                                              365L,
                                                                              TimeUnit.DAYS,
                                                                              new LinkedBlockingQueue<>(400));

    @Test
    public void test() throws InterruptedException {
        // 模拟客户端同时向服务端发送 6 条消息
        for (int i = 0; i < 6; i++) {
            socketPoll.submit(() -> {
                send("localhost", 7007, "nihao");
            });
        }
        Thread.sleep(1000000000);
    }
    /**
     * 发送tcp
     *
     * @param domainName 域名
     * @param port      端口
     * @param content   发送内容
     */
    public static String send(String domainName, int port, String content) {
        log.info("客户端开始运行");
        Socket socket = null;
        OutputStream outputStream = null;
        InputStreamReader isr = null;
        BufferedReader br = null;
        InputStream is = null;
        StringBuffer response = null;
        try {
            if (StringUtils.isBlank(domainName)) {
                return null;
            }
            // 无参构造器初始化 Socket，默认底层协议是 TCP
            socket = new Socket();
            socket.setReuseAddress(true);
            // 客户端准备连接服务端，设置超时时间 10 秒
            socket.connect(new InetSocketAddress(domainName, port), 10000);
            log.info("TCPClient 成功和服务端建立连接");
            // 准备发送消息给服务端
            outputStream = socket.getOutputStream();
            // 设置 UTF 编码，防止乱码
            byte[] bytes = content.getBytes(Charset.forName("UTF-8"));
            // 输出字节码
            segmentWrite(bytes, outputStream);
            // 关闭输出
            socket.shutdownOutput();
            log.info("TCPClient 发送内容为 {}",content);

            // 等待服务端的返回
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (socket != null) socket.close();
            if (outputStream != null) outputStream.close();
            if (isr != null) isr.close();
            if (br != null) br.close();
            if (is != null) is.close();
        }
        return response.toString();
    }
}
```

<div>← 慕课专栏</div> <div>面试官系统精讲Java源码及大厂真题 / 47 工作实战：Socket 结合线程池的使用</div>	
目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	
	<pre>isr = new InputStreamReader(is); br = new BufferedReader(isr); // 从流中读取返回值 response = segmentRead(br); // 关闭输入流 socket.shutdownInput(); //关闭各种流和套接字 close(socket, outputStream, isr, br, is); log.info("TCPClient 接受到服务端返回的内容为 {}",response); return response.toString(); } catch (ConnectException e) { log.error("TCPClient-send socket连接失败", e); throw new RuntimeException("socket连接失败"); } catch (Exception e) { log.error("TCPClient-send unkown error", e); throw new RuntimeException("socket连接失败"); } finally { try { close(socket, outputStream, isr, br, is); } catch (Exception e) { // do nothing } } } /** * 关闭各种流 * * @param socket * @param outputStream * @param isr * @param br * @param is * @throws IOException */ public static void close(Socket socket, OutputStream outputStream, InputStreamReader isr, BufferedReader br, InputStream is) throws IOException { if (null != socket && !socket.isClosed()) { try { socket.shutdownOutput(); } catch (Exception e) { } try { socket.shutdownInput(); } catch (Exception e) { } try { socket.close(); } catch (Exception e) { } } if (null != outputStream) { outputStream.close(); } if (null != br) { br.close(); } if (null != isr) { isr.close(); } if (null != is) {</pre>

<div>← 慕课专栏</div> <div>面试官系统精讲Java源码及大厂真题 / 47 工作实战：Socket 结合线程池的使用</div>	
目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	
	<pre>/** * 分段读 * * @param br * @throws IOException */ public static StringBuffer segmentRead(BufferedReader br) throws IOException { StringBuffer sb = new StringBuffer(); String line; while ((line = br.readLine()) != null) { sb.append(line); } return sb; } /** * 分段写 * * @param bytes * @param outputStream * @throws IOException */ public static void segmentWrite(byte[] bytes, OutputStream outputStream) throws IOException { int length = bytes.length; int start, end = 0; for (int i = 0; end != bytes.length; i++) { start = i == 0 ? 0 : i * SIZE; end = length > SIZE ? start + SIZE : bytes.length; length -= SIZE; outputStream.write(bytes, start, end - start); outputStream.flush(); } }</pre>
	<p>客户端代码中我们也用到了线程池，主要是为了并发模拟客户端一次性发送 6 个请求，按照预期服务端在处理第六个请求的时候，会返回特定的错误信息给客户端。</p>
	<p>以上代码主要方法是 send 方法，主要处理像服务端发送数据，并处理服务端的响应。</p>
	<h3>3 服务端代码</h3>
	<p>服务端的逻辑分成两个部分，第一部分是控制客户端的请求个数，当超过服务端的能力时，拒绝新的请求，当服务端能力可响应时，放入新的请求，第二部分是服务端任务的执行逻辑。</p>
	<h4>3.1 对客户端请求进行控制</h4>
	<pre>public class SocketServiceStart { /** * 服务端的线程池，两个作用 * 1：让服务端的任务可以异步执行 * 2：管理可同时处理的服务端的请求数 */ private static final ThreadPoolExecutor collectPoll = new ThreadPoolExecutor(4, 4,</pre>

目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

```
1));

@Test
public void test(){
    start();
}

/**
 * 启动服务端
 */
public static final void start() {
    log.info("SocketServiceStart 服务端开始启动");
    try {
        // backlog serviceSocket处理阻塞时，客户端最大的可创建连接数，超过客户端连接不上
        // 当线程池能力处理满了之后，我们希望尽量阻塞客户端的连接
        // ServerSocket serverSocket = new ServerSocket(7007,1,null);
        // 初始化服务端
        ServerSocket serverSocket = new ServerSocket();
        serverSocket.setReuseAddress(true);
        // serverSocket.bind(new InetSocketAddress(InetAddress.getLocalHost().getHostAddress(
        serverSocket.bind(new InetSocketAddress("localhost", 7007));
        log.info("SocketServiceStart 服务端启动成功");
        // 自旋，让客户端一直在取客户端的请求，如果客户端暂时没有请求，会一直阻塞
        while (true) {
            // 接受客户端的请求
            Socket socket = serverSocket.accept();

            // 如果队列中有数据了，说明服务端已经到了并发处理的极限了，此时需要返回客户端有意义的
            if (collectPoll.getQueue().size() >= 1) {
                log.info("SocketServiceStart 服务端处理能力到顶，需要控制客户端的请求");
                //返回处理结果给客户端
                rejectRequest(socket);
                continue;
            }
            try {
                // 异步处理客户端提交上来的任务
                collectPoll.submit(new SocketService(socket));
            } catch (Exception e) {
                socket.close();
            }
        }
    } catch (Exception e) {
        log.error("SocketServiceStart - start error", e);
        throw new RuntimeException(e);
    } catch (Throwable e) {
        log.error("SocketServiceStart - start error", e);
        throw new RuntimeException(e);
    }
}

// 返回特定的错误码给客户端
public static void rejectRequest(Socket socket) throws IOException {
    OutputStream outputStream = null;
    try{
        outputStream = socket.getOutputStream();
        byte[] bytes = "服务器太忙了，请稍后重试~".getBytes(Charset.forName("UTF-8"));
        SocketClient.segmentWrite(bytes, outputStream);
        socket.shutdownOutput();
    }finally {
        //关闭流
        SocketClient.close(socket,outputStream,null,null,null);
    }
}
```

<div>← 慕课专栏</div> <div>≡ 面试官系统精讲Java源码及大厂真题 / 47 工作实战：Socket 结合线程池的使用</div>	
目录	
第1章 基础	
01 开篇词：为什么学习本专栏	我们使用 collectPoll.getQueue().size() >= 1 来判断目前服务端是否已经达到处理的极限了，如果队列中有一个任务正在排队，说明当前服务端已经超负荷运行了，新的请求应该拒绝掉，如果队列中没有数据，说明服务端还可以接受新的请求。
02 String、Long 源码解析和面试题	以上代码注释详细，就不累赘说了。
03 Java 常用关键字理解	3.2 服务端任务的处理逻辑
04 Arrays、Collections、Objects 常用方法源码解析	服务端的处理逻辑比较简单，主要步骤是：从客户端的 Socket 中读取输入，进行处理，把响应返回给客户端。
第2章 集合	我们使用线程沉睡 2 秒来模拟服务端的处理逻辑，代码如下：
05 ArrayList 源码解析和设计思路	<pre>public class SocketService implements Runnable { private Socket socket; public SocketService() { } public SocketService(Socket socket) { this.socket = socket; } @Override public void run() { log.info("SocketService 服务端任务开始执行"); OutputStream outputStream = null; InputStream is = null; InputStreamReader isr = null; BufferedReader br = null; try { //接受消息 socket.setSoTimeout(10000);// 10秒还没有得到数据，直接断开连接 is = socket.getInputStream(); isr = new InputStreamReader(is,"UTF-8"); br = new BufferedReader(isr); StringBuffer sb = SocketClient.segmentRead(br); socket.shutdownInput(); log.info("SocketService accept info is {}", sb.toString()); //服务端处理 模拟服务端处理耗时 Thread.sleep(2000); String response = sb.toString(); //返回处理结果给客户端 outputStream = socket.getOutputStream(); byte[] bytes = response.getBytes(Charset.forName("UTF-8")); SocketClient.segmentWrite(bytes, outputStream); socket.shutdownOutput(); //关闭流 SocketClient.close(socket,outputStream,isr,br,is); log.info("SocketService 服务端任务执行完成"); } catch (IOException e) { log.error("SocketService IOException", e); } } }</pre>
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

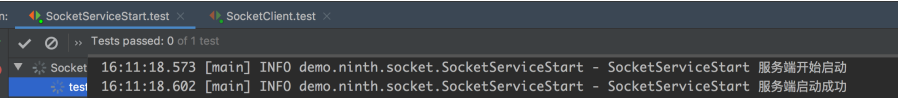
17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

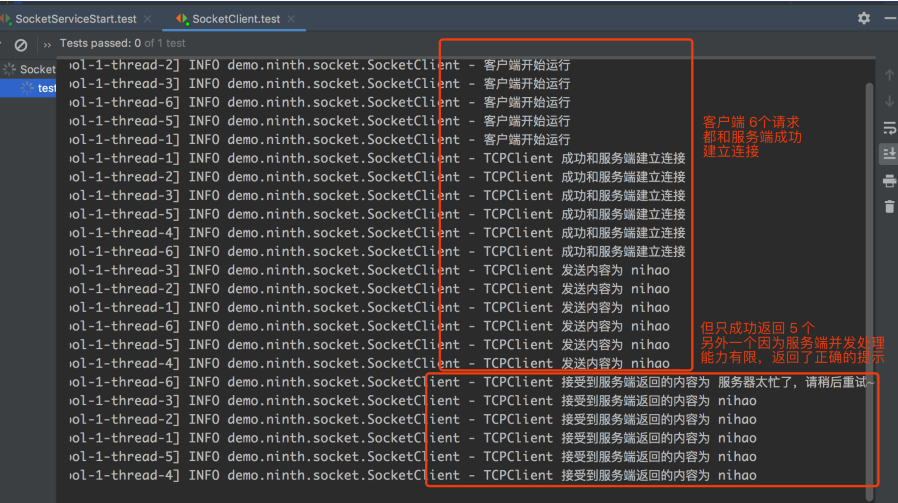
```
try {
    SocketClient.close(socket,outputStream,isr,br,is);
} catch (IOException e) {
    log.error("SocketService IOException", e);
}
}
```

4 测试

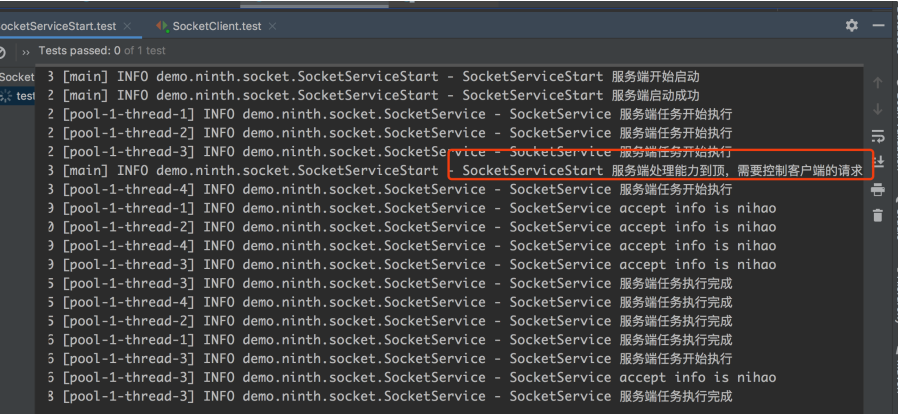
测试的时候，我们必须先启动服务端，然后再启动客户端，首先我们启动服务端，打印日志如下：



接着我们启动客户端，打印日志如下：



我们最后看一下服务端的运行日志：



从以上运行结果中，我们可以看出得出的结果是符合我们预期的，服务端在请求高峰时，能够并发处理5个请求，其余请求可以用正确的提示进行拒绝。

5 总结

← 慕课专栏	面试官系统精讲Java源码及大厂真题 / 47 工作实战：Socket 结合线程池的使用
目录	象。
第1章 基础	← 46 ServerSocket 源码及面试题 48 一起看过的 Java 源码和面试题 真题 →
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	精选留言 0
03 Java 常用关键字理解	欢迎在这里发表留言，作者筛选后可公开显示
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	<div>！</div> <div>目前暂无任何讨论</div>
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	千学不如一看，千看不如一练
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	