**Projecr2 文档**

一：项目简介：

1. **目标：**

实现一个多用户并发模拟仿真系统

**（二）要求：**

1. C/S架构
2. 服务器利用线程池应对并发
3. 客户端模拟上千用户实现对服务器的并发请求

二．项目分析

**（一）.客户端实现**：

采用多线程模拟多客户端，每一个客户端对应于一个线程

调用java封装的CountDownLatch()方法控制客户端多线程可以实现瞬间高并发请求的模拟

**（二）.服务器实现**

搭建一个线程池，对于每一个客户端的请求，都在线程池里调用一个线程去处理请求

三．项目实现

**（一）客户端**

**1.多客户端的模拟：**

for(int i=0;i<=10;i++) {  
 new Thread(() -> {  
 try {  
 *latch*.countDown();  
 System.*out*.println("子线程"+Thread.*currentThread*().getName()+"正在执行");  
 *latch*.await();  
 Thread.*sleep*((long)(Math.*random*()\*10000));  
 CSendMessage cSendMessage = new CSendMessage();  
 cSendMessage.run();  
 /\*System.out.println("子线程"+Thread.currentThread().getName()+"执行完毕");\*/  
  
 } catch (IOException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }).start();

利用for循环控制生成的客户端，每一次for循环都会生成一个新的线程来对服务器发起请求。

**2.瞬间高并发的实现**

**（1）CountDownLatch用法：**

CountDownLatch类位于java.util.concurrent包下，利用它可以实现类似计数器的功能。比如有一个任务A，它要等待其他4个任务执行完毕之后才能执行，此时就可以利用CountDownLatch来实现这种功能了。

　　CountDownLatch类只提供了一个构造器：

|  |  |
| --- | --- |
| 1 | public CountDownLatch(int count) {  };  //参数count为计数值 |

 　　然后下面这3个方法是CountDownLatch类中最重要的方法：

|  |  |
| --- | --- |
| 1  2  3 | public void await() throws InterruptedException { };   //调用await()方法的线程会被挂起  public boolean await(long timeout, TimeUnit unit) };  //和await()类似  public void countDown() { };  //将count值减1 |

**（2）瞬间高并发：**

private static CountDownLatch *latch* = new CountDownLatch(10);  
public static void main(String[] args) throws IOException {  
 for(int i=0;i<=10;i++) {  
 new Thread(() -> {  
 try {  
 *latch*.countDown();  
 System.*out*.println("子线程"+Thread.*currentThread*().getName()+"正在执行");  
 *latch*.await();  
 Thread.*sleep*((long)(Math.*random*()\*10000));  
 CSendMessage cSendMessage = new CSendMessage();  
 cSendMessage.run();  
 /\*System.out.println("子线程"+Thread.currentThread().getName()+"执行完毕");\*/  
  
 } catch (IOException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }).start();

首先利用countdownlatch设置瞬间高并发的线程数目，利用await（）方法使线程数在达到指定目标数之后同时执行，这样就实现了对瞬间高并发请求的模拟。

**（二）服务器端**

**1.线程池,**

Java通过Executors提供四种线程池，分别为：

newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

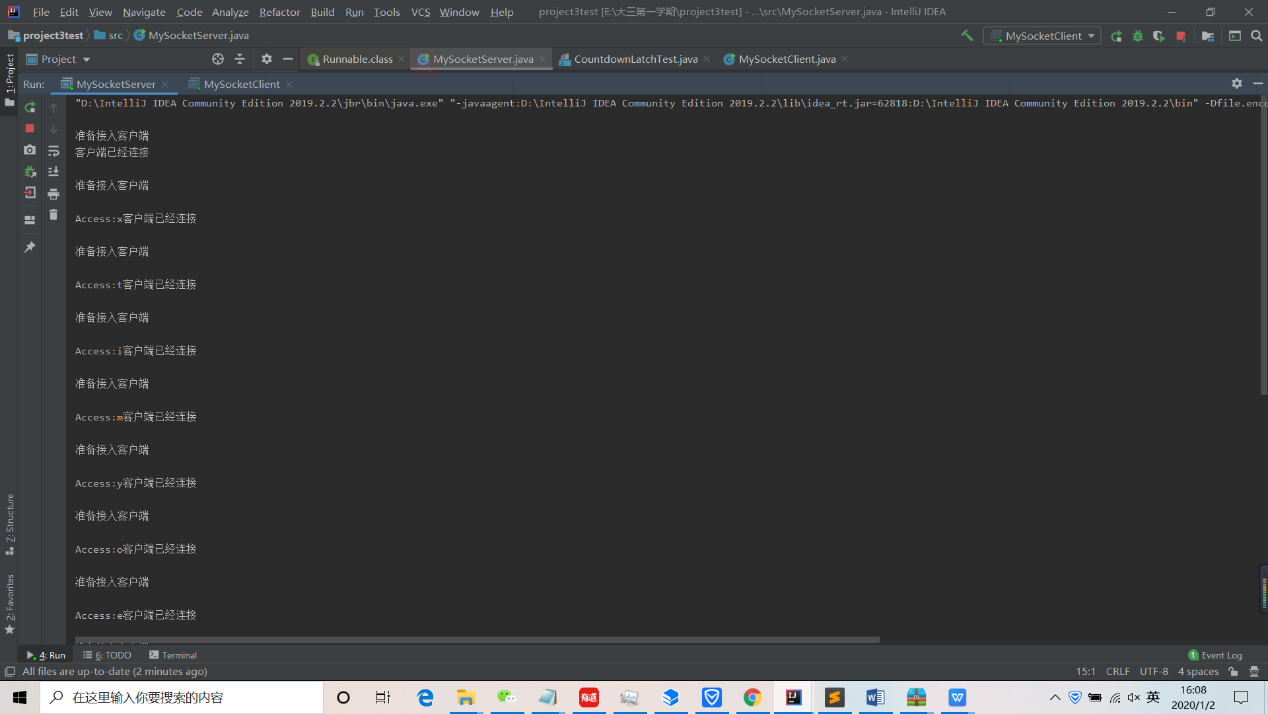
**2.代码**：

try {  
 ServerSocket server = new ServerSocket(20011);  
 ExecutorService executor=Executors.*newFixedThreadPool*(20);  
 boolean flag = true;  
 while (flag) {  
 System.*out*.println("\n准备接入客户端");  
 executor.execute(new Handler(s.accept()));  
 System.*out*.println("客户端已经连接");  
 }  
 server.close();  
} catch (IOException e) {  
 e.printStackTrace();  
}

项目使用了java定长线程池，每当有新线程发起请求的时候会从线程池里调用线程处理请求。

**（三）代码运行结果图：**

客户端：



服务器：

