项目二

要求

- 采用C/S架构
- 客户端模拟并发
- 服务器能够应对并发

分析

- 客户端
  - 为了能够模拟多用户访问服务器，客户端需要开启子线程，每一个子线程代表一个客服端
  - 要想做到多用户同时访问服务器，客户端的每一个子线程就要同时执行连接服务器的操作
- 服务器
  - 每当有一个客户连入服务器时，客户端需要开启一个线程去处理这个连接。
  - Java提供了封装好的线程池方法，可以直接拿来使用

实现

- 客户端
  - CountDownLatch 是Java 1.5引入的一个同步工具类，利用它可以很容易的实现客户端的要求。以下内容来自于博客Java并发编程之CountDownLatch详解：

    > 闭锁是一种同步工具类，可以延迟线程的进度，直到其到达终止状态。闭锁的作用相当于一扇门：在闭锁到达结束状态之前，这扇门一直是关闭的，并且没有任何线程能够通过，当达到结束状态时，这扇门会打开并允许所有的线程通过。当闭锁到达结束状态后，将不会再改变状态，因此这扇门将永远保持打开状态。闭锁可以确保某些活动直到其他活动都完成之后才继续执行。
    >
    > CountDownLatch是一种灵活的闭锁实现，它允许一个或多个线程等待一组事件的产生。闭锁状态包括一个计数器，该计数器初始化为一个正数，表示需要等待的事件数量。CountDownLatch方法递减计数器，表示有一个事件已经发生了，而await方法会一直阻塞直到计数器为0，或者等待中的线程中断，或者等待超时。

  - 根据上面的解释，可以通过以下代码来实现

```
public class Client {
    public static void main(String[] args) throws InterruptedException{
        final int N = 1000;
        CountDownLatch start = new CountDownLatch(1);;
        System.out.println("等待其他线程完成");
        for (int i = 0; i < N; i++) {
            new Thread(new MyRunnable(start)).start();
        }
        System.out.println("创建线程完成");
        start.countDown();
    }
}
```

```
public void run() {
    String string = "Access " + randomLetter();
    try {
        start.await();
    }catch (InterruptedException e){
        e.printStackTrace();
    }
    post(string);
}
```

N代表用户数目，开启的每一个线程，新建一个CountDownLatch 对象start,通过在线程内执行start.await()方法，阻塞用户线程，当所有线程创建完成之后，main方法中执行start.countdown()方法，则会唤醒所有在等待的用户子线程，开始执行post()方法，post()方法即为连接服务器的函数。

- 服务端
  - 服务端采用Java线程池，Java通过Executors提供四种线程池，分比为：
    - 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。
  - 这里我采用的是第一种，可以灵活的根据实际情况来创建线程池，能够更好的利用资源。
  - 以下为服务器的关键部分代码：

```
int number = 0;
    try {
        ServerSocket serverSocket = new ServerSocket(33333);
        System.out.println("***服务器即将启动，等待客户端链接***");
        executorService = Executors.newCachedThreadPool();
        while (true){
            Socket socket = serverSocket.accept();
            number++;
            executorService.submit(new UserThread(number,socket));
        }
    }catch (Exception e){
        System.out.println(number);
        e.printStackTrace();
```

```
    }
```

利用submit()方法，可以将创建的线程交由线程池执行，UserThread(number,socket)是处理用户请求的方法。

以上便实现了项目二。