**并发生成重复订单号**

思路，生成订单号：时间+序号

|  |
| --- |
| **package** com.locks;  **import** java.text.SimpleDateFormat; **import** java.util.Date;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* **public class** OrderCodeGenerator {  *// 自增长序列* **private static int** *i* = 0;   *//生成订单号* **public** String getOrderCode() {  Date date = **new** Date();  SimpleDateFormat sdf = **new** SimpleDateFormat(**"yyyy-MM-dd-HH-mm-ss-"**);  **return** sdf.format(date) + ++*i*;  }   **public static void** main(String[] args){  OrderCodeGenerator orderCodeGenerator = **new** OrderCodeGenerator();  **for** (**int** i = 0; i < 10; i++) {  System.***out***.println(orderCodeGenerator.getOrderCode());  }  } } |

多线程同时生成订单号，出现并发问题会生成相同的订单号

|  |
| --- |
| **package** com.locks;  **import** java.util.concurrent.CountDownLatch;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* **public class** OrderServiceImpl **implements** Runnable {   **private static** OrderCodeGenerator *ong* = **new** OrderCodeGenerator();   *//同时并发的线程数* **private static final int *NUM*** = 10;  *//按照线程数初始化倒计数器，倒计数器* **private static** CountDownLatch *cdl* = **new** CountDownLatch(***NUM***);   *// 创建订单接口* **public void** createOrder() {  String orderCode = **null**;  *//获取订单编号* orderCode = *ong*.getOrderCode();  *//业务代码* System.***out***.println(Thread.*currentThread*().getName() + **"====>"** + orderCode);   }   */\*\*  \* When an object implementing interface <code>Runnable</code> is used  \* to create a thread, starting the thread causes the object's  \* <code>run</code> method to be called in that separately executing  \* thread.  \* <p>  \* The general contract of the method <code>run</code> is that it may  \* take any action whatsoever.  \*  \** ***@see*** *Thread#run()  \*/* @Override  **public void** run() {  **try** {  *//等待其他线程初始化（只有线程数达到预定线程才会同时执行方法）  cdl*.await();  } **catch** (Exception e) {  e.printStackTrace();  }  *//创建订单* createOrder();  }   **public static void** main(String[] args) {  **for** (**int** i = 0; i < ***NUM***; i++) {  **new** Thread(**new** OrderServiceImpl()).start();  *cdl*.countDown();  }  } } |

|  |
| --- |
| 打印结果：  Thread-5====>2020-04-07-20-45-11-1  Thread-6====>2020-04-07-20-45-11-2  Thread-1====>2020-04-07-20-45-11-4  Thread-7====>2020-04-07-20-45-11-3  Thread-4====>2020-04-07-20-45-11-3  Thread-0====>2020-04-07-20-45-11-5  Thread-3====>2020-04-07-20-45-11-6  Thread-2====>2020-04-07-20-45-11-7  Thread-8====>2020-04-07-20-45-11-8  Thread-9====>2020-04-07-20-45-11-9  Process finished with exit code 0 |

可以看出线程4和线程7生成了一样的订单号，所以此方法存在问题。

优化一：

|  |
| --- |
| **加锁：**  **private static** Lock *lock* = **new** ReentrantLock(); |

方法如下：实用单体应用（分布式项目不可用）

|  |
| --- |
| **package** com.locks;  **import** java.util.concurrent.CountDownLatch; **import** java.util.concurrent.locks.Lock; **import** java.util.concurrent.locks.ReentrantLock;  */\*\*  \** ***@author*** *iszengziqiang@163.com  \** ***@date*** *2020/4/7 11:31  \** ***@Version:*** *1.0  \** ***@desc*** *//****todo*** *\*/* **public class** OrderServiceImpl **implements** Runnable {   **private static** OrderCodeGenerator *ong* = **new** OrderCodeGenerator();   *//同时并发的线程数* **private static final int *NUM*** = 10;  *//按照线程数初始化倒计数器，倒计数器* **private static** CountDownLatch *cdl* = **new** CountDownLatch(***NUM***);   **private static** Lock *lock* = **new** ReentrantLock();   *// 创建订单接口* **public void** createOrder() {  String orderCode = **null**;   *lock*.lock();  **try** {  *//获取订单编号* orderCode = *ong*.getOrderCode();  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  *lock*.unlock();  }   *//业务代码* System.***out***.println(Thread.*currentThread*().getName() + **"====>"** + orderCode);   }   */\*\*  \* When an object implementing interface <code>Runnable</code> is used  \* to create a thread, starting the thread causes the object's  \* <code>run</code> method to be called in that separately executing  \* thread.  \* <p>  \* The general contract of the method <code>run</code> is that it may  \* take any action whatsoever.  \*  \** ***@see*** *Thread#run()  \*/* @Override  **public void** run() {  **try** {  *//等待其他线程初始化  cdl*.await();  } **catch** (Exception e) {  e.printStackTrace();  }  *//创建订单* createOrder();  }   **public static void** main(String[] args) {  **for** (**int** i = 0; i < ***NUM***; i++) {  **new** Thread(**new** OrderServiceImpl()).start();  *cdl*.countDown();  }  } } |

优化二：（分布式锁）

<https://github.com/redisson/redisson/wiki/1.-%E6%A6%82%E8%BF%B0>