当主线程崩溃而其它线程继续运行时发生什么（1） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/264/2021_2022__E5_BD_93_E4_B8_BB_E7_BA_BF_E7_c104_264324.htm 在多线程代码中，使用驱动其它线程所负责的动作的单个主线程是常见的。这个主线程发送消息，通常是通过把它们放到一个队列中，然后其它线程处理这些消息。但是如果主线程抛出一个异常，那么剩余的线程会继续运行，等待更多输入到该队列，导致程序冻结。在诊断 Java 代码的这一部分中，专职 Java 开发者兼兼职捉虫者 Eric Allen 讨论检测、修复和避免这一错误模式。请在讨论论坛与作者和其他读者共享您关于本文的心得。用多线程编写代码对程序员大有好处。多线程能使编程（和程序）进行得快得多，而且代码能有效得多地使用资源。然而，跟生活中的很多事情一样，多线程也存在缺点。因为多线程代码天生是非确定性的，出现错误的可能性大得多。而且，确实发生的的错误很难重现，因此也更难解决。 孤线程模式 Java 编程语言为多线程代码提供了丰富的支持，包括一项特别有用的功能：能够在一个线程中抛出一个异常而不影响其它线程。但这项功能会导致很多难以跟踪的错误。 快速跟踪代码 清单 1. 一个线程之间频繁通信的示例程序 清单 2. 演示如何捕捉异常并在退出之前通知问题的依赖线程。 从某个线程的崩溃中恢复过来是有意义，在此种情况下，这种能力能增加程序的健壮性级别。然而，它也使我们难以判断这些线程之一在什么时候抛出了一个异常。因为剩余的线程将继续运行，所以程序会表现出无响应或冻结程序的征兆。对线程之间频繁通信的程序而言尤其如此。 考虑清单 1 所示的示

例，其中的一对线程通过生产者-消费者模型进行通信。 清单 1. 一个简单的、多线程的消费者-生产者程序 public class Server extends Thread { Client client. int counter. public Server(Client _client) { this.client = _client. this.counter = 0. } public void run() { while (counter this.client.queue.addElement(new Integer(counter)). counter . } throw new RuntimeException("counter >= 10"). } public static void main(String[] args) { Client c = new Client(). Server s = new Server(c). c.start(). s.start(). } } class Client extends Thread { Vector queue. public Client() { this.queue = new Vector(). } public void run() { while (true) { if (! (queue.size() == 0)) { processNextElement(). } } } private void processNextElement() { Object next = queue.elementAt(0). queue.removeElementAt(0). System.out.println(next). } } 在诸如这样的案例中，第二个线程接收用于计算的任何数据完全依赖于第一个线程。因此，不可避免地，如果第一个线程崩溃（而在这个样本中，肯定是这样的），那么第二个线程将等待永远不会到来的更多输入。现在您知道我为什么把这种错误叫做孤线程模式。 症状 这种错误模式最常见的症状是我在前面提到的？即，程序好象冻结了。 其它症状可能包括打印到程序标准错误和标准输出的堆栈跟踪实际停止了。 治疗和预防措施 一旦诊断出这种错误模式，查找并修复在崩溃线程中的潜在的错误是显然的治疗之道。但是预防却困难得多。 不用说，如果您使用单线程设计就能侥幸成功，那么将可以免除很多头痛的事情。然而，当程序的性能要求是必须考虑的问题时，就要首先考虑使用多线程设计。 