Java实时多任务调度过程中的安全监控设计 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/287/2021_2022_Java_E5_AE_9E_E6_97_B6_c104_287859.htm 在一系列关联的多任务的实时环境中，如果有一个任务发生失败，可能导致所有任务产生连锁反应，从而造成调度失控的局面。特别是对于核心控制设备尤其重要，为了解决这个问题，必须对每个任务进行实时监控。1. 问题分析 在JAVA环境中，一个任务一般是由一个独立线程来引导实现的，独立线程可能调用一系列子线程。如果在执行过程中，某一个线程发生异常（产生的原因很多，比如软件升级、运行环境改变、系统资抢占等），那么该线程就会停止运行，直到下次任务重新被提交。对于实时环境来说当前任务是失败的。我们无法预测和完全避免异常的发生，但是可以通过一些技术手段来跟踪任务的状态，从而及时发现问题并恢复正常，减少损失。2. 设计原理 对于一个实时任务而言，执行效率是非常关键的，这意味着不可能考虑用非常复杂的方式来实现任务监控，即使这样可以做的比较完善，同时监控代码本身也会引入一些异常，这就要求监控程序必须简单可靠，能够发现大多数问题，并能及时处理。一个可能的简单实现。 我们对每个任务加上一个监控的"壳"，调度程序调用这个"壳"来完成对具体任务的引导和监控，相当于每个任务具有自治能力。这样做的好处有： 分散控制。不需要修改调度主体程序，不增加调度过程的复杂度；控制灵活，安全性高。对于不同任务可定义不同控制方式和控制参数，封装在任务内部，灵活性好，对个别任务控制代码的修改不会影响其他任务，即任务控制实现松藕合，避免

错误扩散。适合团队开发；维护简单，升级方便。对于新的任务加入也无需改变原来调度程序的结构，只需修改任务表及相关参数，这样在性能提高的同时也简化了软件升级过程中的代码维护量。 3. 设计实现 每个线程理论上有四种状态：new 线程对象已经创建，但尚未启动，不可运行 runnable 一旦有时间分片机制有空闲的CPU周期，线程立即开始运行 dead 从run()方法退出后，一个线程即消亡 blocked 线程可运行，但有某种东西阻碍了它。调度机制不给它分配任何CPU时间，直到它进入runnable状态 在实际操作中，为了便于描述，我们重新规定了线程的状态： Actived 线程已被激活，处于运行状态 Sleeping 线程完成一个特定任务后退出，进入休眠状态 Dead 线程运行过程中发生异常，终止运行，处于死亡状态 根据上述理论，我们只需要对Actived状态的线程进行监控，也只有对Actived状态监控才有意义，这是对监控模块做出逻辑简化。那么如何实现监控模块对具体任务的监控呢？对具体任务的监控方式有多种，根据任务的不同，需要采用不同的监控代码，但是在结构上基本相同。这和类的重载概念有点相似。本文附有一个简单的例子。 A任务每秒执行一个简单的代数运算 j = 1/ i，并打印结果。我们故意在其中设置了一个异常陷阱，使得执行过程中出现一次被0除的算术异常，下面结合这个例子讲述监控原理。 为了监控A，我们设计了一个监控线程M。M中定义了一些关键逻辑变量：Keepchecking 持续监控标志 laststatus 保存上次监控状态 maydeadtimes 监控线程可能死亡的计数器 maydeadtimeout 定义判断线程死的边界条件 deadtimes 监控线程死亡次数的计数器 deadtimeout 定义判断线程不正常的边界条件 为了适应监控

，在A任务中相应增加一些可以被监控的状态和行为： dead 死状态标志 dead = !dead. 改变状态 整个监控就是围绕这些状态标志和行为展开的。A任务定期修改自己的dead标志，dead是一个布尔变量，理论上只要A没有死，那么dead肯定是周期变化的（和心跳概念差不多），M需要做的就是监控dead的变化。为了避免一些偶然因素导致的误判，我们在M中设置了一些计数器和边界值（maydeadtimes和maydeadtimeout），当M发现A的可能死亡次数超过一定限制后，判断A已死亡，不在继续等待了，作为实时处理，首先注销A的实例，然后重新启动A（和我们计算机死机的复位很像），然后进入新一轮监控。 如果是因为系统偶然因素导致A死亡，那么在随后的新的任务启动过程中一般可以顺利完成。但是万一由于环境参数改变或软件升级存在版本缺陷，A可能始终会产生异常，那么M是否需要耐心地监控下去呢？一个形象的例子是：如果你连续3次开机都失败，你是否会怀疑机器有问题？当然，你会，那么M也应该会。 为了对A任务重复多次死亡有一个统计，M中又引入了另外对计数器和边界值（deadtimes和deadtimeout），和你开计算机的过程一样，如果连续n次都发现A有问题，可以基本肯定不是由于偶然因素引起的，需要对A的代码或系统的环境进行检查。M会发出告警，通知必须要对A进行审查了，然后清空A，自己自动安全退出。如果在核心调度程序中设置一个标志接受M们的告警，就可以有足够理由终止其他任务的执行。可以看见，在A任务发生异常期间，M承担了核心调度程序的维护功能。特别是当任务数量比较多的情况，核心调度程序只能采用排队方式处理任务异常，而且由于处理异常的复杂程度不同，无

法保证对多任务异常的实时处理。 还要考虑正常情况下A和M的关系。核心调度程序通过M启动A任务后，M处于持续监控状态，当A正常结束任务后，A需要通知M结束监控，这样，当A进入休眠状态后，M也不会占用内存空间，提高了系统资源的利用率。 通过以上描述，可以看到，上述监控思想具有清晰的概念和可操作性，占用资源少，为保证系统连续稳定运行创造了条件。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com