Linux系统内核定时器机制详解（下）（4） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144973.htm 7．6．4．8 扫描并执行当前已经到期的定时器 函数run_timer_list()完成这个功能。如前所述，该函数是被timer_bh()函数所调用的，因此内核定时器是在时钟中断的Bottom Half中被执行的。记住这一点非常重要。全局变量timer_jiffies表示了内核上一次执行run_timer_list()函数的时间，因此jiffies与timer_jiffies的差值就表示了自从上一次处理定时器以来，期间一共发生了多少次时钟中断，显然run_timer_list()函数必须为期间所发生的每一次时钟中断补上定时器服务。该函数的源码如下（kernel/timer.c）：static inline void run_timer_list(void) { spin_lock_irq(amp.amp.timerlist_lock). fn(data). spin_lock_irq(amp. TVR_MASK. } spin_unlock_irq(&amp.timerlist_lock). }函数run_timer_list()的执行过程主要就是用一个大while{}循环来为时钟中断执行定时器服务，每一次循环服务一次时钟中断。因此一共要执行（jiffies－timer_jiffies＋1）次循环。循环体所执行的服务步骤如下：（1）首先，判断tv1.index是否为0，如果为0则需要从tv2中补充定时器到tv1中来。但tv2也可能为空而需要从tv3中补充定时器，因此用一个do{}while循环来调用cascade_timer()函数来依次视需要从tv2中补充tv1，从tv3中补充tv2、…、从tv5中补充tv4。显然如果tvi.index=0（2≤i≤5），则对于tvi执行cascade_timers()函数后，tvi.index肯定为1。反过来讲，如果对tvi执行过cascade_timers()函数后tvi.index不

等于1，那么可以肯定在未对tvi执行cascade_timers()函数之前，tvi.index值肯定不为0，因此这时tvi不需要从tv(i 1)中补充定时器，这时就可以终止do{}while循环。（2）接下来，就要执行定时器向量tv1.vec［tv1.index］中的所有到期定时器。因此这里用一个goto repeat循环从头到尾依次扫描整个定时器对列。由于在执行定时器的关联函数时并不需要关CPU中断，所以在用detach_timer()函数将当前定时器从对列中摘除后，就可以调用spin_unlock_irq()函数进行解锁和开中断，然后在执行完当前定时器的关联函数后重新用spin_lock_irq（）函数加锁和关中断。（3）当执行完定时器向量tv1.vec[tv1.index]中的所有到期定时器后，tv1.vec［tv1.index］应该是个空队列。至此这一次定时器服务也就宣告结束。（4）最后，将timer_jiffies值加1，将tv1.index值加1，当然它的模是256。然后，回到while循环开始下一次定时器服务。