Linux系统内核定时器机制详解（上）（1） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144898.htm Linux内核2.4版中去掉了老版本内核中的静态定时器机制，而只留下动态定时器。相应地在timer_bh()函数中也不再通过run_old_timers()函数来运行老式的静态定时器。动态定时器与静态定时器这二个概念是相对于Linux内核定时器机制的可扩展功能而言的，动态定时器是指内核的定时器队列是可以动态变化的，然而就定时器本身而言，二者并无本质的区别。考虑到静态定时器机制的能力有限，因此Linux内核2.4版中完全去掉了以前的静态定时器机制。 7．6．1 Linux内核对定时器的描述 Linux在include/linux/timer.h头文件中定义了数据结构timer_list来描述一个内核定时器： struct timer_list { struct list_head list. unsigned long expires. unsigned long data. void (*function)(unsigned long). }.各数据成员的含义如下：（1）双向链表元素list：用来将多个定时器连接成一条双向循环队列。（2）expires：指定定时器到期的时间，这个时间被表示成自系统启动以来的时钟滴答计数（也即时钟节拍数）。当一个定时器的expires值小于或等于jiffies变量时，我们就说这个定时器已经超时或到期了。在初始化一个定时器后，通常把它的expires域设置成当前expires变量的当前值加上某个时间间隔值（以时钟滴答次数计）。（3）函数指针function：指向一个可执行函数。当定时器到期时，内核就执行function所指定的函数。而data域则被内核用作function函数的调用参数。内核函数init_timer()用来初始化一个定时器。实际上，这个初

始化函数仅仅将结构中的list成员初始化为空。如下所示（include/linux/timer.h）：static inline void init_timer(struct timer_list * timer) { timer->list.next = timer->list.prev = NULL. }由于定时器通常被连接在一个双向循环队列中等待执行（此时我们说定时器处于pending状态）。因此函数time_pending()就可以用list成员是否为空来判断一个定时器是否处于pending状态。如下所示（include/linux/timer.h）：static inline int timer_pending (const struct timer_list * timer) { return timer->list.next != NULL. }时间比较操作 在定时器应用中经常需要比较两个时间值，以确定timer是否超时，所以Linux内核在timer.h头文件中定义了4个时间关系比较操作宏。这里我们说时刻a在时刻b之后，就意味着时间值a≥b。Linux强烈推荐用户使用它所定义的下列4个时间比较操作宏（include/linux/timer.h）：#define time_after(a,b) ((long)(b) - (long)(a) = 0) #define time_before_eq(a,b) time_after_eq(b,a)