努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处
http://blog.csdn.net/woshixingaaa/archive/2011/06/03/6525504.aspx
由于计算机在工作时不可避免的要受到各种各样因素的干扰，即使再优秀的计算机程序也可能因为这种干扰使计算机进入一个死循环，更严重的就是导致死机。有两种方法来处理这种情况，一是采用人工复位的方法，而是依赖某种硬件来执行这个复位工作。这种硬件通常叫做看门狗。S3C2440A 处理器内部集成了一个看门狗硬件，如下图：

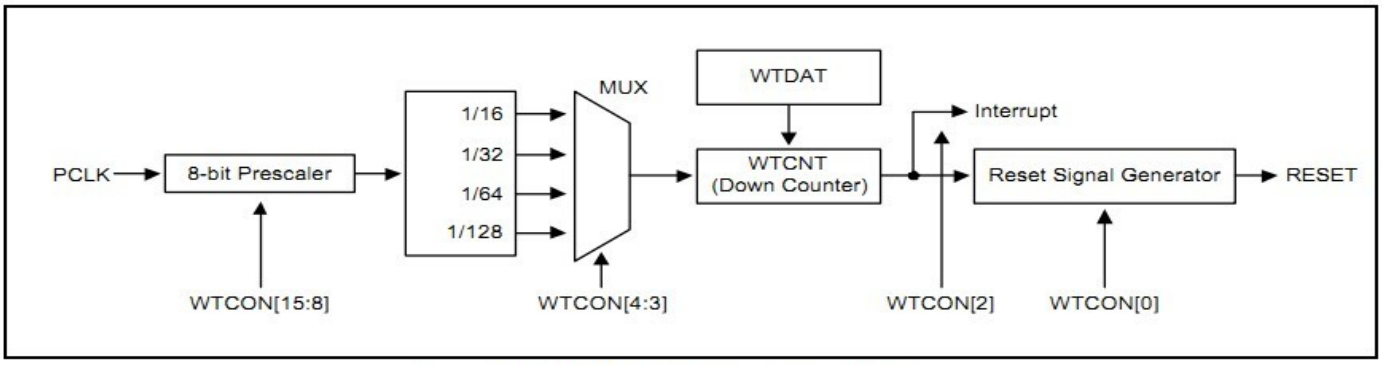

Figure 18-1. Watchdog Timer Block Diagram

看门狗有两个功能，即可作为一个正常的 16 位的内部寄存器来请求中断服务，也可产生 128PCLK 周期的复位信号。下面分析一下看门狗工作过程：首先处理器提供的 PCLK 经过 8 位预分频(这个值在 WTCON[15:8]), 然后通过一个多路选择器，可以选择 4 种分频方式，16，32，64，128。看门狗通过 WTCON[4:3]来选择哪种分频方式，这样可以产生一个时钟频率，每一个时钟周期 WTCNT 中的值就减 1，直到为 0。当为 0 时，如果 WTCON[2]为 1，则产生一个中断信号；如果 WTCON[0]为 1,则产生一个复位信号。看门狗的工作频率=PCLK/(WTCON[15:8]+1)/divider(divider=(16,32,64,128))。公式中 WTCON[15:8]+1 是因为 WTCONT[15:8]的取值范围为 0~255，因为除数不能为 0，所以设计者规定需要加 1。divider 的值由 WTCONT 的第 3,4 位决定，可以取值 16,32,64 和 128。
下面来分析 Linux 内核中看门狗的源码，其实我觉得看门狗相当简单了，因为一共就 3 个寄存器，来看一些全局变量，这些在一些函数中会用到：

```
1. #define CONFIG_S3C2410_WATCHDOG_ATBOOT     (0)
2. #define CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME    (15)
3.
4. static int nowayout = WATCHDOG_NOWAYOUT;
5. static int tmr_margin    = CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME;
6. static int tmr_atboot    = CONFIG_S3C2410_WATCHDOG_ATBOOT;
7. static int soft_noboot;
8. static int debug;
```

nowayout 表示决不允许看门狗关闭，为 1 表示不允许关闭，为 0 表示允许关闭；tmr_margin 表示默认的看门狗喂狗时间为 15s；tmr_atboot 表示系统启动时就

使能看门狗，为 1 表示使能，为 0 表示关闭；soft_noboot 表示看门狗工作的方式，看门狗可以作为定时器使用也可作为复位硬件使用，soft_noboot 为 1 表示看门狗作为定时器使用，不发送复位信号；debug 表示是否使用调试模式来调试代码，该模式中，会打印调试信息。
WATCHDOG_NOWAYOUT 的值由配置选项
CONFIG_WATCHDOG_NOWAYOUT 决定，其宏定义如下：

```
#ifdef CONFIG_WATCHDOG_NOWAYOUT
#define WATCHDOG_NOWAYOUT   1
#else
#define WATCHDOG_NOWAYOUT   0
#endif
```

另外一个重要的枚举值 close_state 用来标识看门狗是否允许关闭，其定义如下：

```
typedef enum close_state {
    CLOSE_STATE_NOT,
    CLOSE_STATE_ALLOW = 0x4021
} close_state_t;
```

CLOSE_STATE_NOT 表示不允许关闭看门狗，CLOSE_STATE_ALLOW 表示允许关闭看门狗。
S3C2410 的看门狗同时具有多重身份：字符设备，混杂设备，平台设备。下面看一下看门狗驱动作为平台驱动的描述：

```
static struct platform_driver s3c2410wdt_driver = {
    .probe      = s3c2410wdt_probe,
    .remove     = s3c2410wdt_remove,
    .shutdown   = s3c2410wdt_shutdown,
    .suspend    = s3c2410wdt_suspend,
    .resume     = s3c2410wdt_resume,
    .driver     = {
        .owner  = THIS_MODULE,
        .name   = "s3c2410-wdt",
    },
};
```

下面是看门狗驱动作为平台驱动的注册：

```
static char banner[] __initdata =
    KERN_INFO "S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics/n";

static int __init watchdog_init(void)
{
    printk(banner);
    return platform_driver_register(&s3c2410wdt_driver);
}
```

```
9.
10.static void __exit watchdog_exit(void)
11.{
12.    platform_driver_unregister(&s3c2410wdt_driver);
13.}
14.
15.module_init(watchdog_init);
16.module_exit(watchdog_exit);
```

下面是看门狗平台设备及其资源：

```
1.  /* Watchdog */
2.  static struct resource s3c_wdt_resource[] = {
3.      [0] = {
4.          /*看门狗 I/O 内存开始位置，被定义为 WTCON 的地址 0x53000000*/
5.          .start = S3C24XX_PA_WATCHDOG,
6.          .end   = S3C24XX_PA_WATCHDOG + S3C24XX_SZ_WATCHDOG - 1,
7.                      /*1M 的地址空间*/
8.          .flags = IORESOURCE_MEM,        /*I/O 内存资源*/
9.      },
10.     [1] = {
11.         .start = IRQ_WDT,       /*看门狗的开始中断号，被定义为 80*/
12.         .end   = IRQ_WDT,       /*看门狗的结束中断号*/
13.         .flags = IORESOURCE_IRQ,    /*中断的 IRQ 资源*/
14.     }
15.
16.};
17.
18.struct platform_device s3c_device_wdt = {
19.     .name           = "s3c2410-wdt",
20.     .id         = -1,
21.     .num_resources      = ARRAY_SIZE(s3c_wdt_resource),   //资源的数目
22.     .resource       = s3c_wdt_resource,
23.};
```

当使用 platform_bus_type 的 platform_match 对 s3c2410wdt_driver 与 s3c_device_wdt 匹配成功时，调用 s3c2410_wdt_probe 函数，下面来看 s3c2410_wdt_probe 的实现：

```
1.  static int s3c2410wdt_probe(struct platform_device *pdev)
2.  {
3.      struct resource *res;
4.      struct device *dev;
5.      unsigned int wtcon;
6.      int started = 0;
7.      int ret;
8.      int size;
9.
10.     DBG("%s: probe=%p/n", __func__, pdev);
```

```
11.  /*平台设备中的设备结构体 device*/
12.  dev = &pdev->dev;
13.  wdt_dev = &pdev->dev;
14.
15.  /* get the memory region for the watchdog timer */
16.  /*获得看门狗的内存资源*/
17.  res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
18.  /*获取失败则推出*/
19.  if (res == NULL) {
20.      dev_err(dev, "no memory resource specified/n");
21.      return -ENOENT;
22.  }
23.  /*内存资源所占的字节数*/
24.  size = (res->end - res->start) + 1;
25.  /*申请一块 IO 内存，对应看门狗的 3 个寄存器*/
26.  wdt_mem = request_mem_region(res->start, size, pdev->name);
27.  if (wdt_mem == NULL) {
28.      dev_err(dev, "failed to get memory region/n");
29.      ret = -ENOENT;
30.      goto err_req;
31.  }
32.  /*获得虚拟地址*/
33.  wdt_base = ioremap(res->start, size);
34.  if (wdt_base == NULL) {
35.      dev_err(dev, "failed to ioremap() region/n");
36.      ret = -EINVAL;
37.      goto err_req;
38.  }
39.
40.  DBG("probe: mapped wdt_base=%p/n", wdt_base);
41.  /*申请中断*/
42.  wdt_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
43.  if (wdt_irq == NULL) {
44.      dev_err(dev, "no irq resource specified/n");
45.      ret = -ENOENT;
46.      goto err_map;
47.  }
48.  /*并注册中断处理函数*/
49.  ret = request_irq(wdt_irq->start, s3c2410wdt_irq, 0, pdev->name, pdev);
50.  if (ret != 0) {
51.      dev_err(dev, "failed to install irq (%d)/n", ret);
52.      goto err_map;
53.  }
54.  /*获得看门狗时钟*/
55.  wdt_clock = clk_get(&pdev->dev, "watchdog");
```

```
56.  if (IS_ERR(wdt_clock)) {
57.      dev_err(dev, "failed to find watchdog clock source/n");
58.      ret = PTR_ERR(wdt_clock);
59.      goto err_irq;
60.  }
61.  /时钟使能/
62.  clk_enable(wdt_clock);
63.
64.  /* see if we can actually set the requested timer margin, and if
65.   * not, try the default value */
66.  /*设置看门狗复位时间，如果成功返回 0*/
67.  if (s3c2410wdt_set_heartbeat(tmr_margin)) {
68.      /*如果不成功，看门狗的复位时间设置成默认值*/
69.      started = s3c2410wdt_set_heartbeat(
70.                  CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME);
71.      if (started == 0)
72.          dev_info(dev,
73.              "tmr_margin value out of range, default %d used/n",
74.                  CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME);
75.      else
76.          dev_info(dev, "default timer value is out of range, cannot start/n");
77.  }
78.  /*注册混杂设备*/
79.  ret = misc_register(&s3c2410wdt_miscdev);
80.  if (ret) {
81.      dev_err(dev, "cannot register miscdev on minor=%d (%d)/n",
82.          WATCHDOG_MINOR, ret);
83.      goto err_clk;
84.  }
85.  /*如果开机的时候就启动看门狗*/
86.  if (tmr_atboot && started == 0) {
87.      dev_info(dev, "starting watchdog timer/n");
88.      s3c2410wdt_start();              //启动看门狗
89.  } else if (!tmr_atboot) {
90.      /* if we're not enabling the watchdog, then ensure it is
91.       * disabled if it has been left running from the bootloader
92.       * or other source */
93.
94.      s3c2410wdt_stop();               //关闭看门狗
95.  }
96.  /* print out a statement of readiness */
97.  /*读出看门狗控制寄存器的值*/
98.  wtcon = readl(wdt_base + S3C2410_WTCON);
99.  /*打印看门狗是否使能，是否允许发出复位信号，是否允许发出中断信号*/
100.   dev_info(dev, "watchdog %sactive, reset %sabled, irq %sabled/n",
```

```
        (wtcon & S3C2410_WTCON_ENABLE) ?  "" : "in",
        (wtcon & S3C2410_WTCON_RSTEN) ? "" : "dis",
        (wtcon & S3C2410_WTCON_INTEN) ? "" : "en");

    return 0;
err_clk:
    clk_disable(wdt_clock);
    clk_put(wdt_clock);
err_irq:
    free_irq(wdt_irq->start, pdev);
err_map:
    iounmap(wdt_base);
err_req:
    release_resource(wdt_mem);
    kfree(wdt_mem);
    return ret;
}
```

下面看一下刚才调用的那个设置看门狗复位时间的方法：

```
static int s3c2410wdt_set_heartbeat(int timeout)
{
    /*首先获得看门狗的时钟*/
    unsigned int freq = clk_get_rate(wdt_clock);
    unsigned int count;
    unsigned int divisor = 1;
    unsigned long wtcon;

    if (timeout < 1)
        return -EINVAL;
    /divider 为 128/
    freq /= 128;
    /*秒数乘以每秒的时间滴答等于计数值*/
    count = timeout * freq;

    DBG("%s: count=%d, timeout=%d, freq=%d/n",
        __func__, count, timeout, freq);

    /* if the count is bigger than the watchdog register,
       then work out what we need to do (and if) we can
       actually make this value
    */
    /*计数值不能大于 WTCNT 的最大范围，WTCNT 是一个 16 位的计数器，最大值是 0x10000*/
    if (count >= 0x10000) {
        /*0x100 为 256，即从 1~256 找一个合适的预分频值*/
        for (divisor = 1; divisor <= 0x100; divisor++) {
```

```
        if ((count / divisor) < 0x10000)
            break;
    }
    /*未找到返回错误*/
    if ((count / divisor) >= 0x10000) {
        dev_err(wdt_dev, "timeout %d too big/n", timeout);
        return -EINVAL;
    }
  }
  /*看门狗的喂狗时间*/
  tmr_margin = timeout;

  DBG("%s: timeout=%d, divisor=%d, count=%d (%08x)/n",
      __func__, timeout, divisor, count, count/divisor);
  /*分频后最终的计数值*/
  count /= divisor;
  wdt_count = count;

  /* update the pre-scaler */
  wtcon = readl(wdt_base + S3C2410_WTCON);     //读看门狗的控制寄存器
  wtcon &= ~S3C2410_WTCON_PRESCALE_MASK;       //看门狗控制器高 8 位清零，也就是预分频部分清零
  wtcon |= S3C2410_WTCON_PRESCALE(divisor-1);  //填入预分频系数

  writel(count, wdt_base + S3C2410_WTDAT);     //填入计数值
  writel(wtcon, wdt_base + S3C2410_WTCON);     //填入控制寄存器的值

  return 0;
}
```

平台驱动的移除函数：

```
static int s3c2410wdt_remove(struct platform_device *dev)
{
  release_resource(wdt_mem);          //释放平台资源
  kfree(wdt_mem);                     //释放 I/O 内存
  wdt_mem = NULL;

  free_irq(wdt_irq->start, dev);      //释放中断号
  wdt_irq = NULL;

  clk_disable(wdt_clock);             //关闭时钟
  clk_put(wdt_clock);                 //减少时钟引用计数
  wdt_clock = NULL;

  iounmap(wdt_base);                  //关闭内存映射
  misc_deregister(&s3c2410wdt_miscdev);  //注销混杂设备
```

```
    return 0;
}
```

平台驱动中的关闭函数：

```
static void s3c2410wdt_shutdown(struct platform_device *dev)
{
    s3c2410wdt_stop();
}
```

平台驱动中的电源管理部分：

```
#ifdef CONFIG_PM

static unsigned long wtcon_save;
static unsigned long wtdat_save;

static int s3c2410wdt_suspend(struct platform_device *dev, pm_message_t state)
{
    /*只要保存 S3C2410_WTDAT 就可以了，不需要保存 S3C2410_WTCNT，S3C2410_WTCNT 的值会在系统还原的时候直接赋为 S3C2410_WTDAT 中的值*/
    wtcon_save = readl(wdt_base + S3C2410_WTCON);
    wtdat_save = readl(wdt_base + S3C2410_WTDAT);

    /* Note that WTCNT doesn't need to be saved. */
    s3c2410wdt_stop();

    return 0;
}

static int s3c2410wdt_resume(struct platform_device *dev)
{
    /* Restore watchdog state. */

    writel(wtdat_save, wdt_base + S3C2410_WTDAT);
    writel(wtdat_save, wdt_base + S3C2410_WTCNT); /* Reset count */
    writel(wtcon_save, wdt_base + S3C2410_WTCON);

    printk(KERN_INFO PFX "watchdog %sabled/n",
          (wtcon_save & S3C2410_WTCON_ENABLE) ? "en" : "dis");

    return 0;
}

#else
#define s3c2410wdt_suspend NULL
#define s3c2410wdt_resume  NULL
#endif /* CONFIG_PM */
```

来看看看门狗驱动作为字符驱动的 file_operations 结构：

```
static const struct file_operations s3c2410wdt_fops = {
```

```
    .owner      = THIS_MODULE,
    .llseek     = no_llseek,
    .write      = s3c2410wdt_write,
    .unlocked_ioctl = s3c2410wdt_ioctl,
    .open       = s3c2410wdt_open,
    .release    = s3c2410wdt_release,
};
```

看门狗作为字符设备的打开函数：

```
static int s3c2410wdt_open(struct inode *inode, struct file *file)
{
    /*检查 open_lock 的第 0 位，如果 open_lock 的第 0 位为 0，则表示 test_and_set_bit 的返回值为 0，
        表示 wdt 设备没有被其他进程打开，如果为 1，表示被其他进程打开，返回 EBUSY*/
    if (test_and_set_bit(0, &open_lock))
        return -EBUSY;
    /*如果决不允许关闭看门狗，增加引用计数*/
    if (nowayout)
        __module_get(THIS_MODULE);
    /*设为不允许关闭*/
    allow_close = CLOSE_STATE_NOT;

    /*开启看门狗*/
    s3c2410wdt_start();
    /*这些寄存器不需要像文件一样对位置进行寻址*/
    return nonseekable_open(inode, file);
}
```

下面看一下开启看门狗的函数：

```
static void s3c2410wdt_start(void)
{
    unsigned long wtcon;
    spin_lock(&wdt_lock);
    /*首先停止看门狗*/
    __s3c2410wdt_stop();
    /*读看门狗的控制寄存器*/
    wtcon = readl(wdt_base + S3C2410_WTCON);
    /*使能看门狗，设置时钟分频值为 128*/
    wtcon |= S3C2410_WTCON_ENABLE | S3C2410_WTCON_DIV128;
    /*上边说过看门狗一个作为一个正常的 16 位内部定时器也可作为复位硬件*/
    if (soft_noboot) {                  //如果作为内部定时器
        wtcon |= S3C2410_WTCON_INTEN;   //开中断使能
        wtcon &= ~S3C2410_WTCON_RSTEN;  //关闭复位使能
    } else {                            //如果作为复位硬件
        wtcon &= ~S3C2410_WTCON_INTEN;  //关闭中断使能
        wtcon |= S3C2410_WTCON_RSTEN;   //开复位使能
    }
```

```
19.
20.    DBG("%s: wdt_count=0x%08x, wtcon=%08lx/n",
21.        __func__, wdt_count, wtcon);
22.    /*这里为什么要设置 WTCNT 的值，不是 WTDAT 中的值自动载入 WTCNT 吗？看看 datasheet 就知道了，看门狗定时器最初使能的时候，WTDAT 寄存器的值不会自动载入时　　间计数器，所以 WTCNT 寄存器必须在使能它之前设置一个初始值*/
23.    writel(wdt_count, wdt_base + S3C2410_WTDAT);   //设置计数值
24.    writel(wdt_count, wdt_base + S3C2410_WTCNT);   //设置计数值
25.    writel(wtcon, wdt_base + S3C2410_WTCON);       //设置 WTCON，刚才的使能为开启了看门狗
26.    spin_unlock(&wdt_lock);
27.}
```

关闭函数：

```
1.  static int s3c2410wdt_release(struct inode *inode, struct file *file)
2.  {
3.    /*
4.     *  Shut off the timer.
5.     *  Lock it in if it's a module and we set nowayout
6.     */
7.
8.    if (allow_close == CLOSE_STATE_ALLOW)     //看门狗为允许关闭状态
9.       s3c2410wdt_stop();                 //关闭看门狗
10.   else {
11.      dev_err(wdt_dev, "Unexpected close, not stopping watchdog/n");
12.      s3c2410wdt_keepalive();          //否则喂狗
13.   }
14.   allow_close = CLOSE_STATE_NOT;          //设为不允许关闭
15.   clear_bit(0, &open_lock);               //将 open_lock 的第 0 位设为 0，是原子操作
16.   return 0;
17.}
```

下边是关闭函数中调用的看门狗停止函数，可以看出实际关闭看门狗的操作是由__s3c2410wdt_stop 函数完成的。

```
1.  static void s3c2410wdt_stop(void)
2.  {
3.    spin_lock(&wdt_lock);
4.    __s3c2410wdt_stop();
5.    spin_unlock(&wdt_lock);
6.  }
7.
8.  static void __s3c2410wdt_stop(void)
9.  {
10.   unsigned long wtcon;
11.   wtcon = readl(wdt_base + S3C2410_WTCON);
12.   wtcon &= ~(S3C2410_WTCON_ENABLE | S3C2410_WTCON_RSTEN);
13.   writel(wtcon, wdt_base + S3C2410_WTCON);
14.}
```

写函数：

```
1. static ssize_t s3c2410wdt_write(struct file *file, const char __user *data,
2.                 size_t len, loff_t *ppos)
3. {
4.     /*
5.      *  Refresh the timer.
6.      */
7.     if (len) {                          //有数据写入 len 不为 0
8.         if (!nowayout) {                //允许关闭
9.             size_t i;
10.
11.            /* In case it was set long ago */
12.            allow_close = CLOSE_STATE_NOT;   //关闭允许关闭状态
13.
14.            for (i = 0; i != len; i++) {
15.                char c;
16.
17.                if (get_user(c, data + i))
18.                    return -EFAULT;
19.                if (c == 'V')           //如果向设备写入'V'，就允许关闭设备
20.                    allow_close = CLOSE_STATE_ALLOW;
21.            }
22.        }
23.        s3c2410wdt_keepalive();
24.    }
25.    return len;
26.}
```

io 控制函数：

```
1. static long s3c2410wdt_ioctl(struct file *file, unsigned int cmd,
2.                              unsigned long arg)
3. {
4.     void __user *argp = (void __user *)arg;
5.     int __user *p = argp;
6.     int new_margin;
7.
8.     switch (cmd) {
9.     case WDIOC_GETSUPPORT:                          //获得看门狗设备的信息
10.        return copy_to_user(argp, &s3c2410_wdt_ident,
11.            sizeof(s3c2410_wdt_ident)) ? -EFAULT : 0;
12.    case WDIOC_GETSTATUS:                           //获得门狗设备的信息
13.    case WDIOC_GETBOOTSTATUS:
14.        return put_user(0, p);
15.    case WDIOC_KEEPALIVE:                           //对看门狗进行喂狗
16.        s3c2410wdt_keepalive();
```

```
        return 0;
    case WDIOC_SETTIMEOUT:                                  //设置看门狗的超时时间
        if (get_user(new_margin, p))
            return -EFAULT;
        if (s3c2410wdt_set_heartbeat(new_margin))
            return -EINVAL;
        s3c2410wdt_keepalive();
        return put_user(tmr_margin, p);
    case WDIOC_GETTIMEOUT:                                  //获得门狗设备的信息
        return put_user(tmr_margin, p);
    default:
        return -ENOTTY;
    }
}
```

如果选择了看门狗作为内部定时器，则当计数值为 0 时调用中断处理函数，中断处理函数的主要功能就是喂狗：

```
static irqreturn_t s3c2410wdt_irq(int irqno, void *param)
{
    dev_info(wdt_dev, "watchdog timer expired (irq)/n");
    /*中断处理中调用喂狗*/
    s3c2410wdt_keepalive();
    return IRQ_HANDLED;
}
```

喂狗函数：

```
static void s3c2410wdt_keepalive(void)
{
    spin_lock(&wdt_lock);
    writel(wdt_count, wdt_base + S3C2410_WTCNT);
    spin_unlock(&wdt_lock);
}
```

总结一下：看门狗设备可以在系统启动的时候开启也可以在打开设备的时候启动，看门狗设备不是用来读取或者写入神马数据的，只可写入一个'V'用来允许关闭看门狗设备，打开看门狗设备后主要就是通过 ioctl 来发各种命令，进行获得门狗设备的信息，门狗设备的信息，门狗设备的信息，设置看门狗的超时时间，对看门狗进行喂狗等操作在内核和用户空间之间传递信息。