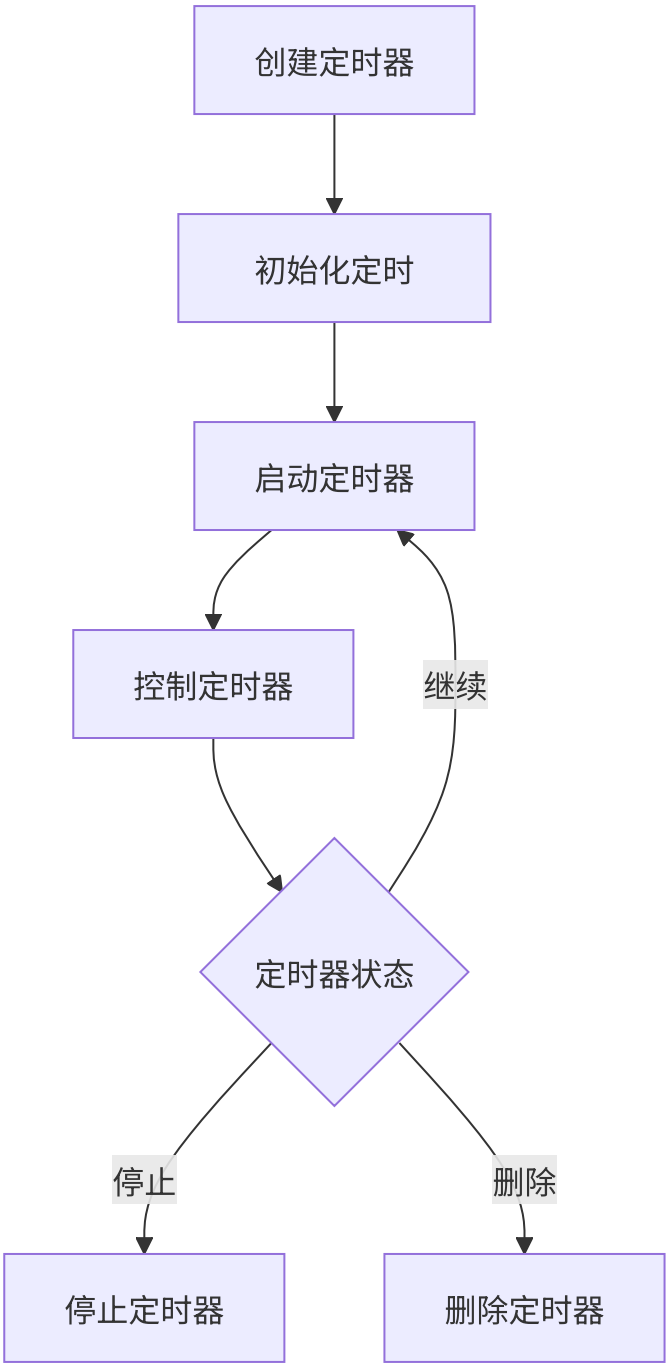
RT定时器介绍

RT的定时器分为HARD_TIMER模式和SOFT_TIMER模式

- 其中HARD_TIMER模式的超时函数在中断上下文环境中执行并且RT定时器默认采用HARD_TIMER模式。
- SOFT_TIMER模式，会创建一个timer线程，定时器超时函数在线程的上下文执行。
- 可在初始化/创建定时器时使用参数来设置定时器模式。

定时器使用流程图



管理方法

1. 创建和新建定时器:
   函数原型:

```
 rt_timer_t rt_timer_create(const char *name,
                          void (*timeout)(void *parameter),
                          void       *parameter,
                          rt_tick_t   time,
                          rt_uint8_t  flag)
{
 struct rt_timer *timer;

 /* allocate a object */
 timer = (struct rt_timer *)rt_object_allocate(RT_Object_Class_Timer,
name);
 if (timer == RT_NULL)
 {
     return RT_NULL;
 }

 _rt_timer_init(timer, timeout, parameter, time, flag);

 return timer;
}
RTM_EXPORT(rt_timer_create);
```

参数	描述
name	定时器名称
void(timeout)(void parameter)	定时器超时函数指针
parameter	定时器超时函数的入口参数
time	定时器的超时时间
flag	定时器创建时的参数，支持的值包括(单次定时、周期定时、硬件定时器、软件定时器等)并且可以用"或"关系取多个值
返回	描述
------	------
RT_NULL	创建失败
定时器的句柄	定时器创建成功
注:不再使用该定时器时,可以利用rt_err_t rt_timer_delete(rt_timer_t timer);	
2.启动和停止定时器	
启动:rt_err_t rt_timer_start(rt_timer_t timer)	
停止:rt_err_t rt_timer_stop(rt_timer_t timer)	

参数	描述
注:timer是定时器的句柄	

创建动态定时器例程:

```c
#include <rtthread.h>

#define DBG_TAG "main"
#define DBG_LVL DBG_LOG
#include <rtdbg.h>

int main(void)
{

    return RT_EOK;
}

static rt_timer_t timer1;
static rt_timer_t timer2;
static int cnt = 0;
/* 定时器1超时函数 */
static void timeout1(void *parameter)
{
    rt_kprintf("periodic timer is timeout %d\n",cnt);

    /* 运行第十次时,停止周期性定时器 */
    if (cnt++ >= 9)
    {
        rt_timer_stop(timer1);
        rt_kprintf("periodic timer was stopped! \n");
    }
}
/*定时器2超时函数*/
static void timeout2(void *parameter)
{
    rt_kprintf("one shot timer is timeout\n");

}

int timer_sample(void)
{
    timer1 = rt_timer_create("timer1",timeout1,RT_NULL, 10,
RT_TIMER_FLAG_PERIODIC);//创建定时器1
    if(timer1 != RT_NULL) rt_timer_start(timer1);启动定时器1

    timer2 = rt_timer_create("timer2",timeout2,RT_NULL,
30,RT_TIMER_FLAG_ONE_SHOT);创建定时器2

    if(timer2!=RT_NULL) rt_timer_start(timer2);
    return 0;启动定时器2
}

MSH_CMD_EXPORT(timer_sample, timer sample);//将命令移植到终端
```

##现象:

```
msh >timer_sample
msh >periodic timer is timeout 0
periodic timer is timeout 1
one shot timer is timeout
periodic timer is timeout 2
periodic timer is timeout 3
periodic timer is timeout 4
periodic timer is timeout 5
periodic timer is timeout 6
periodic timer is timeout 7
periodic timer is timeout 8
periodic timer is timeout 9
periodic timer was stopped!
```

3.初始化和脱离定时器

选择静态创建定时器时,可利用rt_timer_init接口来初始化该定时器,函数接口如下:

```
void rt_timer_init(rt_timer_t  timer,
                   const char *name,
                   void (timeout)(void *parameter),
                   void       *parameter,
                   rt_tick_t   time,
                   rt_uint8_t  flag)
```

参数	描述
timer	定时器句柄
name	定时器名称
void (timeout) (void parameter)	定时器超时函数指针
parameter	定时器超时函数的入口参数
time	超时时间，单位是时钟节拍
flag	定时器创建时的参数，支持的值包括(单次定时、周期定时、硬件定时器、软件定时器等)并且可以用"或"关系取多个值
返回值	描述
RT_NULL	创建失败
定时器的句柄	定时器创建成功

创建静态定时器的例程

```c
#include <rtthread.h>

#define DBG_TAG "main"
#define DBG_LVL DBG_LOG
#include <rtdbg.h>

int main(void)
{

    return RT_EOK;
}

static struct rt_timer timer1;
static struct rt_timer timer2;
static int cnt =0;

static void timeout1(void* parameter)
{
    rt_kprintf("periodic timer is timeout\n");
    if(cnt++ >= 9 )
    {
        rt_timer_stop(&timer1);
    }

}

static void timeout2(void* parameter)
{
    rt_kprintf("one shot timer is timeout\n");
}

int timer_static_sample(void)
{

 rt_timer_init(&timer1,"timer1",timeout1,RT_NULL,10,RT_TIMER_FLAG_PERIODIC);

 rt_timer_init(&timer2,"timer2",timeout2,RT_NULL,30,RT_TIMER_FLAG_ONE_SHOT);

    rt_timer_start(&timer1);
    rt_timer_start(&timer2);
    return 0;
}

MSH_CMD_EXPORT(timer_static_sample, timer_static sample)
```

现象:

```
msh >timer_static_sample
msh >periodic timer is timeout
periodic timer is timeout
one shot timer is timeout
periodic timer is timeout
periodic timer is timeout
periodic timer is timeout
periodic timer is timeout
periodic timer is timeout
periodic timer is timeout
periodic timer is timeout
periodic timer is timeout
```