APRUS Lua 计时器和定时器配置说明

概述

本说明书将介绍定时器和计时器在 aprus Lua 程序中的使用。定时器和计时器是两种不同的用于控制程序执行和时间管理的工具，它们在各种应用程序中都扮演着重要的角色。

1 APRUS.lua 配置说明

```
package.cpath="./?.so"
package.path="./?.lua"

function AlarmCallBack()
    ClockSecond = ClockSecond + 1              --lua 运行时长自加 1
    print("[Alarm]: ClockSecond=" .. ClockSecond)        --打印运行时长
    print("[Alarm]: ClockTimerCounter=" .. user.getClockTimer(myTimer))        --打印计时器的计数值
end

function delay_NmS(Nmsec)                  --一种阻塞 lua 流程的定时方法。用于控制上的延时。（非必要不要使用这种方法。）
        user.resetClockTimer(myTimer)              --复位计时器
        while(user.getClockTimer(myTimer) < Nmsec) do     --读取计时器当前值，并判断是否满足条件
            os.execute("usleep 1000")
        end
end

function start()
    myTimer = user.newClockTimer()             -- 创建计时器 myTimer（未复位前，计时器的值会一直增加下去）， 用于延时函数 delay_NmS()
    ClockSecond = 0

    user.addtimer("1sec",-1,1);                --设置定时器(最多 10 个)： "1sec":定时器的名称为"1sec"（最长 19 字符） -1: 无限定时次数；  1: 定时周期 1 秒一次；  （最多可以添加 10 个定时器）

    print("ClockTimer Test")
    delay_NmS(500)          --延时 0.5 秒
    print("block delay 5.5S")
    delay_NmS(5500)         --延时 5.5 秒
    print("ClockTimer TimeOut")
    delay_NmS(500)

    while true do
```

```
        msg = user.waitmsg()

        if msg.from == "timer" then
            if msg.name == "1sec" then
                AlarmCallBack()
            end
        end
    end
end

start()
```

1.1 此处仅解释一下计时器和定时器相关的函数和操作：

1、计时器：

（1）创建计时器 myTimer，未复位前，计时器的值会一直增加下去。

```
myTimer = user.newClockTimer()
```

（2）示例应用

例如：计时器可用于延时函数 delay_NmS()

```
function delay_NmS(Nmsec)
        user.resetClockTimer(myTimer)
        while(user.getClockTimer(myTimer) < Nmsec) do
            os.execute("usleep 1000")
        end
end
```

delay_NmS 函数是一种阻塞 lua 流程的定时方法。用于控制上的延时。（非必要情况下不要使用这种方法）

首先调用 user.resetClockTimer(myTimer)复位 myTimer 计时器，然后调用 user.getClockTimer(myTimer)获取 myTimer 计时器当前值，并判断是否满足条件，若满足条件则执行 os.execute("usleep 1000")。

2、定时器：

（1）使用以下命令设置一个名为"1sec"的定时器：

user.addtimer("1sec",-1,1)，此定时器会在每秒钟触发一次。

（2）主循环中的消息处理

为了确保定时器触发时能够正确执行相关操作，需要在 Lua 的主循环中监听消息，并对其进行处理。示例如下：

```
while true do
    local msg = user.waitmsg()   -- 等待消息到来
    if msg.from == "timer" then   -- 检查消息是否来自定时器
        if msg.name == "1sec" then   -- 检查是否是"1sec"定时器触发的消息
            AlarmCallBack()   -- 调用闹钟的回调函数
        end
    end
```

```
end
```

(3)回调函数 AlarmCallBack

当定时器触发时，会调用名为 AlarmCallBack 的函数。此函数的原型如下：

```
function AlarmCallBack()
    ClockSecond = ClockSecond + 1   -- Lua 运行时长自增 1 秒
    print("[Alarm]: ClockSecond=" .. ClockSecond)   -- 打印当前的运行时长
end
```

每次进入 AlarmCallBack 函数时，都会通过累加 ClockSecond 变量的方式来实现对当前 Lua 运行时间的校准，并使用 print 函数打印其值。需要注意的是确保全局变量 `ClockSecond` 已经被初始化（例如，设置为 0）。

1.2 计时器和定时器 函数 API

1.1.1. user.addtimer

功能	设置定时器(一个 Lua 程序中最多可存在 10 个定时器)
接口描述	user.addtimer("timerName", Count, Cycle)
"timerName"	定时器的名称为"timerName"（最长 19 字符）
Count	总共需要执行的定时器次数。如果 Count 为-1，则表示定时器将无限次执行。
Cycle	定时器执行的间隔时间，以秒为单位。每隔 Cycle 秒，定时器就会执行一次回调函数。

1.1.2. user.closetimer

功能	停止定时器
接口描述	user.closetimer("timerName")
"timerName"	定时器的名称为 timerName

1.1.3. user.newClockTimer

功能	user.newTimer 返回一个计时器实例对象；未复位前，计时器的值会从当前时间开始一直增加下去；一个 LUA 程序可以有多个计时器；可用于延时函数
接口描述	myTimer = user.newClockTimer()
myTimer	user.newClockTimer 返回值；计时器实例对象

1.1.4. user.resetClockTimer

功能	复位计时器
接口描述	user.resetClockTimer(myTimer)
myTimer	计时器实例对象；

1.1.5. user.getClockTimer

功能	获取计时器当前计时值
接口描述	num = user.getClockTimer(myTimer)

myTimer	计时器实例对象；
num	计时器当前计时值，单位 ms

2 config.lua 配置说明

计时器和定时器无需配置 config.lua