NV32F100x 中断处理机制

本文以 RTC_demo 为例讲解一下 NV32F100 中断服务流程

首先，看一下 NV32F100x 的中断向量表。RTC 中断在向量表里的中断号为 36

向量	IRQ	NVIC IPR 编号	中断源	说明	地址
ARM 内核系统处理程序向量					
0	–	–	ARM 内核	初始堆栈指针	0x0000_0000
1	–	–	ARM 内核	初始程序计数	0x0000_0004
2	–	–	ARM 内核	不可屏蔽中断(NMI)	0x0000_0008
3	–	–	ARM 内核	硬故障	0x0000_000C
4	–	–	–	–	0x0000_0010
5	–	–	–	–	0x0000_0014
6	–	–	–	–	0x0000_0018
7	–	–	–	–	0x0000_001C
8	–	–	–	–	0x0000_0020
9	–	–	–	–	0x0000_0024
10	–	–	–	–	0x0000_0028
11	–	–	ARM 内核	管理程序调用(SVCall)	0x0000_002C
12	–	–	–	–	0x0000_0030
13	–	–	–	–	0x0000_0034
14	–	–	ARM 内核	可挂起的系统服务请求	0x0000_0038
15	–	–	ARM 内核	系统节拍定时器(SysTick)	0x0000_003C
非内核向量					
16	0	0	–	–	0x0000_0040
17	1	0	–	–	0x0000_0044
18	2	0	–	–	0x0000_0048
19	3	0	–	–	0x0000_004C
20	4	1	–	–	0x0000_0050
21	5	1	–	–	0x0000_0054
22	6	1	PMC	低电压警告	0x0000_0058
23	7	1	IRQ	外部中断	0x0000_005C
24	8	2	I2C	所有源的单个中断向量	0x0000_0060
25	9	2	–	–	0x0000_0064
26	10	2	SPI0	所有源的单个中断向量	0x0000_0068
27	11	2	SPI1	所有源的单个中断向量	0x0000_006C
28	12	3	UART0	状态和错误	0x0000_0070
29	13	3	UART1	状态和错误	0x0000_0074
30	14	3	UART2	状态和错误	0x0000_0078
31	15	3	ADC0	ADC 转换完成中断	0x0000_007C

32	16	4	ACMP0	模拟比较器 0 中断	0x0000_0080
33	17	4	ETimer0	所有源的单个中断向量	0x0000_0084
34	18	4	ETimer1	所有源的单个中断向量	0x0000_0088
35	19	4	ETimer2	所有源的单个中断向量	0x0000_008C
36	20	5	RTC	RTC 溢出中断	0x0000_0090
37	21	5	ACMP1	模拟比较器 1 中断	0x0000_0094
38	22	5	PIT_CH0	PIT CH0 溢出中断	0x0000_0098
39	23	5	PIT_CH1	PIT CH1 溢出中断	0x0000_009C
40	24	6	KBI0(32bit)	键盘中断 0（32 位）	0x0000_00A0
41	25	6	KBI1(32bit)	键盘中断 1（32 位）	0x0000_00A4
42	26	6	-	-	0x0000_00A8
43	27	6	ICS	时钟失锁	0x0000_00AC
44	28	7	WDOG	看门狗超时中断	0x0000_00B0
45	29	7	-	-	0x0000_00B4
46	30	7	-	-	0x0000_00B8
47	31	7	-	-	0x0000_00BC

在我们提供的服务包里，为大家提供了中断服务的框架，采用中断回调的机制，方便用户调用。

在 Vector.c 下的 isr.h 中注册中断服务函数

```
#undef  VECTOR_036
#define VECTOR_036      RTC_Isr     /*!< Vector 36 points to RTC interrupt service routine */

extern void RTC_Isr(void);
```

编写中断服务函数 RTC_Isr，用户可以直接在函数中编写中断处理程序，而我们采用回调的机制，方便在原有子工程中直接操作。

```
void RTC_Isr(void)
{
    RTC_ClrFlags();
    if (RTC_Callback[0])
    {
        RTC_Callback[0]();
    }
}
```

在中断服务函数中进行操作，清除中断标志位等。

在 RTC.h 中定义了回调类型为函数指针

```
typedef void (*RTC_CallbackType)(void);
```

在 RTC.c 中，定义了回调的数组

```
RTC_CallbackType RTC_Callback[1] = {(RTC_CallbackType)NULL};    /*!< RTC initial callback */
```

为了方便用户调用，通过如下函数实现中断任务函数入口的设置

```
void RTC_SetCallback(RTC_CallbackType pfnCallback)
{
  RTC_Callback[0] = pfnCallback;
}
```

在 RTC_demo.c 中调用，设置入口函数地址，RTC_Task 即为中断任务函数

```
RTC_SetCallback(RTC_Task);
```

在了解我们中断处理机制后，还需注意的是，在开启中断的时候，一定要注册中断号，使能中断位，在进入中断服务函数的时候，一定要清中断标志位，否则会一直进入默认中断。

7 RTIF	实时中断标志 （进入中断，实时清除） 该状态位指示 RTC 计数器寄存器已达到 RTC 模数寄存器中的值。写入逻辑 0 无效。写入逻辑 1 会将该位清零并清除实时中断请求。复位会将 RTIF 清除为 0。 0 RTC 计数器未达到 RTC 模数寄存器中的值。 1 RTC 计数器已达到 RTC 模数寄存器中的值。
6 RTIE	实时中断使能（开启中断，中断使能） 该读/写位使能实时中断。如果 RTIE 置位，那么在 RTIF 置位时会生成中断。复位会将 RTIE 清除为 0。 0 实时中断请求禁用。使用软件轮询。 1 实时中断请求使能。