Week4-2 中断

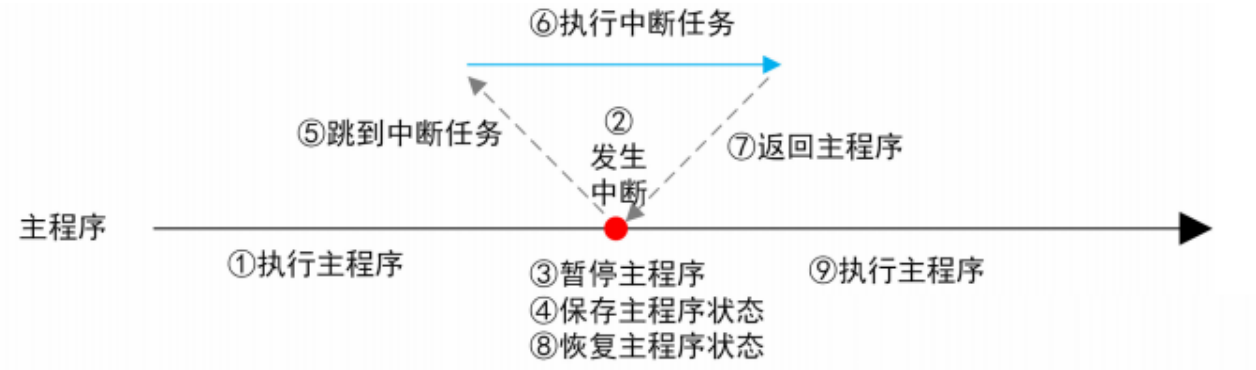
一、中断的基本概念

1.1什么是中断

处理器中的中断：

在处理器中，中断是一个过程，即**CPU在正常执行程序的过程中，遇到外部/内部的紧急事件需要处理，暂时中止当前程序的执行，转而去为处理紧急的事件，待处理完毕后再返回被打断的程序处继续往下执行。**中断在计算机多任务处理，尤其是即时系统中尤为重要。

一张经典的中断图示（按着①-⑨的顺序耐心看一遍！描述一遍过程！）



意义：

中断能提高CPU的效率，同时能对突发事件做出实时处理。实现程序的并行化，实现嵌入式系统进程之间的切换。

1.2NVIC

STM32有84个中断，包括16个内核中断和68个可屏蔽中断，具有16级可编程的中断优先级。

我们可以看一下Document-*STM32中文参考手册.pdf* p132-134

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000_0000
	-3	固定	Reset	复位	0x0000_0004
	-2	固定	NMI	不可屏蔽中断 RCC时钟安全系统(CSS)联接到NMI向量	0x0000_0008
	-1	固定	硬件失效(HardFault)	所有类型的失效	0x0000_000C
	0	可设置	存储管理(MemManage)	存储器管理	0x0000_0010
	1	可设置	总线错误(BusFault)	预取指失败，存储器访问失败	0x0000_0014
	2	可设置	错误应用(UsageFault)	未定义的指令或非法状态	0x0000_0018
	-	-	-	保留	0x0000_001C ~0x0000_002B
	3	可设置	SVCall	通过SWI指令的系统服务调用	0x0000_002C
	4	可设置	调试监控(DebugMonitor)	调试监控器	0x0000_0030
	-	-	-	保留	0x0000_0034
	5	可设置	PendSV	可挂起的系统服务	0x0000_0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000_003C

0	7	可设置	WWDG	窗口定时器中断	0x0000_0040
1	8	可设置	PVD	连到EXTI的电源电压检测(PVD)中断	0x0000_0044
2	9	可设置	TAMPER	侵入检测中断	0x0000_0048
3	10	可设置	RTC	实时时钟(RTC)全局中断	0x0000_004C
4	11	可设置	FLASH	闪存全局中断	0x0000_0050
5	12	可设置	RCC	复位和时钟控制(RCC)中断	0x0000_0054
6	13	可设置	EXTI0	EXTI线0中断	0x0000_0058
7	14	可设置	EXTI1	EXTI线1中断	0x0000_005C
8	15	可设置	EXTI2	EXTI线2中断	0x0000_0060
9	16	可设置	EXTI3	EXTI线3中断	0x0000_0064
10	17	可设置	EXTI4	EXTI线4中断	0x0000_0068
11	18	可设置	DMA1通道1	DMA1通道1全局中断	0x0000_006C
12	19	可设置	DMA1通道2	DMA1通道2全局中断	0x0000_0070
13	20	可设置	DMA1通道3	DMA1通道3全局中断	0x0000_0074
14	21	可设置	DMA1通道4	DMA1通道4全局中断	0x0000_0078
15	22	可设置	DMA1通道5	DMA1通道5全局中断	0x0000_007C
16	23	可设置	DMA1通道6	DMA1通道6全局中断	0x0000_0080
17	24	可设置	DMA1通道7	DMA1通道7全局中断	0x0000_0084
18	25	可设置	ADC1_2	ADC1和ADC2的全局中断	0x0000_0088
19	26	可设置	USB_HP_CAN_TX	USB高优先级或CAN发送中断	0x0000_008C
20	27	可设置	USB_LP_CAN_RX0	USB低优先级或CAN接收0中断	0x0000_0090
21	28	可设置	CAN_RX1	CAN接收1中断	0x0000_0094
21	28	可设置	CAN_RX1	CAN接收1中断	0x0000_0094
22	29	可设置	CAN_SCE	CAN SCE中断	0x0000_0098
23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
24	31	可设置	TIM1_BRK	TIM1刹车中断	0x0000_00A0
25	32	可设置	TIM1_UP	TIM1更新中断	0x0000_00A4
26	33	可设置	TIM1_TRG_COM	TIM1触发和通信中断	0x0000_00A8
27	34	可设置	TIM1_CC	TIM1捕获比较中断	0x0000_00AC
28	35	可设置	TIM2	TIM2全局中断	0x0000_00B0
29	36	可设置	TIM3	TIM3全局中断	0x0000_00B4

30	37	可设置	TIM4	TIM4全局中断	0x0000_00B8
31	38	可设置	I2C1_EV	I^2C1事件中断	0x0000_00BC
32	39	可设置	I2C1_ER	I^2C1错误中断	0x0000_00C0
33	40	可设置	I2C2_EV	I^2C2事件中断	0x0000_00C4
34	41	可设置	I2C2_ER	I^2C2错误中断	0x0000_00C8
35	42	可设置	SPI1	SPI1全局中断	0x0000_00CC
36	43	可设置	SPI2	SPI2全局中断	0x0000_00D0
37	44	可设置	USART1	USART1全局中断	0x0000_00D4
38	45	可设置	USART2	USART2全局中断	0x0000_00D8
39	46	可设置	USART3	USART3全局中断	0x0000_00DC
40	47	可设置	EXTI15_10	EXTI线[15:10]中断	0x0000_00E0
41	48	可设置	RTCAlarm	连到EXTI的RTC闹钟中断	0x0000_00E4
42	49	可设置	USB唤醒	连到EXTI的从USB待机唤醒中断	0x0000_00E8
43	50	可设置	TIM8_BRK	TIM8刹车中断	0x0000_00EC
44	51	可设置	TIM8_UP	TIM8更新中断	0x0000_00F0
45	52	可设置	TIM8_TRG_COM	TIM8触发和通信中断	0x0000_00F4
46	53	可设置	TIM8_CC	TIM8捕获比较中断	0x0000_00F8
47	54	可设置	ADC3	ADC3全局中断	0x0000_00FC
48	55	可设置	FSMC	FSMC全局中断	0x0000_0100
49	56	可设置	SDIO	SDIO全局中断	0x0000_0104
50	57	可设置	TIM5	TIM5全局中断	0x0000_0108
51	58	可设置	SPI3	SPI3全局中断	0x0000_010C
52	59	可设置	UART4	UART4全局中断	0x0000_0110
53	60	可设置	UART5	UART5全局中断	0x0000_0114
54	61	可设置	TIM6	TIM6全局中断	0x0000_0118
55	62	可设置	TIM7	TIM7全局中断	0x0000_011C
56	63	可设置	DMA2通道1	DMA2通道1全局中断	0x0000_0120
57	64	可设置	DMA2通道2	DMA2通道2全局中断	0x0000_0124
58	65	可设置	DMA2通道3	DMA2通道3全局中断	0x0000_0128
59	66	可设置	DMA2通道4_5	DMA2通道4和DMA2通道5全局中断	0x0000_012C

STM32具有十分强大的中断系统，将中断分为了两个类型：**内核异常和外部中断。**

-3到6这个区域被标黑了，这个区域就是内核异常。**内核异常不能够被打断，不能被设置优先级（也就是说优先级是凌驾于外部中断之上的）。**常见的内核异常有以下几种：复位(reset)，不可屏蔽中断(NMI)，硬错误(Hardfault)，其他的也可以在表上找到。

从第7个开始，后面所有的中断都是外部中断，包含**EXTI外部中断，TIM定时中断，ADC数模中断，USART串口中断，SPI通讯中断，IIC通讯中断，RTC实时时钟等多个外设中断。**

数十个中断！**那该如何管理它们呢?**

——**在Cortex-M3 内核有一个专门管理中断的外设 NVIC （ Nested Vectored Interrupt Controller，嵌套向量中断控制器）**，通过优先级控制中断的嵌套和调度。NVIC 是一个总的中断控制器，无论是来在内核的异常还是外设的外部中断，都由NVIC 统一进行管理。

1.3抢占优先级和响应优先级

我们先来介绍两个概念：**抢占优先级和响应优先级，每个中断源都需要被指定这两种优先级。**

首先先来认识一下它们~

- **什么是抢占优先级**?

  抢占优先级比较霸道，一言不和就插队。抢占优先级高的，能够打断优先级低的任务，等优先级较高的任务执行完毕后，再回来继续执行之前的任务。所以当存在多个抢占优先级不同的任务时，很有可能会产生任务的**嵌套。**

- **什么是响应优先级**?

  响应优先级则稍微谦逊些，比较有礼貌。响应优先级又被称为次优先级，若两个任务的抢占式优先级一样，那么响应优先级较高的任务则先执行，且在执行的同时**不能**被下一个响应优先级更高的任务打断，所以说它比较有有礼貌。

当两个中断源的抢占式优先级相同时，这两个中断将没有嵌套关系，当一个中断到来后，如果正在处理另一个中断，这个后到来的中断就要等到前一个中断处理完之后才能被处理。

如果这两个中断同时到达，则中断控制器根据他们的响应优先级高低来决定先处理哪一个；

如果他们的抢占式优先级和响应优先级都相等，则根据他们在中断表中的排位顺序决定先处理哪一个。

1.4优先级分组

既然每个中断源都需要被指定这两种优先级，就需要有相应的寄存器位记录每个中断的优先级；在Cortex-M3中**定义了8个比特位用于设置中断源的优先级**，这8个比特位可以有8种分配方式，这就是**优先级分组**的概念。分配方式如下：

- 所有8位用于指定响应优先级
- 最高1位用于指定抢占式优先级，最低7位用于指定响应优先级
- 最高2位用于指定抢占式优先级，最低6位用于指定响应优先级
- 最高3位用于指定抢占式优先级，最低5位用于指定响应优先级
- 最高4位用于指定抢占式优先级，最低4位用于指定响应优先级
- 最高5位用于指定抢占式优先级，最低3位用于指定响应优先级
- 最高6位用于指定抢占式优先级，最低2位用于指定响应优先级
- 最高7位用于指定抢占式优先级，最低1位用于指定响应优先级

Cortex-M3允许具有较少中断源时使用较少的寄存器位指定中断源的优先级，因此STM32把指定中断优先级的寄存器位**减少到4位**，这4个寄存器位的分组方式如下：

- 第0组：所有4位用于指定响应优先级
- 第1组：最高1位用于指定抢占式优先级，最低3位用于指定响应优先级
- 第2组：最高2位用于指定抢占式优先级，最低2位用于指定响应优先级
- 第3组：最高3位用于指定抢占式优先级，最低1位用于指定响应优先级
- 第4组：所有4位用于指定抢占式优先级

1.5指定中断源的优先级

我们知道了STM32使用4个寄存器位对优先级进行分组，以下为代码实现：

可以通过调用STM32的固件库中的函数**NVIC_PriorityGroupConfig()**选择使用哪种优先级分组方式，这个函数的参数有下列5种，代表不同的分配方式：

- NVIC_PriorityGroup_0 => 选择第0组
- NVIC_PriorityGroup_1 => 选择第1组
- NVIC_PriorityGroup_2 => 选择第2组
- NVIC_PriorityGroup_3 => 选择第3组
- NVIC_PriorityGroup_4 => 选择第4组

详细代码是：

```
1 /**
2 * 配置中断优先级分组：抢占优先级和响应优先级
3 * 形参如下：
4 * @arg NVIC_PriorityGroup_0: 0bit for 抢占优先级
5 *                          4 bits for 响应优先级
6 * @arg NVIC_PriorityGroup_1: 1 bit for 抢占优先级
7 *                          3 bits for 响应优先级
8 * @arg NVIC_PriorityGroup_2: 2 bit for 抢占优先级
9 *                          2 bits for 响应优先级
10 * @arg NVIC_PriorityGroup_3: 3 bit for 抢占优先级
11 *                           1 bits for 响应优先级
12 * @arg NVIC_PriorityGroup_4: 4 bit for 抢占优先级
13 *                           0 bits for 响应优先级
14 * @注意 如果优先级分组为 0，则抢占优先级就不存在，优先级就全部由响应优先级控制
15 */
16 void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)
17 {
18 // 设置优先级分组
19 SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
20 }
```

1.6再次理解！

下面是一个牛人的理解，请好好看哦！很形象！

首先这俩单词先混混脸熟.....**抢占优先级（Preempt Priority）和子优先级（Subpriority）**

STM32有43个channel的settable的中断源；AIRC(Application Interrupt and Reset Register)寄存器中有用于指定优先级的4 bits。**这4个bits用于分配preemption优先级和sub优先级**，在STM32的固件库中定义如下

```
/* Preemption Priority Group ------------------------------------------------*/
#define NVIC_PriorityGroup_0         ((u32)0x700) /* 0 bits for pre-emption priority
                                     4 bits for subpriority */
#define NVIC_PriorityGroup_1         ((u32)0x600) /* 1 bits for pre-emption priority
                                     3 bits for subpriority */
#define NVIC_PriorityGroup_2         ((u32)0x500) /* 2 bits for pre-emption priority
                                     2 bits for subpriority */
#define NVIC_PriorityGroup_3         ((u32)0x400) /* 3 bits for pre-emption priority
                                     1 bits for subpriority */
#define NVIC_PriorityGroup_4         ((u32)0x300) /* 4 bits for pre-emption priority
                                     0 bits for subpriority */
```

形象化的理解是：

你是上帝，造了43个人，这么多人要分**社会阶级**和**社会阶层**了；

因为“阶级”的词性比较重；"阶层"比较中性，所以

preemption优先级->阶级；

每个阶级内部，有一些阶层，sub优先级->阶层；

如果按照NVIC_PriorityGroup_4这么分，就分为了16个阶级（1个阶层就是1个preemption优先级），0个阶层；高阶级的人，可以打断低阶级的正在做事的人（嵌套），最多可以完成1个中断和15级嵌套。

每个阶级（每个preemption优先级），你来指定这43人中，谁进入该阶级；

一个人叫**EXTI0_IRQChannel**，你指定他进入“阶级8”，则

```
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 8; // 指定抢占式优先级别，可取0-15
```

另外，在同一阶级内部，一个人在做事的时候，另外一个人不能打断他；（preemption优先级别相同的中断源之间没有嵌套关系）
还有，如果他们两个同时想做事，因为没有阶层，那么就根据Vector table中的物理排序，让排名靠前的人去做；

又有一个人**SPI1_IRQChannel**，设定如下

```
NVIC_InitStructure.NVIC_IRQChannel = SPI1_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // 指定抢占式优先级别，可取0-15
```

SPI1_IRQChannel的阶级高，EXTI0_IRQChannel做事的时候可以打断（嵌套）。

如果按照NVIC_PriorityGroup_3这么分，就分为了8个阶级（1个阶级是1个preemption优先级），每个阶级内有2个阶层（sub优先级）；高阶级的人，可以打断低阶级的正在做事的人（嵌套），最多可以完成1个中断和7级嵌套。

每个阶级（每个preemption优先级），你来指定这43人中，谁进入该阶级；

一个人叫**EXTI0_IRQChannel**，你指定他进入“阶级3”，则：

```
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3; // 指定抢占式优先级别，可取0-7
```

还需要指定他的阶层：

```
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // 指定响应优先级别，可取0-1
```

另有一个人叫**EXTI9_5_IRQChannel**，他的阶级和阶层设定如下

```
NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3; // 指定抢占式优先级别，可取0-7
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; // 指定响应优先级别
```

那么这两个人是同一阶级的兄弟，一个人在做事的时候，另外一个人不能打断他；（preemption优先级别相同的中断源之间没有嵌套关系）如果他们两个同时想做事，因为前者的阶层高，所以前者优先。

还有一个人叫**USART1_IRQChannel**，他的阶级和阶层设定如下

```
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2; // 指定抢占式优先级别，可取0-7
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; // 指定响应优先级别
```

USART1_IRQChannel的优先级最高，当前面两个人做事的时候，他都可以打断（嵌套）。

其他的类推。

1.7Quiz time!

假定设置中断优先级组为2，然后设置中断3(RTC中断)的抢占优先级为2，响应优先级为1。 中断6（外部中断0）的抢占优先级为3，响应优先级为0。中断7（外部中断1）的抢占优先级为2，响应优先级为0。

Q：那么这3个中断的优先级顺序为?

A：中断7>中断3>中断6。

特别说明：

一般情况下，系统代码执行过程中，**只设置一次中断优先级分组**，比如分组2，设置好分组之后一般不会再改变分组。随意改变分组会导致中断管理混乱，程序出现意想不到的执行结果。

看懂了1.6的形象化理解，这个quiz是不是小菜一碟！如果还是有点晕，不妨再看一遍上面的概念哦，中断这里有点复杂，一定要理解好~

二、中断优先级设置

我们在第一部分介绍了什么是中断、管理中断的控制器NVIC、抢占优先级和响应优先级。对于优先级的分组设置好之后，**怎么设置单个中断的抢占优先级和响应优先级呢?**

我们先来看一下**中断设置相关的寄存器**都有哪些：

- __IO uint8_t IP[240]; //中断优先级控制的寄存器组

- __IO uint32_t ISER[8]; //中断使能寄存器组
- __IO uint32_t ICER[8]; //中断失能寄存器组
- __IO uint32_t ISPR[8]; //中断挂起寄存器组
- __IO uint32_t ICPR[8]; //中断解挂寄存器组
- __IO uint32_t IABR[8]; //中断激活标志位寄存器组

MDK中NVIC寄存器结构体

```
typedef struct
{
  __IO uint32_t ISER[8];
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;
}  NVIC_Type;
```

用法！举个例子~

```
// 选择使用优先级分组第1组
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
// 使能EXTI0中断
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; // 指定抢占式优先级别1
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // 指定响应优先级别0
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
// 使能EXTI9_5中断
NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQChannel;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // 指定抢占式优先级别0
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; // 指定响应优先级别1
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

上面代码块的最后一行使用了**中断参数初始化函数**， 函数为

```
void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
```

```
typedef struct
{
  uint8_t NVIC_IRQChannel; //设置中断通道
  uint8_t NVIC_IRQChannelPreemptionPriority;//设置响应优先级
  uint8_t NVIC_IRQChannelSubPriority; //设置抢占优先级
  FunctionalState NVIC_IRQChannelCmd; //使能/使能
} NVIC_InitTypeDef;
```

```
NVIC_InitTypeDef   NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;//串口1中断
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1 ;// 抢占优先级为1
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;// 子优先级位2
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;//IRQ通道使能
NVIC_Init(&NVIC_InitStructure); //根据上面指定的参数初始化NVIC寄存器
```

三、总结：中断优先级设置步骤

1. 系统运行后先设置**中断优先级分组**。调用函数：

```
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);
```

整个系统执行过程中，只设置一次中断分组。

2. 针对**每个中断**，设置对应的抢占优先级和响应优先级：

```
void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
```

3. 如果需要挂起/解挂，查看中断当前激活状态，分别调用相关函数即可。