本章教程通过CH32V103开发板板载两个LED灯显示程序运行状态，具体情况如下：

- 串口调式助手大约每隔500ms打印一次”Enter interrupt“；
- 开发板上LED1与LED2不停闪烁，其中，LED1闪烁较快，大约为100ms一次；LED2闪烁较慢，大约为500ms闪烁一次。

1、TIM简介及相关函数介绍

CH32V103定时器包含1个高级16位定时器、3个通用16位定时器，以及2个看门狗定时器和1个系统时基定时器。

高级控制定时器(TIM1)是一个 16 位的自动装载计数器，具有可编程的预分频器。除了完整的通用定时器功能外， 可以被看成是分配到 6 个通道的三相 PWM 发生器， 具有带死区插入的互补 PWM 输出功能，允许在指定数目的计数器周期之后更新定时器进行重复计数周期，刹车功能等。高级控制定时器的很多功能都与通用定时器相同，内部结构也相同，因此高级控制定时器可以通过定时器链接功能与 TIM 定时器协同操作，提供同步或事件链接功能。

通用定时器（TIM2、TIM3和TIM4），其可同步运行，每个定时器都有一个 16 位的自动装载递加/递减计数器、一个可编程的 16 位预分频器和 4 个独立的通道，每个通道都可用于输入捕获、输出比较、PWM 生成和单脉冲模式输出。其能通过定时器链接功能与高级控制定时器共同工作，提供同步或事件链接功能。在调试模式下，计数器可以被冻结，同时 PWM 输出被禁止，从而切断由这些输出所控制的开关。 任意通用定时器都能用于产生 PWM 输出。每个定时器都有独立的 DMA 请求机制。这些定时器还能够处理增量编码器的信号，也能处理 1 至 3 个霍尔传感器的数字输出。

系统时基定时器（SysTick），这是内核控制器自带的一个定时器，用于产生 SYSTICK 异常，可专用于实时操作系统，为系统提供“心跳”节律，也可当成一个标准的64位递增计数器。以AHB时钟的8分频为基准时钟源。当计数器递增到设置比较值时，产生一个可屏蔽系统中

断。关于2个看门狗定时器，在前面教程有过介绍，在此不做过多赘述。

关于CH32V103定时器具体信息及其相关功能和实现等，可参考CH32V103数据手册和应用手册。本章教程主要通过CH32V103通用定时器TIM3编写一个定时器中断程序，并下载到开发板进行验证，程序编写所需相关函数在库函数中进行调用，库函数文件中函数介绍如下：

1. void TIM_DeInit(TIM_TypeDef* TIMx);
2. void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct);
3. void TIM_OC1Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct);
4. void TIM_OC2Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct);
5. void TIM_OC3Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct);
6. void TIM_OC4Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct);
7. void TIM_ICInit(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct);
8. void TIM_PWMIConfig(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct);
9. void TIM_BDTRConfig(TIM_TypeDef* TIMx, TIM_BDTRInitTypeDef *TIM_BDTRInitStruct);
10. void TIM_TimeBaseStructInit(TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct);
11. void TIM_OCStructInit(TIM_OCInitTypeDef* TIM_OCInitStruct);
12. void TIM_ICStructInit(TIM_ICInitTypeDef* TIM_ICInitStruct);
13. void TIM_BDTRStructInit(TIM_BDTRInitTypeDef* TIM_BDTRInitStruct);
14. void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState);
15. void TIM_CtrlPWMOutputs(TIM_TypeDef* TIMx, FunctionalState NewState);
16. void TIM_ITConfig(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState);
17. void TIM_GenerateEvent(TIM_TypeDef* TIMx, uint16_t TIM_EventSource);
18. void TIM_DMAConfig(TIM_TypeDef* TIMx, uint16_t TIM_DMABase, uint16_t TIM_DMABurstLength);
19. void TIM_DMACmd(TIM_TypeDef* TIMx, uint16_t TIM_DMASource, FunctionalState NewState);
20. void TIM_InternalClockConfig(TIM_TypeDef* TIMx);
21. void TIM_ITRxExternalClockConfig(TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource);
22. void TIM_TIxExternalClockConfig(TIM_TypeDef* TIMx, uint16_t TIM_TIxExternalCLKSource,uint16_t TIM_ICPolarity, uint16_t ICFilter);

23. void TIM_ETRClockMode1Config(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity,uint16_t ExtTRGFilter);
24. void TIM_ETRClockMode2Config(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity, uint16_t ExtTRGFilter);
25. void TIM_ETRConfig(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity,uint16_t ExtTRGFilter);
26. void TIM_PrescalerConfig(TIM_TypeDef* TIMx, uint16_t Prescaler, uint16_t TIM_PSCReloadMode);
27. void TIM_CounterModeConfig(TIM_TypeDef* TIMx, uint16_t TIM_CounterMode);
28. void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource);
29. void TIM_EncoderInterfaceConfig(TIM_TypeDef* TIMx, uint16_t TIM_EncoderMode,uint16_t TIM_IC1Polarity, uint16_t TIM_IC2Polarity);
30. void TIM_ForcedOC1Config(TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction);
31. void TIM_ForcedOC2Config(TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction);
32. void TIM_ForcedOC3Config(TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction);
33. void TIM_ForcedOC4Config(TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction);
34. void TIM_ARRPreloadConfig(TIM_TypeDef* TIMx, FunctionalState NewState);
35. void TIM_SelectCOM(TIM_TypeDef* TIMx, FunctionalState NewState);
36. void TIM_SelectCCDMA(TIM_TypeDef* TIMx, FunctionalState NewState);
37. void TIM_CCPreloadControl(TIM_TypeDef* TIMx, FunctionalState NewState);
38. void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPreload);
39. void TIM_OC2PreloadConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPreload);
40. void TIM_OC3PreloadConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPreload);
41. void TIM_OC4PreloadConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPreload);
42. void TIM_OC1FastConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCFast);
43. void TIM_OC2FastConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCFast);
44. void TIM_OC3FastConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCFast);

45. void TIM_OC4FastConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCFast);
46. void TIM_ClearOC1Ref(TIM_TypeDef* TIMx, uint16_t TIM_OCClear);
47. void TIM_ClearOC2Ref(TIM_TypeDef* TIMx, uint16_t TIM_OCClear);
48. void TIM_ClearOC3Ref(TIM_TypeDef* TIMx, uint16_t TIM_OCClear);
49. void TIM_ClearOC4Ref(TIM_TypeDef* TIMx, uint16_t TIM_OCClear);
50. void TIM_OC1PolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPolarity);
51. void TIM_OC1NPolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCNPolarity);
52. void TIM_OC2PolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPolarity);
53. void TIM_OC2NPolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCNPolarity);
54. void TIM_OC3PolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPolarity);
55. void TIM_OC3NPolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCNPolarity);
56. void TIM_OC4PolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPolarity);
57. void TIM_CCxCmd(TIM_TypeDef* TIMx, uint16_t TIM_Channel, uint16_t TIM_CCx);
58. void TIM_CCxNCmd(TIM_TypeDef* TIMx, uint16_t TIM_Channel, uint16_t TIM_CCxN);
59. void TIM_SelectOCxM(TIM_TypeDef* TIMx, uint16_t TIM_Channel, uint16_t TIM_OCMode);
60. void TIM_UpdateDisableConfig(TIM_TypeDef* TIMx, FunctionalState NewState);
61. void TIM_UpdateRequestConfig(TIM_TypeDef* TIMx, uint16_t TIM_UpdateSource);
62. void TIM_SelectHallSensor(TIM_TypeDef* TIMx, FunctionalState NewState);
63. void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, uint16_t TIM_OPMode);
64. void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_TRGOSource);
65. void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_SlaveMode);
66. void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_MasterSlaveMode);
67. void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter);
68. void TIM_SetAutoreload(TIM_TypeDef* TIMx, uint16_t Autoreload);
69. void TIM_SetCompare1(TIM_TypeDef* TIMx, uint16_t Compare1);
70. void TIM_SetCompare2(TIM_TypeDef* TIMx, uint16_t Compare2);
71. void TIM_SetCompare3(TIM_TypeDef* TIMx, uint16_t Compare3);
72. void TIM_SetCompare4(TIM_TypeDef* TIMx, uint16_t Compare4);
73. void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
74. void TIM_SetIC2Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
75. void TIM_SetIC3Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);

```
76. void TIM_SetIC4Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
77. void TIM_SetClockDivision(TIM_TypeDef* TIMx, uint16_t TIM_CKD);
78. uint16_t TIM_GetCapture1(TIM_TypeDef* TIMx);
79. uint16_t TIM_GetCapture2(TIM_TypeDef* TIMx);
80. uint16_t TIM_GetCapture3(TIM_TypeDef* TIMx);
81. uint16_t TIM_GetCapture4(TIM_TypeDef* TIMx);
82. uint16_t TIM_GetCounter(TIM_TypeDef* TIMx);
83. uint16_t TIM_GetPrescaler(TIM_TypeDef* TIMx);
84. FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, uint16_t
    TIM_FLAG);
85. void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG);
86. ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, uint16_t TIM_IT);
87. void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, uint16_t TIM_IT);
```

复制代码

1.1、void TIM_DeInit(TIM_TypeDef* TIMx)
功 能：将TIMx外围寄存器初始化为其默认重置值。
输 入：无

1.2、void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功 能：根据TIM_TimeBaseInitStruct中指定的参数初始化TIMx时基单元外围设备。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TimeBaseInitStruct：指向TIM_TimeBaseInitTypeDef结构的指针。

1.3、void TIM_OC1Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
功 能：根据TIM_OCInitStruct中指定的参数初始化TIMx Channel1。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_OCInitStruct：指向TIM_OCInitTypeDef结构的指针。

1.4、void TIM_ICInit(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)
功 能：根据TIM_ICInitStruct中指定的参数初始化TIM外围设备。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_ICInitStruct：指向TIM_ICInitTypeDef结构的指针。
1.5、void TIM_PWMIConfig(TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)
功 能：根据TIM结构中的指定参数配置TIM外围设备，以测量外部PWM信号。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_ICInitStruct：指向TIM_ICInitTypeDef结构的指针。

1.6、void TIM_BDTRConfig(TIM_TypeDef* TIMx, TIM_BDTRInitTypeDef *TIM_BDTRInitStruct)

功 能：配置：中断特性、死区时间、锁定级别、OSSI、OSSR状态和AOE（自动输出启用）。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_BDTRInitStruct：指向TIM_BDTRInitTypeDef结构的指针。

1.7、void TIM_TimeBaseStructInit(TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功 能：用默认值填充每个TIM_TimeBaseInitStruct成员。
输 入：TimeBaseInitStruct：指向TIM_TimeBaseInitTypeDef结构的指针。

1.8、void TIM_OCStructInit(TIM_OCInitTypeDef* TIM_OCInitStruct)
功 能：用默认值填充每个TIM_OCInitStruct成员。
输 入：TIM_OCInitStruct：指向TIM_OCInitTypeDef结构的指针。

1.9、void TIM_ICStructInit(TIM_ICInitTypeDef* TIM_ICInitStruct)
功 能：用其默认值填充每个TIM_ICInitStruct成员。
输 入：TIM_ICInitStruct：指向TIM_ICInitTypeDef结构的指针。

1.10、void TIM_BDTRStructInit(TIM_BDTRInitTypeDef* TIM_BDTRInitStruct)
功 能：用默认值填充每个TIM_BDTRInitStruct成员。
输 入：TIM_BDTRInitStruct：指向TIM_BDTRInitTypeDef结构的指针。

1.11、void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：启用或禁用指定的TIM外围设备。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.12、void TIM_CtrlPWMOutputs(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：启用或禁用TIM外围设备主输出。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.13、void TIM_ITConfig(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState)
功 能：启用或禁用指定的TIM中断。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_IT：指定要启用或禁用的TIM中断源；NewState:启用或禁用。

1.14、void TIM_GenerateEvent(TIM_TypeDef* TIMx, uint16_t TIM_EventSource)
功 能：将TIMx外围寄存器初始化为其默认重置值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_EventSource：指定事件源。

1.15、void TIM_DMAConfig(TIM_TypeDef* TIMx, uint16_t TIM_DMABase, uint16_t TIM_DMABurstLength)
功 能：配置TIMx的DMA接口。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_DMABase：DMA基址；TIM_DMABurstLength：DMA突发长度。

1.16、void TIM_DMACmd(TIM_TypeDef* TIMx, uint16_t TIM_DMASource, FunctionalState NewState)
功 能：启用或禁用TIMx的DMA请求。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_DMASource：指定DMA请求源；NewState:启用或禁用。

1.17、void TIM_InternalClockConfig(TIM_TypeDef* TIMx)
功 能：配置TIMx内部时钟。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备。

1.18、void TIM_ITRxExternalClockConfig(TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource)
功 能：将TIMx内部触发器配置为外部时钟。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_InputTriggerSource：触发源。

1.19、void TIM_TIxExternalClockConfig(TIM_TypeDef* TIMx, uint16_t TIM_TIxExternalCLKSource,uint16_t TIM_ICPolarity, uint16_t ICFilter)
功 能：将TIMx触发器配置为外部时钟。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_TIxExternalCLKSource: 触发器源；ICFilter：指定过滤器值。此参数的值必须介于0x0和0xF之间。

1.20、void TIM_ETRClockMode1Config(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity,uint16_t ExtTRGFilter)
功 能：配置外部时钟模式1。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_ExtTRGPrescaler：外部触发预分频器；TIM_ExtTRGPolarity：外部触发极性；ExtTRGFilter：外部触发器筛选器。

1.21、void TIM_ETRConfig(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity,uint16_t ExtTRGFilter)
功 能：配置TIMx外部触发器（ETR）。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_ExtTRGPrescaler：外部触发预分频器；TIM_ExtTRGPolarity：外部触发极性；ExtTRGFilter：外部触发器筛选器。

1.22、void TIM_PrescalerConfig(TIM_TypeDef* TIMx, uint16_t Prescaler, uint16_t TIM_PSCReloadMode)
功 能：配置TIMx预分频器。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；Prescaler：指定预分频器寄存器值；TIM_PSCReloadMode：指定TIM预分频器重新加载模式。

1.23、void TIM_CounterModeConfig(TIM_TypeDef* TIMx, uint16_t TIM_CounterMode)
功 能：指定要使用的TIMx计数器模式。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_CounterMode：指定要使用的计数器模式。

1.24、void TIM_SelectInputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource)
功 能：选择输入触发源。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_InputTriggerSource:输入触发源。

1.25、void TIM_EncoderInterfaceConfig(TIM_TypeDef* TIMx, uint16_t TIM_EncoderMode,uint16_t TIM_IC1Polarity, uint16_t TIM_IC2Polarity)
功 能：配置TIMx编码器接口。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_EncoderMode：指定TIMx编码器模式；TIM_IC1Polarity：指定IC1极性；TIM_IC2Polarity：指定IC2极性。

1.26、void TIM_ForcedOC1Config(TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction)
功 能：强制TIMx输出1波形为激活或非激活电平。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_ForcedAction：指定要设置为输出波形的强制操作。

1.27、void TIM_ARRPreloadConfig(TIM_TypeDef* TIMx, FunctionalState NewState)

功 能：在ARR上启用或禁用TIMx外设预加载寄存器。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.28、void TIM_SelectCOM(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：选择TIM外围换向事件。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.29、void TIM_SelectCCDMA(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：选择TIMx外围捕获比较DMA源。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.30、void TIM_CCPreloadControl(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：设置或重置TIM外围捕获比较预加载控制位。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.31、void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPreload)
功 能：启用或禁用CCR1上的TIMx外围预加载寄存器。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_OCPreload：TIMx外围预加载寄存器的新状态。

1.32、void TIM_OC1FastConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCFast)
功 能：配置TIMx输出比较1快速功能。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_OCFast：输出比较快速启用位的新状态。

1.33、void TIM_ClearOC1Ref(TIM_TypeDef* TIMx, uint16_t TIM_OCClear)
功 能：清除或保护外部事件上的OCREF1信号。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_OCClear：输出比较清除启用位的新状态。

1.34、void TIM_OC1PolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPolarity)
功 能：配置TIMx通道1极性。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；

TIM_OCPolarity：指定OC1极性。

1.35、void TIM_OC1NPolarityConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCNPolarity)
功 能：配置TIMx通道1极性。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_OCNPolarity：指定OC1N极性。

1.36、void TIM_CCxCmd(TIM_TypeDef* TIMx, uint16_t TIM_Channel, uint16_t TIM_CCx)
功 能：启用或禁用TIM捕获比较通道x。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_Channel：指定TIM通道；TIM_CCx：指定TIM信道CCxE位的新状态。

1.37、void TIM_CCxNCmd(TIM_TypeDef* TIMx, uint16_t TIM_Channel, uint16_t TIM_CCxN)
功 能：启用或禁用TIM捕获比较通道xN。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_Channel：指定TIM通道；TIM_CCxN：指定TIM信道CCxNE位的新状态。

1.38、void TIM_SelectOCxM(TIM_TypeDef* TIMx, uint16_t TIM_Channel, uint16_t TIM_OCMode)
功 能：选择TIM输出比较模式。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_Channel：指定TIM通道；TIM_OCMODE：指定TIM输出比较模式。

1.39、void TIM_UpdateDisableConfig(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：启用或禁用TIMx更新事件。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.40、void TIM_UpdateRequestConfig(TIM_TypeDef* TIMx, uint16_t TIM_UpdateSource)
功 能：配置TIMx更新请求中断源。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_UpdateSource：指定更新源。

1.41、void TIM_SelectHallSensor(TIM_TypeDef* TIMx, FunctionalState NewState)
功 能：启用或禁用TIMx的霍尔传感器接口。

输 入：TIMx：其中x可以是1到4来选择TIM外围设备；NewState:启用或禁用。

1.42、void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, uint16_t TIM_OPMode)
功 能：选择TIMx的单脉冲模式。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_OPMode：指定要使用的OPM模式。

1.43、void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_TRGOSource)
功 能：选择TIMx触发器输出模式。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_TRGOSource：指定触发器输出源。

1.44、void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_SlaveMode)
功 能：选择TIMx从机模式。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_SlaveMode：指定定时器从模式。

1.45、void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_MasterSlaveMode)
功 能：设置或重置TIMx主/从模式。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_MasterSlaveMode：指定定时器主从模式。

1.46、void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter)
功 能：设置TIMx计数器寄存器值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；Counter：指定计数器寄存器的新值。

1.47、void TIM_SetAutoreload(TIM_TypeDef* TIMx, uint16_t Autoreload)
功 能：设置TIMx自动重新加载寄存器值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；Autoreload：指定Autoreload寄存器的新值。

1.48、void TIM_SetCompare1(TIM_TypeDef* TIMx, uint16_t Compare1)
功 能：设置TIMx捕获比较1寄存器值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；Compare1：指定捕获Compare1寄存器的新值。

1.49、void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC)
功 能：设置TIMx输入捕捉1预分频器。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_ICPSC：指定输入Capture1预分频器的新值。

1.50、void TIM_SetClockDivision(TIM_TypeDef* TIMx, uint16_t TIM_CKD)
功 能：设置TIMx时钟刻度值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_CKD：指定时钟分频值。

1.51、uint16_t TIM_GetCapture1(TIM_TypeDef* TIMx)
功 能：获取TIMx输入捕获1的值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备。

1.52、uint16_t TIM_GetCounter(TIM_TypeDef* TIMx)
功 能：获取TIMx计数器值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备。

1.53、uint16_t TIM_GetPrescaler(TIM_TypeDef* TIMx)
功 能：获取TIMx预分频器值。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备。

1.54、FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)
功 能：检查是否设置了指定的TIM标志。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_FLAG：指定要检查的标志。

1.55、void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)
功 能：清除TIMx的挂起标志。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_FLAG：指定要清除的标志位。

1.56、ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, uint16_t TIM_IT)
功 能：检查TIM中断是否发生。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_IT：指定要检查的TIM中断源。

1.57、void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, uint16_t

TIM_IT)
功 能：清除TIMx的中断挂起的位。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；TIM_IT：指定要清除的挂起位。

1.58、static void TI1_Config(TIM_TypeDef* TIMx, uint16_t TIM_ICPolarity, uint16_t TIM_ICSelection,uint16_t TIM_ICFilter)
功 能：将TI1配置为输入。
输 入：TIMx：其中x可以是1到4来选择TIM外围设备；
TIM_ICPolarity：输入极性；TIM_ICSelection：指定要使用的输入；
TIM_ICFilter：指定输入捕获筛选器。

以上函数均为库函数内部函数，在进行使用时只需在程序中进行调用即可。

2、硬件设计

本章教程通过定时器中断控制LED灯闪烁，其中，定时器为CH32V103内部资源，无需进行硬件设计，只需进行LED连接即可，LED与GPIO引脚连接方式如下：

- LED1与PA0连接；
- LED2与PA1连接。

3、软件设计

本章教程主要通过定时器中断控制LED灯闪烁，其中，led.c文件与led.h文件前面GPIO教程中已讲解，在此不再介绍，本章主要介绍time.h文件、time.c文件以及main.c文件，具体程序如下：
time.h文件

```
#ifndef __TIME_H
#define __TIME_H

#include "ch32v10x_conf.h"

void TIM3_Int_Init(u16 arr,u16 psc);
```

复制代码
time.h文件主要是定时器初始化配置函数的声明。
time.c文件

```
#include "time.h"
#include "led.h"

void TIM3_IRQHandler(void) __attribute__((interrupt("WCH-Interrupt-fast")));
```

```
5.
6. void TIM3_Int_Init(u16 arr,u16 psc)
7. {
8.     TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
9.     NVIC_InitTypeDef NVIC_InitStructure;
10.
11.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); //
    使能TIM3时钟
12.
13.     TIM_TimeBaseStructure.TIM_Period = arr;  //指定下次更新事件时
    要加载到活动自动重新加载寄存器中的周期值。
14.     TIM_TimeBaseStructure.TIM_Prescaler =psc; //指定用于划分TIM时
    钟的预分频器值。
15.     TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
      //时钟分频因子
16.     TIM_TimeBaseStructure.TIM_CounterMode =
    TIM_CounterMode_Up; //TIM计数模式，向上计数模式
17.     TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); //根据指定的
    参数初始化TIMx的时间基数单位
18.
19.     TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE ); //使能TIM3中断，允
    许更新中断
20.
21.     //初始化TIM NVIC，设置中断优先级分组
22.     NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;          //TIM3
    中断
23.     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //设
    置抢占优先级0
24.     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;       //设置
    响应优先级3
25.     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;          //使
    能通道1中断
26.     NVIC_Init(&NVIC_InitStructure); //初始化NVIC
27.
28.     TIM_Cmd(TIM3, ENABLE); //TIM3使能
29. }
30.
31. void TIM3_IRQHandler(void)
32. {
33.     static u8 i=0;
34.
35.     if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) //检查TIM3中
    断是否发生。
36.       {
37.         TIM_ClearITPendingBit(TIM3,TIM_IT_Update);    //清除TIM3的
    中断挂起位。
38.         printf("Enter interrupt\n");
39.         GPIO_WriteBit(GPIOA, GPIO_Pin_1, (i==0) ? (i=Bit_SET):
    (i=Bit_RESET));
```

```
40.     }
41. }
```

复制代码

time.c文件主要包含一个定时器3初始化函数和一个中断服务函数，程序具体执行步骤如下：

1、使能定时器时钟；

```
1. RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); //使能TIM3时钟
```

复制代码

2、设置定时器初始化结构体参数，包括定时器周期、预分频器、时钟分频、计数模式等；

```
1. TIM_TimeBaseStructure.TIM_Period = arr;  //指定下次更新事件时要加载到活动自动重新加载寄存器中的周期值。
2.  TIM_TimeBaseStructure.TIM_Prescaler =psc; //指定用于划分TIM时钟的预分频器值。
3.  TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  //时钟分频因子
4.  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM计数模式，向上计数模式
5.  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); //根据指定的参数初始化TIMx的时间基数单位
```

复制代码

3、使能定时器中断；

```
1. TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE ); //使能TIM3中断，允许更新中断
```

复制代码

4、定时器中断优先级设置；

```
1.  //初始化TIM NVIC，设置中断优先级分组
2.  NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;          //TIM3中断
3.  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //设置抢占优先级0
4.  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;      //设置响应优先级3
5.  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;         //使能通道1中断
6.  NVIC_Init(&NVIC_InitStructure); //初始化NVIC
```

复制代码

5、使能定时器；

```
1. TIM_Cmd(TIM3, ENABLE); //TIM3使能
```

复制代码
6、编写中断服务函数。

```
void TIM3_IRQHandler(void)
{
    static u8 i=0;

    if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) //检查TIM3中断是否发生。
      {
        TIM_ClearITPendingBit(TIM3,TIM_IT_Update);    //清除TIM3的中断挂起位。
        printf("Enter interrupt\n");
        GPIO_WriteBit(GPIOA, GPIO_Pin_1, (i==0) ? (i=Bit_SET): (i=Bit_RESET));
      }
}
```

复制代码
main.c文件

```
int main(void)
{
        u8 j=0;

        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
        Delay_Init();
        USART_Printf_Init(115200);
        LED_Init();
        TIM3_Int_Init(4999,7199);
        printf("SystemClk:%d\r\n",SystemCoreClock);

        while(1)
        {
            GPIO_WriteBit(GPIOA, GPIO_Pin_0, (j==0) ? (j=Bit_SET): (j=Bit_RESET));
            Delay_Ms(100);
        }
}
```

复制代码
main.c文件主要包含相关函数初始化以及一个while循环，其中while循环中LED1闪烁用来和中断函数中LED2闪烁形成对比，LED1闪烁较快，LED2闪烁较慢。

4、下载验证

将编译好的程序下载到开发板并复位，可以看到开发板两个LED灯闪烁，其中，LED1闪烁较快，LED2闪烁较慢。同时，串口调试助手打印显示：Enter interrupt，表示程序进入中断，大约500ms打印一次。

串口调试助手打印显示如下：