本章教程通过CH32V103的串口与上位机之间进行数据的发送和接收，具体执行步骤如下：

- 上位机向CH32V103发送数据；
- CH32V103接收数据并发送给上位机；
- 上位机接收CH32V103发送的数据并显示。

1、USART简介及相关函数介绍

USART全称为Universal Synchronous/Asynchronous Receiver/Transmitter（通用同步/异步串行接收/发送器），是一个全双工通用同步/异步串行收发模块，是一个高度灵活的串行通信设备。嵌入式中所说串口一般为UART，其全称为Universal Asynchronous Receiver/Transmitter（通用异步串行接收/发送器）。相较于USART，UART只有异步通信功能，即串口通信时无需对外提供时钟输出。

CH32V103的USART支持全双工或半双工的同步或异步通信，支持NRZ数据格式，支持分数波特率发生器（最高4.5Mbps）、支持可编程的数据长度和可配置的停止位，支持LIN、IrDA编码器和智能卡，支持DMA并具有多种中断源。

USART双向通信至少需要2个引脚：接收数据输入（RX）和发送数据输出（TX）。

- RX：接收数据输入。
- TX：发送数据输出。

串口收发通过调用库函数内部相关函数并结合相关寄存器进行配置实现通信。库函数内部函数及介绍如下：

1. void USART_DeInit(USART_TypeDef* USARTx);
2. void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct);
3. void USART_StructInit(USART_InitTypeDef* USART_InitStruct);
4. void USART_ClockInit(USART_TypeDef* USARTx, USART_ClockInitTypeDef* USART_ClockInitStruct);
5. void USART_ClockStructInit(USART_ClockInitTypeDef* USART_ClockInitStruct);

6. void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
7. void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState);
8. void USART_DMACmd(USART_TypeDef* USARTx, uint16_t USART_DMAReq, FunctionalState NewState);
9. void USART_SetAddress(USART_TypeDef* USARTx, uint8_t USART_Address);
10. void USART_WakeUpConfig(USART_TypeDef* USARTx, uint16_t USART_WakeUp);
11. void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState NewState);
12. void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, uint16_t USART_LINBreakDetectLength);
13. void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState NewState);
14. void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
15. uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
16. void USART_SendBreak(USART_TypeDef* USARTx);
17. void USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime);
18. void USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler);
19. void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState);
20. void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState NewState);
21. void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState NewState);
22. void USART_OverSampling8Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
23. void USART_OneBitMethodCmd(USART_TypeDef* USARTx, FunctionalState NewState);
24. void USART_IrDAConfig(USART_TypeDef* USARTx, uint16_t USART_IrDAMode);
25. void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState);
26. FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);
27. void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG);
28. ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT);
29. void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);

复制代码

1.1、void USART_DeInit(USART_TypeDef* USARTx)

功 能：将USARTx外围寄存器初始化为其默认重置值。通俗讲即可通过该函数实现串口复位。

输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。

1.2、void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)

功 能：串口参数初始化。根据USART_InitStruct中的指定参数初始化USART外设。

输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；
USART_InitStruct指向包含指定USART外围设备的配置信息的USART_InitTypeDef结构的指针。

1.3、void USART_StructInit(USART_InitTypeDef* USART_InitStruct)
功 能：用默认值填充每个USART_InitStruct成员。
输 入：USART_InitStruct：指向要初始化的USART_InitTypeDef结构的指针。

1.4、void USART_ClockInit(USART_TypeDef* USARTx,
USART_ClockInitTypeDef* USART_ClockInitStruct)
功 能：根据USART_ClockInitStruct中指定的参数初始化USARTx外围时钟。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；
USART_ClockInitStruct：指向包含指定USART外围设备的配置信息的USART_ClockInitTypeDef结构的指针。

1.5、void USART_ClockStructInit(USART_ClockInitTypeDef*
USART_ClockInitStruct)
功 能：用默认值填充每个USART_ClockInitStruct成员。
输 入：USART_ClockInitStruct：指向将被初始化的USART_ClockInitTypeDef结构的指针。

1.6、void USART_Cmd(USART_TypeDef* USARTx, FunctionalState
NewState)
功 能：启用或禁用指定的USART外围设备。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；NewState为对应状态：ENABLE，使能；DISABLE，禁止使能。

1.7、void USART_ITConfig(USART_TypeDef* USARTx, uint16_t
USART_IT, FunctionalState NewState)
功 能：启用或禁用指定的USART中断。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；USART_IT为指定要启用或禁用的USART中断源；NewState为对应状态：ENABLE，使能中断；DISABLE，禁止使能中断。

1.8、void USART_DMACmd(USART_TypeDef* USARTx, uint16_t
USART_DMAReq, FunctionalState NewState)
功 能：启用或禁用USART DMA接口。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；
USART_DMAReq：指定DMA请求。USART_DMAReq_Tx:USART DMA发送请求。USART_DMAReq_Rx:USART DMA接收请求。NewState为对应状态：ENABLE，使能USART DMA；DISABLE，禁止使能USART DMA。

1.9、void USART_SetAddress(USART_TypeDef* USARTx, uint8_t
USART_Address)
功 能：设置USART节点的地址。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；

USART_Address：表示USART节点的地址。

1.10、void USART_WakeUpConfig(USART_TypeDef* USARTx, uint16_t USART_WakeUp)
功 能：选择USART唤醒方法。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；
USART_WakeUp：指定USART唤醒方法。USART_WakeUp_IdleLine：由空闲线路检测唤醒。USART_WakeUp_AddressMark：通过地址标记唤醒。

1.11、void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：确定USART是否处于静音模式。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；NewState为对应状态：ENABLE，启用静音模式；DISABLE，禁止启用静音模式。

1.12、void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, uint16_t USART_LINBreakDetectLength)
功 能：设置USART LIN中断检测长度。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；
USART_LINBreakDetectLength：指定LIN中断检测长度。
USART_LINBreakDetectLength_10b：10位中断检测。
USART_LINBreakDetectLength_11b：11位中断检测。

1.13、void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用USART LIN模式。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；NewState为对应状态：ENABLE，启用USART LIN模式；DISABLE，禁止启用USART LIN模式。

1.14、void USART_SendData(USART_TypeDef* USARTx, uint16_t Data)
功 能：通过USARTx外围设备传输单个数据。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备；Data为要发送传输的数据。

1.15、uint16_t USART_ReceiveData(USART_TypeDef* USARTx)
功 能：返回USARTx外围设备最近接收的数据。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。

1.16、void USART_SendBreak(USART_TypeDef* USARTx)
功 能：传输中断字符。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。

1.17、void USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime)
功 能：USART设置指定的时间。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。

USART_GuardTime：指定保护时间。

1.18、void USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler)
功 能：设置系统时钟预分频器。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。
USART_Prescaler：指定预分频器时钟。

1.19、void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用USART智能卡模式。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。NewState为对应状态：ENABLE，启用USART智能卡模式；DISABLE，禁止启用USART智能卡模式。

1.20、void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用NACK传输。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。NewState为对应状态：ENABLE，启用NACK传输；DISABLE，禁止启用NACK传输。

1.21、void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用USART半双工通信。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。NewState为对应状态：ENABLE，启用USART半双工通信；DISABLE，禁止启用USART半双工通信。

1.22、void USART_OverSampling8Cmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用USART的8x过采样模式。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。NewState为对应状态：ENABLE，启用USART的8x过采样模式；DISABLE，禁止启用USART的8x过采样模式。

1.23、void USART_OneBitMethodCmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用USART的一位采样方法。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。NewState为对应状态：ENABLE，启用USART的一位采样方法；DISABLE，禁止启用USART的一位采样方法。

1.24、void USART_IrDAConfig(USART_TypeDef* USARTx, uint16_t USART_IrDAMode)
功 能：配置USART的IrDA接口。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。
USART_IrDAMode：指定IrDA模式。USART_IrDAMode_LowPower和

USART_IrDAMode_Normal。

1.25、void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState)
功 能：启用或禁用USART的IrDA接口。
输 入：USARTx：其中x可以是1、2或3来选择UART外围设备。NewState为对应状态：ENABLE，启用USART的IrDA接口；DISABLE，禁止启用USART的IrDA接口。

1.26、FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG)
功 能：检查是否设置了指定的USART标志。
输 入：USARTx：其中x可以是1、2、3来选择USART外围设备；
USART_FLAG：指定要检查的标志。

1.27、void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG)
函 能：清除USARTx的挂起标志。
输 入：USARTx：其中x可以是1、2、3来选择USART外围设备；
USART_FLAG：指定要清除的标志。

1.28、ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT)
功 能：检查指定的USART中断是否已发生。
输 入：USARTx：其中x可以是1、2、3来选择USART外围设备；
USART_IT：指定要检查的USART中断源。

1.29、void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT)
功 能：用于清除USARTx的中断挂起位。
输 入：USARTx：其中x可以是1、2、3来选择USART外围设备；
USART_IT：指定要清除的中断挂起位。

串口收发程序除了要用到上述相关函数以外，还需用到中断，中断相关函数保存在ch32v10x_misc.c和ch32v10x_misc.h文件中，ch32v10x_misc.h文件主要是相关函数的声明定义，此处主要对ch32v10x_misc.c文件内部相关中断函数进行介绍，ch32v10x_misc.c文件内部函数如下：

```
1. void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);
2. void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
```

复制代码
1.30、void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)
功 能：配置优先级分组：抢占优先级和子优先级。
输 入：NVIC_PriorityGroup：指定优先级分组位长度。

1.31、void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)
功 能：根据NVIC_InitStruct中指定的参数初始化NVIC外围设备。

输 入：NVIC_InitStruct：指向包含指定NVIC外围设备配置信息的NVIC_InitTypeDef结构的指针。

沁恒为了实现CH32V103系列MCU与CH32F103系列MCU的兼容，在函数中将CH32V103系列MCU的中断PFIC（快速可编程中断控制器）同样定义为了NVIC（内置嵌套向量中断控制器），并在core_riscv.h文件中进行相关定义说明，如下所示：

```
1. #define PFIC            ((PFIC_Type *) 0xE000E000 )
2. #define NVIC             PFIC
```

复制代码
以上函数均为本次串口收发实验需要用到或可能用到的函数，在进行串口收发程序编写时，相关函数只需在程序中进行调用即可。

2、硬件设计

由于CH32V103系列MCU的串口1在debug文件中被用于调试打印，因此本次教程使用串口2进行收发验证。由CH32V103数据手册可知，串口2对应引脚为PA2和PA3引脚，PA2为USART2_TX，PA3为USART2_RX。使用杜邦线将WCH-Link模块与CH32V103开发板串口2连接起来，连接方式如下：

- WCH-Link模块RX引脚与CH32V103开发板PA2引脚连接；
- WCH-Link模块TX引脚与CH32V103开发板PA3引脚连接。

3、软件设计

串口收发在传输过程中，串口发送端将字节数据以串行方式逐个比特发送出去，串口接收端逐个比特接收数据，然后重新将其组织为字节数据。串口收发程序具体实现步骤为：

- 定义一个GPIO_InitTypeDef类型结构体，一个USART_InitTypeDef类型结构体、NVIC_InitTypeDef类型体；
- 使能串口2 RX引脚和TX引脚GPIO时钟和USART2时钟；
- GPIO端口模式设置及串口参数初始化；
- 配置中断控制器；
- 使能串口并使能USART接收中断；
- 编写发送单个数据函数和发送字符串函数；
- 编写USART接收中断服务函数并实现数据接收和发送。

根据上述步骤编写串口收发程序，程序如下：
usart.h文件

```
1. #ifndef __USART_H
2. #define __USART_H
3.
4. #include "ch32v10x_conf.h"
5.
6. void USARTx_CFG(void);
7. void USARTx_SendByte(USART_TypeDef* pUSARTx, uint8_t data);
```

```
8. void USARTx_SendStr(USART_TypeDef* pUSARTx, char *str);
9. void USART2_IRQHandler(void);
10.
11. #endif
```

复制代码

usart.h文件为USART头文件，主要用于保存函数声明。

usart.c文件

```
1. #include "usart.h"
2.
3. void USART2_IRQHandler(void) __attribute__((interrupt("WCH-Interrupt-
   fast")));
4.
5. /*******************************************************************************
6. * Function Name  : USARTx_CFG
7. * Description    : Initializes the USART2  peripheral.
8. * Input          : None
9. * Return         : None
10. *******************************************************************************/
11. void USARTx_CFG(void)
12. {
13.   GPIO_InitTypeDef  GPIO_InitStructure;
14.   USART_InitTypeDef USART_InitStructure;
15.   NVIC_InitTypeDef  NVIC_InitStructure;
16.
17.   RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE); //使能串
   口2时钟
18.   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  //使能
   GPIOA时钟
19.
20.   USART_DeInit(USART2);
21.
22.   /* USART2 TX-->A.2   RX-->A.3 */
23.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
24.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
25.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;              //设置PA2
   为复用推挽输出
26.   GPIO_Init(GPIOA, &GPIO_InitStructure);
27.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
28.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;        //设置
   PA3为浮空输入
29.   GPIO_Init(GPIOA, &GPIO_InitStructure);
30.
31.   USART_InitStructure.USART_BaudRate = 115200;                 //设置串口波特
   率为115200
32.   USART_InitStructure.USART_WordLength = USART_WordLength_8b;  //字
   长为8位数据格式
33.   USART_InitStructure.USART_StopBits = USART_StopBits_1;       //1个停止位
34.   USART_InitStructure.USART_Parity = USART_Parity_No;          //无奇偶校验
   位
```

```
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;//无硬件流控制
  USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx; //发送和接收模式
  USART_Init(USART2, &USART_InitStructure);                    //初始化串口

  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;      //抢占优先级为1
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;           //子优先级为1
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;              //IRQ通道使能
  NVIC_Init(&NVIC_InitStructure);                              //中断优先级初始化

  USART_Cmd(USART2, ENABLE);                                   //使能串口
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);               //开启中断
}

void USARTx_SendByte(USART_TypeDef* pUSARTx, uint8_t data)
{
    USART_SendData(pUSARTx, data);
    while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
}

void USARTx_SendStr(USART_TypeDef* pUSARTx, char *str)
{
    uint8_t i = 0;
    do
    {
       USARTx_SendByte(pUSARTx, *(str+i));
       i++;
    }while(*(str+i) != '\0');
    while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TC) == RESET);
}

/*****************************************************************************
* Function Name  : USART2_IRQHandler
* Description    : This function handles USART2 global interrupt request.
* Input          : None
* Return         : None
*****************************************************************************/
void USART2_IRQHandler(void)
{
    uint8_t ucTemp;
    if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) //中断产生
    {
       USART_ClearITPendingBit(USART2,USART_IT_RXNE);    //清除中断标志
       ucTemp = USART_ReceiveData(USART2);               //接收数据
       USART_SendData(USART2, ucTemp);                   //发送数据
    }
}
```

复制代码

usart.c文件是USART串口的配置程序，本教程通过串口2进行数据发送和接收，其配置流程如下：

1.串口时钟使能以及GPIO时钟使能；

```
1. RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE); //使能串口2时钟
2. RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  //使能GPIOA时钟
```

复制代码

2.串口复位

```
1. USART_DeInit(USART2);
```

复制代码

3.GPIO端口模式设置及串口参数初始化

```
1.  /* USART2 TX-->A.2   RX-->A.3 */
2.  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
3.  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
4.  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;             //设置PA2为复用推挽输出
5.  GPIO_Init(GPIOA, &GPIO_InitStructure);
6.  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
7.  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;       //设置PA3为浮空输入
8.  GPIO_Init(GPIOA, &GPIO_InitStructure);
9.
10.  USART_InitStructure.USART_BaudRate = 115200;               //设置串口波特率为115200
11.  USART_InitStructure.USART_WordLength = USART_WordLength_8b;  //字长为8位数据格式
12.  USART_InitStructure.USART_StopBits = USART_StopBits_1;      //1个停止位
13.  USART_InitStructure.USART_Parity = USART_Parity_No;        //无奇偶校验位
14.  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;//无硬件流控制
15.  USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;//发送和接收模式
16.  USART_Init(USART2, &USART_InitStructure);                  //初始化串口
```

复制代码

4.中断初始化

```
1.  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
2.  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;     //抢占优先级为1
3.  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;          //子优先级为1
4.  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;             //IRQ通道使能
5.  NVIC_Init(&NVIC_InitStructure);                             //中断优先级初始化
```

复制代码

5.串口使能及开启中断

```
1. USART_Cmd(USART2, ENABLE);                                   //使能串口
2. USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);               //开启中断
```

复制代码

以上内容均为USARTx_CFG函数，主要进行USART初始化配置。

6.发送字符函数

```
1. void USARTx_SendByte(USART_TypeDef* pUSARTx, uint8_t data)
2. {
3.     USART_SendData(pUSARTx, data);
4.     while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
5. }
```

复制代码

Usart_SendByte 函数通过调用库函数USART_SendData 实现发送一个ASCLL 码值字符，并通过使用 USART_GetFlagStatus 函数来获取 USART事件标志来实现发送完成功能等待。通过循环检测发送数据寄存器为空这个标志，当跳出 while 循环时说明发送数据寄存器为空。

7.发送字符串函数

```
1. void USARTx_SendStr(USART_TypeDef* pUSARTx, char *str)
2. {
3.     uint8_t i = 0;
4.     do
5.     {
6.         USARTx_SendByte(pUSARTx, *(str+i));
7.       i++;
8.     }while(*(str+i) != '\0');
9.     while(USART_GetFlagStatus(pUSARTx, USART_FLAG_TC) == RESET);
10. }
```

复制代码

Usart_SendString 函数用来发送一个字符串，其通过调用 Usart_SendByte 函数发送每个字符，直到遇到空字符即停止发送。最后使用循环检测发送完成的事件标志 TC 来实现保证数据发送完成后才退出函数。

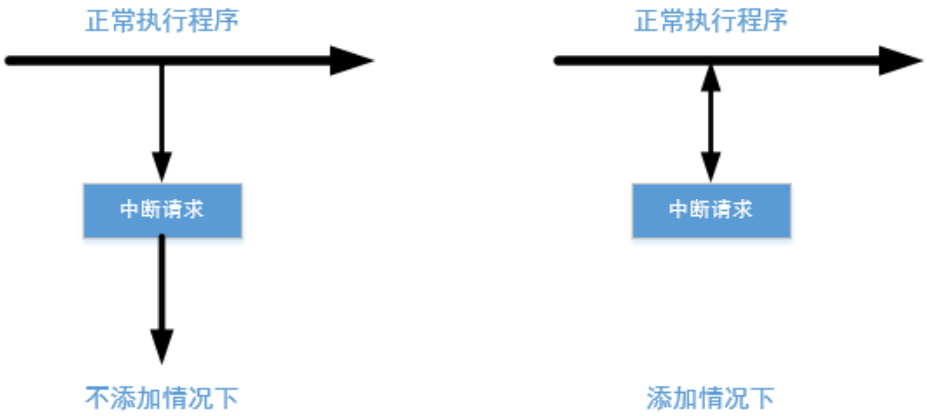
8.USART中断服务函数

```
1. void USART2_IRQHandler(void)
2. {
3.   uint8_t ucTemp;
4.   if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) //中断产生
5.    {
6.     USART_ClearITPendingBit(USART1,USART_IT_RXNE);    //清除中断标志
7.     ucTemp = USART_ReceiveData(USART2);               //接收数据
8.     USART_SendData(USART2, ucTemp);                   //发送数据
9.    }
10. }
```

复制代码

在串口初始化配置中使能USART接收中断，当 USART 接收到数据就会执行 USART_IRQHandler 函数。 其中，USART_GetITStatus 函数专门用来获取中断事件标志，并返回该标志位状态；if 语句用来判断是否真的产生 USART 数据接收这个中断事件，若为真则通过调用函数 USART_ReceiveData 读取数据到指定存储区，再通过调用函数USART_SendData 把数据发送到串口调试助手。

关于文件头部void USART2_IRQHandler(void) __attribute__((interrupt("WCH-Interrupt-fast")))函数，其用于保证中断执行完成之后程序正常运行。若不添加此函数，中断执行完成之后，中断执行之前保存的信息将不会被返回，中断会被认为一个正常程序继续执行下去，但中断之后并没有程序，从而导致程序出现类似跑飞状况。添加此函数，中断执行完成之后，中断执行之前保存的信息将会被返回，从而程序可以继续正常执行，如图所示。



main.c文件

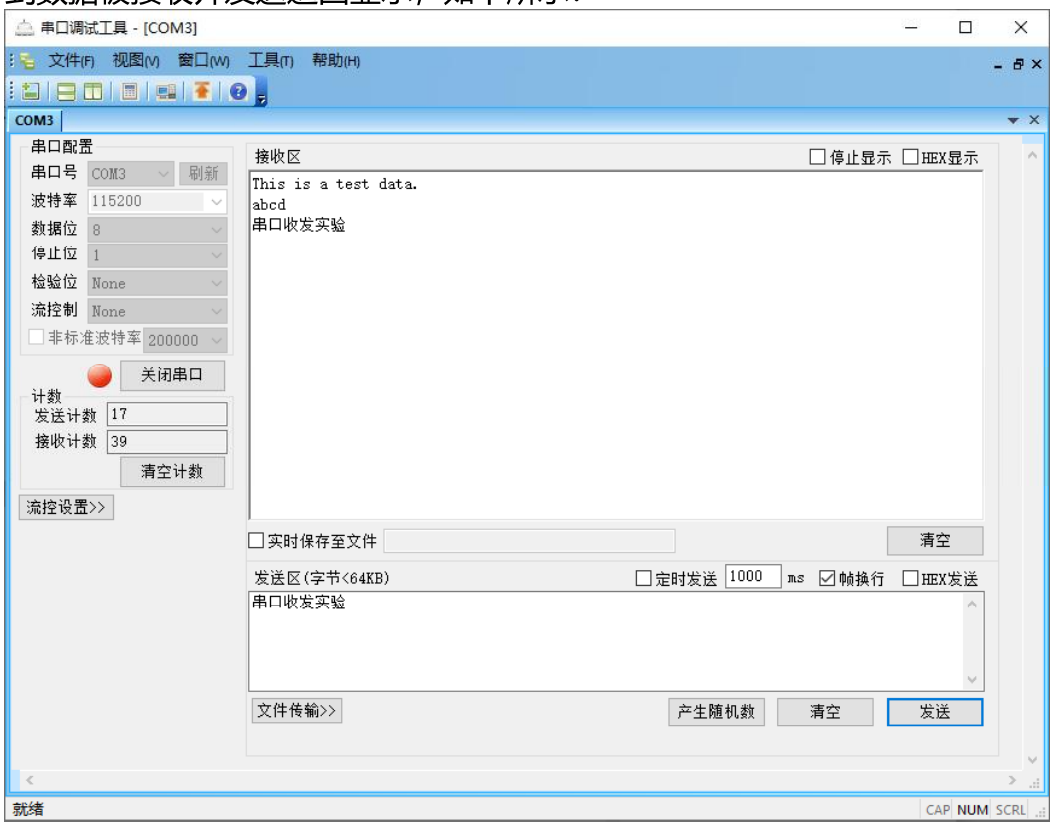
```
#include "debug.h"
#include "usart.h"

/*********************************************************************
* Function Name  : main
* Description    : Main program.
* Input          : None
* Return         : None
*********************************************************************/
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    USARTx_CFG();

    USARTx_SendStr(USART2, "This is a test data.\n");

    while(1);
```

```
19. }
```

复制代码
main函数首先设置NVIC中断分组，然后初始化USART并发送一个字符串，最后循环等待。

4、下载验证

将编译好的程序下载到开发板并复位，打开串口调试助手，发送数据，可以看到数据被接收并发送返回显示，如下所示：



本教程只是进行简单的数据发送和接收，无法保证批量数据收发准确性。