1.1.1 软件设计：中断的方式接收（寄存器）

USART 提供了多个中断事件。

中断事件	事件标志	使能位
发送数据寄存器空	TXE	TXEIE
CTS标志	CTS	CTSIE
发送完成	TC	TCIE
接收数据就绪可读	TXNE	TXNEIE
检测到数据溢出	ORE	
检测到空闲线路	IDLE	IDLEIE
奇偶检验错	PE	PEIE
断开标志	LBD	LBDIE
噪声标志，多缓冲通信中的溢出错误和帧错误	NE或ORT或FE	EIE(1)

1．仅当使用DMA接收数据时，才使用这个标志位。

USART的各种中断事件被连接到同一个中断向量(见下图)，有以下各种中断事件：

- 发送期间：发送完成、清除发送、发送数据寄存器空。
- 接收期间：空闲总线检测、溢出错误、接收数据寄存器非空、校验错误、LIN断开符号检测、噪音标志(仅在多缓冲器通信)和帧错误(仅在多缓冲器通信)。

如果设置了对应的使能控制位，这些事件就可以产生各自的中断。

1.1.1.1 Driver_USART.c 添加中断相关代码

```
#include "Driver_USART.h"

/**
 * @description: 初始化串口 1
 */
void Driver_USART1_Init(void)
{
    /* 1. 开启时钟 */
    /* 1.1 串口 1 外设的时钟 */
    RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    /* 1.2 GPIO 时钟 */
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

    /* 2. 配置 GPIO 引脚的工作模式  PA9=Tx(复用推挽 CNF=10 MODE=11)  PA10=Rx(浮空输入
CNF=01 MODE=00)*/
    GPIOA->CRH |= GPIO_CRH_CNF9_1;
    GPIOA->CRH &= ~GPIO_CRH_CNF9_0;
    GPIOA->CRH |= GPIO_CRH_MODE9;

    GPIOA->CRH &= ~GPIO_CRH_CNF10_1;
    GPIOA->CRH |= GPIO_CRH_CNF10_0;
    GPIOA->CRH &= ~GPIO_CRH_MODE10;
```

```
    /* 3. 串口的参数配置 */
    /* 3.1 配置波特率 115200 */
    USART1->BRR = 0x271;
    /* 3.2 配置一个字的长度 8 位 */
    USART1->CR1 &= ~USART_CR1_M;
    /* 3.3 配置不需要校验位 */
    USART1->CR1 &= ~USART_CR1_PCE;
    /* 3.4 配置停止位的长度 */
    USART1->CR2 &= ~USART_CR2_STOP;
    /* 3.5 使能接收和发送 */
    USART1->CR1 |= USART_CR1_TE;
    USART1->CR1 |= USART_CR1_RE;

    /* 3.6 使能串口的各种中断 */
    USART1->CR1 |= USART_CR1_RXNEIE; /* 接收非空中断 */
    USART1->CR1 |= USART_CR1_IDLEIE; /* 空闲中断 */

    /* 4. 配置 NVIC */
    /* 4.1 配置优先级组 */
    NVIC_SetPriorityGrouping(3);
    /* 4.2 设置优先级 */
    NVIC_SetPriority(USART1_IRQn, 2);
    /* 4.3 使能串口 1 的中断 */
    NVIC_EnableIRQ(USART1_IRQn);

    /* 4. 使能串口 */
    USART1->CR1 |= USART_CR1_UE;
}

/**
 * @description: 发送一个字节
 * @param {uint8_t} byte 要发送的字节
 */
void Driver_USART1_SendChar(uint8_t byte)
{
    /* 1. 等待发送寄存器为空 */
    while ((USART1->SR & USART_SR_TXE) == 0)
        ;

    /* 2. 数据写出到数据寄存器 */
    USART1->DR = byte;
}

/**
```

```
 * @description: 发送一个字符串
 * @param {uint8_t} *str 要发送的字符串
 * @param {uint16_t} len 字符串中字节的长度
 * @return {*}
 */
void Driver_USART1_SendString(uint8_t *str, uint16_t len)
{
    for (uint16_t i = 0; i < len; i++)
    {
        Driver_USART1_SendChar(str[i]);
    }
}

/**
 * @description: 接收一个字节的数据
 * @return {*} 接收到的字节
 */
uint8_t Driver_USART1_ReceiveChar(void)
{
    /* 等待数据寄存器非空 */
    while ((USART1->SR & USART_SR_RXNE) == 0)
        ;
    return USART1->DR;
}

/**
 * @description: 接收变长数据.接收到的数据存入到 buff 中
 * @param {uint8_t} buff 存放接收到的数据
 * @param {uint8_t} *len 存放收到的数据的字节的长度
 */
void Driver_USART1_ReceiveString(uint8_t buff[], uint8_t *len)
{
    uint8_t i = 0;
    while (1)
    {
        // 等待接收非空
        while ((USART1->SR & USART_SR_RXNE) == 0)
        {
            // 在等待期间，判断是否收到空闲帧
            if (USART1->SR & USART_SR_IDLE)
            {

                *len = i;
                return;
```

```
            }
        }
        buff[i] = USART1->DR;
        i++;
    }
}

/* 缓冲接收到的数据 */
uint8_t buff[100] = {0};
/* 存储接收到的字节的长度 */
uint8_t len = 0;
uint8_t isToSend = 0;
void USART1_IRQHandler(void)
{
    /* 数据接收寄存器非空 */
    if (USART1->SR & USART_SR_RXNE)
    {
        // 对 USART_DR 的读操作可以将接收非空的中断位清零。 所以不用单独清除了.
        //USART1->SR &= ~USART_SR_RXNE;
        buff[len] = USART1->DR;
        len++;
    }
    else if (USART1->SR & USART_SR_IDLE)
    {
        /* 清除空闲中断标志位: 先读 sr,再读 dr.就可以实现清除了 */
        USART1->SR;
        USART1->DR;
        /* 变长数据接收完毕 */
        //Driver_USART1_SendString(buff, len);
        isToSend = 1;
        /* 把接收字节的长度清 0 */
        // len = 0;
    }
}
```

1.1.1.2 main.c

```
#include "Driver_USART.h"
/* 缓冲接收到的数据 */
extern uint8_t buff[100];
/* 存储接收到的字节的长度 */
extern uint8_t len;
extern uint8_t isToSend;

int main()
{
```

```
    Driver_USART1_Init();
    Driver_USART1_SendString("abc", 3);
    while (1)
    {
        if(isToSend){
            Driver_USART1_SendString(buff, len);
            isToSend = 0;
            len = 0;
        }
    }
}
```