HAL库是比较全面的，封装比较彻底的，也是功能比较强大的。  
使用HAL库，我们直接调用它的API函数，不用关心它的底层操作过程。  
使用HAL库，省去了好多繁琐的处理过程，不再需要我们自己写如等待等过程。  
HAL库也包含如Ethernet、USB等高级外设的驱动。  
对于初接触它的人来说，尤其是用惯了标准库的人，总会有各种不适应和排斥感。  
就拿UART来说，我们通过中断方式接受或发送数据。如果仿真调试的话，会发现UART有开关中断的现象，而不是中断一直开着。  
  
下面，就讲解UART常用函数以及中断处理过程。  
  
**HAL\_StatusTypeDef HAL\_UART\_Transmit(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size, uint32\_t Timeout);**  
**串口发送，发送指定长度的数据。如果超时没发送完成，则不再发送，返回超时标志（HAL\_TIMEOUT）。**  
**HAL\_StatusTypeDef HAL\_UART\_Receive(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size, uint32\_t Timeout);**  
**串口接收，接收指定长度的数据。如果超时没接收完成，则不再接收数据到指定缓冲区，返回超时标志（HAL\_TIMEOUT）。**  
  
**HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)；**  
**串口中断发送，以中断方式发送指定长度的数据。**  
**大致过程是，把 发送缓冲区指针 指向 要发送的数据，设置 发送长度，发送计数器初值，然后使能串口发送中断，触发串口中断。**  
**再然后，串口中断函数处理，直到数据发送完成，而后关闭中断，不再发送数据，串口发送完成回调函数。**  
**HAL\_StatusTypeDef HAL\_UART\_Receive\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size);**  
**串口中断接收，以中断方式接收指定长度数据。**  
**大致过程是，把 接收缓冲区指针 指向 要存放接收数据的数组，设置 接收长度，接收计数器初值，然后使能串口接收中断。接收到数据时，会触发串口中断。**  
**再然后，串口中断函数处理，直到接收到指定长度数据，而后关闭中断，不再触发接收中断，调用串口接收完成回调函数。**  
  
**HAL\_StatusTypeDef HAL\_UART\_Transmit\_DMA(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size);**  
**串口DMA发送，以DMA方式发送指定长度的数据。**  
**过程是，把 发送缓冲区指针 指向 要发送的数据，设置 发送长度，发送计数器初值，设置 DMA传输完成中断的回调函数，使能DMA控制器中断，使能DMA控制器传输，使能UART的DMA传输请求。**  
**然后，UART便会发送数据，直到发送完成，触发DMA中断。**  
**DMA中断处理，如果 DMA模式 是 循环模式，则 直接 调用 DMA传输完成中断的回调函数。**  
**如果 DMA模式 是 正常模式，则 先 关闭DMA传输完成中断，不再触发DMA中断，再 调用 DMA传输完成中断的回调函数。**  
**DMA传输完成中断的回调函数处理过程，如果 DMA模式 是 循环模式，则 直接 调用 串口发送完成回调函数。**  
**如果 DMA模式 是 正常模式，则 先关闭 UART的DMA传输请求， 再 使能串口传输完成中断，直到传输完成，触发中断。**  
**串口传输完成中断处理，关闭中断，调用串口发送完成回调函数。**  
**HAL\_StatusTypeDef HAL\_UART\_Receive\_DMA(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size);**  
**串口DMA接收，以DMA方式接收指定长度的数据。**  
**过程是，把 接收缓冲区指针 指向 要存放接收数据的数组，设置 接收长度，接收计数器初值，设置 DMA传输完成中断的回调函数，使能DMA控制器中断，使能DMA控制器传输，使能UART的DMA传输请求。**  
**然后，UART接收到数据，便会通过DMA把数据存到接收缓冲区，直到接收到指定长度数据，触发DMA中断。**  
**DMA中断处理，如果 DMA模式 是 循环模式，则 直接 调用 DMA传输完成中断的回调函数。**  
**如果 DMA模式 是 正常模式，则 先 关闭DMA传输完成中断，不再触发DMA中断，再 调用 DMA传输完成中断的回调函数。**  
**DMA传输完成中断的回调函数处理过程，如果 DMA模式 是 循环模式，则 直接 调用 串口接收完成回调函数。**  
**如果 DMA模式 是 正常模式，则 先关闭 UART的DMA传输请求， 再 调用 串口接收完成回调函数。**  
  
  
**由于函数较多和过长的缘故，下面仅以HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)为例，分析源码：**  
/\*\*  
  \* @brief  Sends an amount of data in non blocking mode.  
  \* @param  huart: pointer to a UART\_HandleTypeDef structure that contains  
  \*                the configuration information for the specified UART module.  
  \* @param  pData: Pointer to data buffer  
  \* @param  Size: Amount of data to be sent  
  \* @retval HAL status  
  \*/                   **串口中断发送，以中断方式发送指定长度的数据。**  
HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)  
{                  
  /\* Check that a Tx process is not already ongoing \*/  
  if(huart->gState == HAL\_UART\_STATE\_READY)    **如果 串口空闲，则执行以下语句。**  
  {  
    if((pData == NULL ) || (Size == 0U))      **如果发送数据为空或者发送长度为0，则返回错误。**  
    {  
      return HAL\_ERROR;  
    }  
  
    /\* Process Locked \*/  
    \_\_HAL\_LOCK(huart);                **上锁。**  
  
    huart->pTxBuffPtr = pData;               **结构体变量 huart 的 参数设置。发送缓冲区，发送长度，发送计数器。**  
    huart->TxXferSize = Size;  
    huart->TxXferCount = Size;  
  
    huart->ErrorCode = HAL\_UART\_ERROR\_NONE;  
    huart->gState = HAL\_UART\_STATE\_BUSY\_TX;           **状态设为 发送繁忙。**  
  
    /\* Process Unlocked \*/  
    \_\_HAL\_UNLOCK(huart);             **解锁。**  
  
   /\* Enable the UART Transmit data register empty Interrupt \*/  
    SET\_BIT(huart->Instance->CR1, USART\_CR1\_TXEIE);              **使能UART发送数据寄存器空中断，则会触发串口中断（发送中断）。**  
  
    return HAL\_OK;  
  }  
  else               **如果 串口忙，则返回 忙状态。**  
  {  
    return HAL\_BUSY;     
  }  
}