USART1（串口 1）实验

作者	fire
E-Mail	firestm32@foxmail.com
QQ	313303034
博客	firestm32.blog.chinaunix.net
硬件平台	野火 STM32 开发板
库版本	ST3.0.0

实验描述：重新实现 C 库中的 printf() 函数到串口 1，这样我们就可以像用 C 库中的 printf() 函数一样将信息通过串口打印到电脑，非常方便我们程序的调试。

硬件连接：PA9 - USART1(Tx)

PA10 - USART1(Rx)

库文件 ：startup/start_stm32f10x_hd.c

CMSIS/core_cm3.c

CMSIS/system_stm32f10x.c

FWlib/stm32f10x_gpio.c

FWlib/stm32f10x_rcc.c

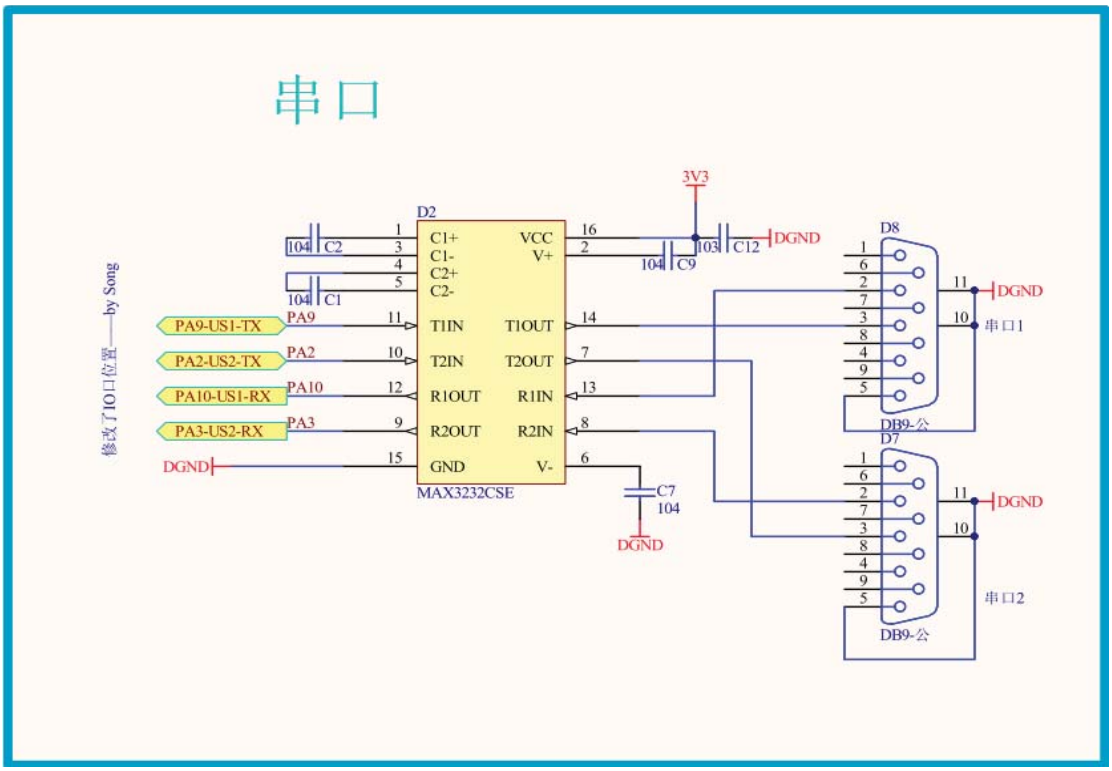
FWlib/stm32f10x_usart.c

用户文件：USER/main.c

USER/stm32f10x_it.c

USER/usart1.c

野火 STM32 开发板串口硬件原理图：



当我们在学习一款 CPU 的时候，最经典的实验莫过于流水灯了，会了流水灯的话就基本等于学会会操作 I/O 口了。那么在学会操作 I/O 之后，面对那么多的片上外设我们又应该先学什么呢？有些朋友会说用到什么就学什么，听起来这也不无道理呀。但对于我个人来说我会把学习串口的操作放在第二位，在程序运行的时候我们可以通过点亮一个 LED 来显示我们的状态，但有时候我们还想把某些中间量或者其他程序状态信息打印出来显示在电脑上，那么这时串口的作用就可想而知了。

实验讲解->

串口实验中我们用到了 GPIO、RCC、USART 这三个外设的库文件 stm32f10x_gpio.c、stm32f10x_rcc.c、stm32f10x_usart.c，所以我们先要把这个库文件添加进来，并在 stm32f10x_conf.h 中把相应的头文件的注释去掉，如下所示：

```
1.  /* Includes ------------------------------------------------------------------
    */
```

```
/* Uncomment the line below to enable peripheral header file inclusion */
/* #include "stm32f10x_adc.h" */
/* #include "stm32f10x_bkp.h" */
/* #include "stm32f10x_can.h" */
/* #include "stm32f10x_crc.h" */
/* #include "stm32f10x_dac.h" */
/* #include "stm32f10x_dbgmcu.h" */
/* #include "stm32f10x_dma.h" */
/* #include "stm32f10x_exti.h" */
/*#include "stm32f10x_flash.h"*/
/* #include "stm32f10x_fsmc.h" */
#include "stm32f10x_gpio.h"
/* #include "stm32f10x_i2c.h" */
/* #include "stm32f10x_iwdg.h" */
/* #include "stm32f10x_pwr.h" */
#include "stm32f10x_rcc.h"
/* #include "stm32f10x_rtc.h" */
/* #include "stm32f10x_sdio.h" */
/* #include "stm32f10x_spi.h" */
/* #include "stm32f10x_tim.h" */
#include "stm32f10x_usart.h"
/* #include "stm32f10x_wwdg.h" */
/*#include "misc.h"*/  /* High level functions for NVIC and SysTick (add-
on to CMSIS functions) */
```

配置好要用的库的环境之后，我们就从 main 函数看起，层层剥离源代码。

```
/*
 * 函数名：main
 * 描述  ：主函数
 * 输入  ：无
 * 输出  ：无
 */
int main(void)
{
    /* 配置系统时钟为 72M */
  SystemInit();
  /* USART1 config 115200 8-N-1 */
```

```
12.     USART1_Config();
13.     printf("\r\n this is a printf demo \r\n");
14.
15.     USART1_printf(USART1, "\r\n This is a USART1_printf demo \r\n");
16.     USART1_printf(USART1, "\r\n ("__DATE__ " - " __TIME__ ") \r\n");
17.     while (1)
18.     {  }   // 无限循环
19. }
20.
```

首先调用库函数 SystemInit()启动 CPU 的心脏，配置系统时钟为 72M。

紧 接着 调用函数 USART1_Config()，函数 USART1_Config()主要做了如下工作：

1-> 使能了串口 1 的时钟

2-> 配置好了 usart1 的 I/O

3- 配置好了 usart1 的工作模式，具体为波特率为 115200 、8 个数据位、1 个停止位、无硬件流控制。即 115200 8-N-1

USART1_Config() 在 usart1.c 这个文件中实现：

```
1.  /*
2.   * 函数名：USART1_Config
3.   * 描述  ：USART1 GPIO 配置,工作模式配置。115200 8-N-1
4.   * 输入  ：无
5.   * 输出  ：无
6.   * 调用  ：外部调用
7.   */
8.  void USART1_Config(void)
9.  {
10.     GPIO_InitTypeDef GPIO_InitStructure;
11.     USART_InitTypeDef USART_InitStructure;
12.
13.     /* config USART1 clock */
14.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, ENABLE
    );
15.
16.     /* USART1 GPIO config */
17.    /* Configure USART1 Tx (PA.09) as alternate function push-pull */
```

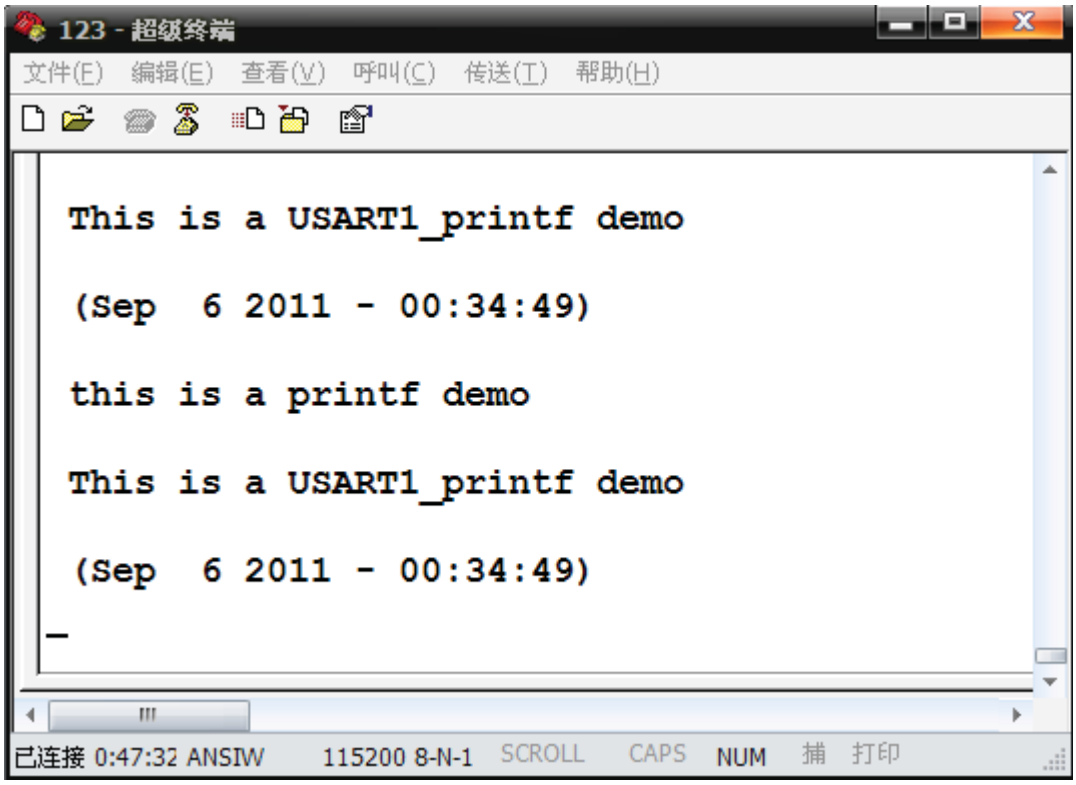
```
18.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
19.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
20.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
21.   GPIO_Init(GPIOA, &GPIO_InitStructure);
22.   /* Configure USART1 Rx (PA.10) as input floating */
23.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
24.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
25.   GPIO_Init(GPIOA, &GPIO_InitStructure);
26.
27.   /* USART1 mode config */
28.   USART_InitStructure.USART_BaudRate = 115200;
29.   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
30.   USART_InitStructure.USART_StopBits = USART_StopBits_1;
31.   USART_InitStructure.USART_Parity = USART_Parity_No ;
32. USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;

33.   USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
34.   USART_Init(USART1, &USART_InitStructure);
35.   USART_Cmd(USART1, ENABLE);
36. }
```

然后我们就通过下面的三行代码由串口往电脑里面的超级终端打印信息，打印的信息为一些字符串和当前的日期。

```
1.  printf("\r\n this is a printf demo \r\n");
2.
3.  USART1_printf(USART1, "\r\n This is a USART1_printf demo \r\n");
4.
5.  USART1_printf(USART1, "\r\n ("__DATE__ " - " __TIME__ ") \r\n");
```

下面是电脑超级终端的截图，从图可以看出程序是运行正确的。



调用这三个函数看似很简单，但在这三个函数的背后还得做些工作，我们先来看 printf() 这个函数，要想 printf() 函数工作的话，我们需要把 printf() 重新定向到串口中，这部分的工作是由 fputc(int ch, FILE *f) 这个函数来完成的，这个函数在 usart.c 中实现：

```
1.  /*
2.    * 函数名：fputc
3.    * 描述   ：重定向 c 库函数 printf 到 USART1
4.    * 输入   ：无
5.    * 输出   ：无
6.    * 调用   ：由 printf 调用
7.    */
8.  int fputc(int ch, FILE *f)
9.  {
10. /* 将 Printf 内容发往串口 */
11.    USART_SendData(USART1, (unsigned char) ch);
12.    while (!(USART1->SR & USART_FLAG_TXE));
13.
14.    return (ch);
15. }
```

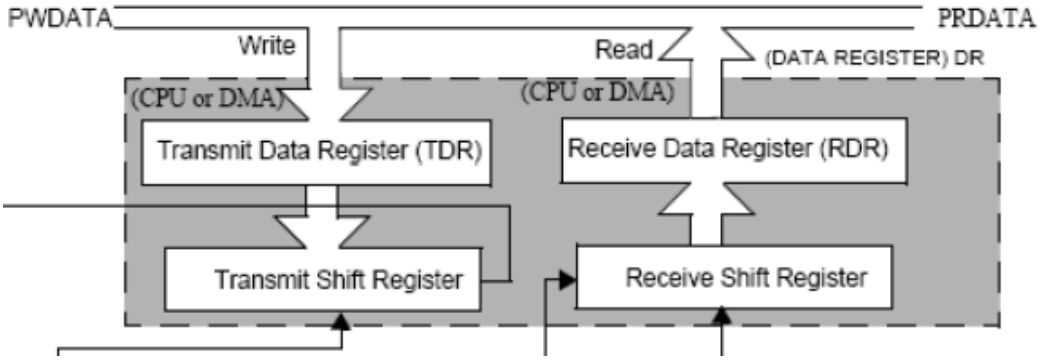
为了能够读懂 `fputc(int ch, FILE *f)` 里面的代码，我们需要了解下 usart 的几个关键的寄存器位，这样我们就会明白串口是如何将数据发送出去的，做到知其然又知其所以然。

下面是 USART_SR 寄存器（状态寄存器）中的位 7、位 6、位 5 的截图。理解好这几位的功能的话就可以大概理解串口是如何发送数据的了。其实我们在用库写自己的应用程序的时候要关注的最多的也就是这几个状态位，至于其他更底层的功能实现我们交给库就行了，倘我们要深究的话，只要阅读库中相应的源码即可。截图来自《stm32 中文参考手册》usart 这一章。

位7	**TXE**:发送数据寄存器空 当TDR寄存器中的数据被硬件转移到移位寄存器的时候，该位被硬件置位。如果USART_CR1寄存器中的TXEIE为1，则产生中断。对USART_DR的写操作，将该位清零。 0：数据还没有被转移到移位寄存器； 1：数据已经被转移到移位寄存器。 注意：单缓冲器传输中使用该位。
位6	**TC**：发送完成 当包含有数据的一帧发送完成后，由硬件将该位置位。如果USART_CR1中的TCIE为1，则产生中断。由软件序列清除该位(先读USART_SR，然后写入USART_DR)。TC位也可以通过写入0来清除，只有在多缓存通讯中才推荐这种清除程序。 0：发送还未完成； 1：发送完成成。
位5	**RXNE**：读数据寄存器非空 当RDR移位寄存器中的数据被转移到USART_DR寄存器中，该位被硬件置位。如果USART_CR1寄存器中的RXNEIE为1，则产生中断。对USART_DR的读操作可以将该位清零。RXNE位也可以通过写入0来清除，只有在多缓存通讯中才推荐这种清除程序。 0：数据没有收到； 1：收到数据，可以读出。

当 TXE 置位时，就表示发送数据寄存器 (TDR) 中的数据已经移到了发送移位寄存中，如果使能中断的话，就产生中断，在中断服务程序中我们就可以调用 USART_SendData() 函数将发送移位寄存中的数据通过串口发送出去，同时将该位清 0(硬件自动清 0)。当 RXNE 置位时，表示接收移位寄存器中的数据已经移到接收数据寄存器（RDR）中了，如果中断使能的话，则产生中断，在中断服务程序中我们可以调用 USART_ReceiveData() 将接收数据寄存器中的内容送到我们自定义的缓冲区中，比如数组，同时该位清 0(硬件自动清 0)。这只是描述了 usart 工作在中断模式的情况。

但 fputc(int ch, FILE *f) 函数里面用到的是查询的模式，其实道理还是一样的。我们先调用 USART_SendData(USART1, (unsigned char) ch); 将我们要发送的数据送到 TDR 中，之后我们就等待 TXE 置位，当 TXE 置位时就表示 TDR 中的 数据转移到了发送移位寄存器中了，这时我们就不用管后面的工作了，发送移位寄存器中的数据会由串口硬件自动发送，如此循环，直到将我们要发送的数据全部发送完为止。



现在我们再使点劲，在我们的 main.c 文件中把 stdio.h 这个头文件包含进来，这样我们就可以使用 printf() 这个函数了。

趁热打铁，让我们再来看看 USART1_printf(USART_TypeDef* USARTx, uint8_t *Data,...) 这个函数吧，它又调用了 itoa(int value, char *string, int radix)函数。关于这两个函数的具体实现请看 usart.c 中的源代码。这两个函数中有些变量是定义在 stdarg.h 这个头文件中的，所以在 usart.c 中我们需要把这个头文件包含进来，这个头文件位于 KDE 的根目录下。我们可以在这个路径下找到它：C:\Keil\ARM\RV31\INC。

```
1.  /*
2.   * 函数名：itoa
3.   * 描述  ：将整形数据转换成字符串
4.   * 输入  ：-radix =10 表示 10 进制，其他结果为 0
5.   *         -value 要转换的整形数
6.   *         -buf 转换后的字符串
7.   *         -radix = 10
8.   * 输出  ：无
9.   * 返回  ：无
10.  * 调用  ：被 USART1_printf()调用
11.  */
```

```
12. static char *itoa(int value, char *string, int radix)
13. {
14.     int     i, d;
15.     int     flag = 0;
16.     char    *ptr = string;
17. 
18.     /* This implementation only works for decimal numbers. */
19.     if (radix != 10)
20.     {
21.         *ptr = 0;
22.         return string;
23.     }
24. 
25.     if (!value)
26.     {
27.         *ptr++ = 0x30;
28.         *ptr = 0;
29.         return string;
30.     }
31. 
32.     /* if this is a negative value insert the minus sign. */
33.     if (value < 0)
34.     {
35.         *ptr++ = '-';
36.         /* Make the value positive. */
37.         value *= -1;
38.     }
39.     for (i = 10000; i > 0; i /= 10)
40.     {
41.         d = value / i;
42.         if (d || flag)
43.         {
44.             *ptr++ = (char)(d + 0x30);
45.             value -= (d * i);
46.             flag = 1;
47.         }
48.     }
49. 
50.     /* Null terminate the string. */
51.     *ptr = 0;
52.     return string;
53. } /* NCL_Itoa */
54. 
```

```
55. /*
56.  * 函数名：USART1 printf
57.  * 描述  ：格式化输出，类似于 C 库中的 printf，但这里没有用到 C 库
58.  * 输入  ：-USARTx 串口通道，这里只用到了串口 1，即 USART1
59.  *           -Data   要发送到串口的内容的指针
60.  *             -...    其他参数
61.  * 输出  ：无
62.  * 返回  ：无
63.  * 调用  ：外部调用
64.  *         典型应用 USART1_printf( USART1, "\r\n this is a demo \r\n" );
65.  *                  USART1_printf( USART1, "\r\n %d \r\n", i );
66.  *                  USART1_printf( USART1, "\r\n %s \r\n", j );
67.  */
68. void USART1 printf(USART TypeDef* USARTx, uint8 t *Data,...)
69. {
70.   const char *s;
71.   int d;
72.   char buf[16];
73.   va_list ap;
74.   va start(ap, Data);
75.     while ( *Data != 0)     // 判断是否到达字符串结束符
76.     {
77.         if ( *Data == 0x5c )  //'\'
78.         {
79.             switch ( *++Data )
```

```
            {
                case 'r':                                           //回车符
                    USART_SendData(USARTx, 0x0d);
                    Data ++;
                    break;

                case 'n':                                           //换行符
                    USART_SendData(USARTx, 0x0a);
                    Data ++;
                    break;

                default:
                    Data ++;
                    break;
            }
        }
        else if ( *Data == '%')
        {                                       //
            switch ( *++Data )
            {
                 case 's':      //字符串
                       s = va_arg(ap, const char *);
                 for ( ; *s; s++)
                 {
                  USART_SendData(USARTx,*s);
                 while( USART GetFlagStatus(USARTx, USART FLAG TC) == RESET );
                 }
                 Data++;
                 break;

                 case 'd':                                          //十进制
                 d = va_arg(ap, int);
                 itoa(d, buf, 10);
                 for (s = buf; *s; s++)
                 {
                 USART_SendData(USARTx,*s);
                 while( USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET );
                 }
                 Data++;
                 break;
                 default:
                       Data++;
                       break;
                   }
               } /* end of else if */
               else USART SendData(USARTx, *Data++);
               while( USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET );
            }
        }
```

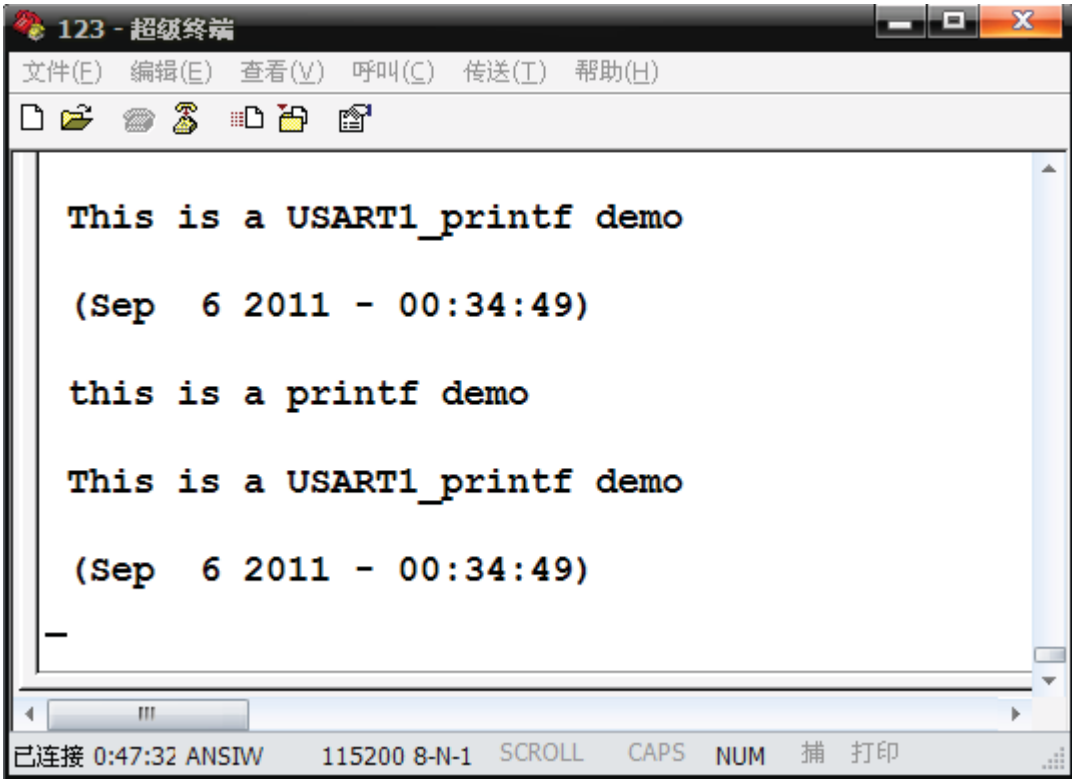
这部分代码有点多，在格式上编排不是很好，野火推荐大家直接看源码好点^_^

综上，我们已经可以用 printf() 和 USART1_printf() 这两个函数来打印信息了，但到底用哪个比较好呢？其实各有千秋， printf() 函数会受缓冲区大小的影响，有时候在用它打印的时候程序会发生莫名奇妙的错误，而实际上就是由于使用 printf() 这个函数引起的，其优点就是这种情况很少见且支持的格式较多。而 USART1_printf()则不会受缓冲区的影响产生莫名的错误，但其支持的格式较少。不过，相比之下，我还是比较喜欢用 USART1_printf()。

至此，关于串口的普通应用就差不多了，至于它的中断操作将会在接下来的教程中讲解。

实验现象->

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，打开超级终端，配置超级终端为 115200 8-N-1，将编译好的程序下载到开发板，即可看到超级终端打印出如下信息：



实验讲解完毕，野火祝大家学习愉快^_^。