

优秀不够，你是否无可替代

知识从未如此性感。烂程序员关心的是代码,好程序员关心的是数据结构和它们之间的关系 --QQ群: 607064330 --本人

QQ:946029359 --淘宝 <https://shop411638453.taobao.com/>

随笔 - 673, 文章 - 0, 评论 - 307, 阅读 - 166万

导航

博客园

首页

新随笔

联系

订阅 

管理

公告

渡我不渡她 -

Not available

00:00 / 00:00

1 渡我不渡她

2 小镇姑娘

3 PDD洪荒之力

 加入QQ群

昵称：杨奉武

园龄：5年6个月

粉丝：582

关注：1

搜索

找找看

谷歌搜索

我的标签

8266(88)

MQTT(50)

GPRS(33)

SDK(29)

Air202(28)

云服务器(21)

ESP8266(21)

Lua(18)

小程序(17)

STM32(16)

更多

随笔分类

Android(22)

Android 开发(8)

C# 开发(4)

CH395Q学习开发(1)

ESP32学习开发(4)

ESP8266 AT指令开发(基于STC89C52单片机)(3)

ESP8266 AT指令开发(基于STM32)(1)

ESP8266 AT指令开发基础入门篇备份(12)

ESP8266 LUA脚本语言开发(13)

4-1-关于环形队列

`<p><iframe name="ifd" src="https://mnifdv.cn/resource/cnblogs/单片机知识点总结/directory.html" frameborder="0" scrolling="auto" width="100%" height="1500"></iframe></p>`

数据处理思想和程序架构

资料源

码:<https://gitee.com/yang456/OpenProgrammingModule/>

点击加入群聊【单片机,物联网,上位机】：

说明1:知识从未如此性感。烂程序员关心的是代码,好程序员关心的是数据结构和它们之间的关系！

说明2:学的是思想，而非程序！此代码思路适用于所有的单片机。

说明3:学会以后,下面的代码可能会跟你一辈子！

说明4:这一系列文章是为大幅度裁剪本人博客文章！使博客文章更有条理。便于推其它教程！

目录:

- [01-来看下我的程序架子吧](#)
- [02-看看是不是你想要的按键处理](#)
- [03-单片机接收数据之空闲中断](#)
- [04_1-关于环形队列](#)
- [04_2-单片机接收数据之环形队列](#)
- [05-单片机接收数据之缓存管理,DMA](#)
- [06-单片机发送数据之中断发送](#)
- [07-单片机发送数据之环形队列](#)
- [08-单片机发送数据之缓存管理,DMA](#)
- [09-μCOS-II中内存管理程序使用说明](#)
- [10-数据缓存封装-内存管理实现](#)
- [11-给单片机写个回调函数怎么样](#)
- [12-单片机AT指令配置模块程序模板\(阻塞版\)](#)
- [13-单片机AT指令配置模块程序模板\(非阻塞版\)](#)
- [14-单片机加入JSON是个不错的选择](#)
- [15-IEEE754规约,浮点数和16进制之间的转换](#)

ESP8266 LUA开发基础入门篇
备份(22)
ESP8266 SDK开发(31)
ESP8266 SDK开发基础入门篇
备份(30)
GPRS Air202 LUA开发(11)
NB-IOT Air302 AT指令和LUA
脚本语言开发(24)
PLC(三菱PLC)基础入门篇(2)
STM32+Air724UG(4G模组)
物联网开发(41)
STM32+BC26/260Y物联网开
发(37)
STM32+ESP8266(ZLESP8266/
物联网开发(1)
STM32+ESP8266+AIR202/30:
基本控制方案(阿里云物联网平
台)(17)
STM32+ESP8266+AIR202/30:
远程升级方案(16)
STM32+ESP8266+AIR202/30:
终端管理方案(6)
STM32+ESP8266+Air302物
联网开发(40)
STM32+W5500+AIR202/302
基本控制方案(25)
STM32+W5500+AIR202/302
远程升级方案(6)
UCOSii操作系统(1)
W5500 学习开发(8)
编程语言C#(11)
编程语言Lua脚本语言基础入
门篇(6)
编程语言Python(1)
单片机(LPC1778)LPC1778(2)
单片机(MSP430)开发基础入门
篇(4)
单片机(STC89C51)单片机开发
板学习入门篇(3)
单片机(STM32)基础入门篇(3)
单片机(STM32)综合应用系列
(16)
电路模块使用说明(10)
感想(6)
软件安装使用: MQTT(8)
软件安装使用: OpenResty(6)
数据处理思想和程序架构(24)
数据库学习开发(12)
更多

最新评论

1. Re:ESP8266 SDK开发: 物
联网篇-ESP8266连接阿里云
物联网平台使用自定义Topic
实现自定义数据的上报和数
据下发
请问 如果我用ESP8266做了
一个路由器, 让其他设备用
它联网, 我还能用这个
ESP8266上云吗?
--糖果超甜会会长
2. Re:ESP8266 SDK开发: 物
联网篇-ESP8266连接阿里云
物联网平台使用自定义Topic
实现自定义数据的上报和数
据下发
跟着前辈高效学习!
--糖果超甜会会长

阅读排行榜

- [16-CRC校验](#)
- [17-1-单片机stm32的flash保存数据优化方案\(让擦写次数达到上百万至上千万次\)](#)
- [17-2-单片机STM32F407xx,F405xx,F415xx,417xx系列flash存储方案](#)
- [18-关于SSL](#)
- [19-单片机移植Mbedtls](#)
- [20-使用Mbedtls包中的SSL,和服务器进行网络加密通信](#)
- [21-对使用的数据进行优先等级排序的缓存](#)
-
-
-
-

环形队列是啥?

一看到名词就显得高大上了!!!

首先哈,对于做程序而言.一看到什么缓存什么队列,其实就是对数组进行操作.

话说以前有一个数组,这个数组假设是5个的



然后呢有人把这个数组交给了一套程序去管理

调用这套程序就可以往数组里存数据和取数据

但是呢,这套控制程序比较与众不同.

一开始调用控制程序往里面存一个字符A,A便会存储到数组的第一个位置

1. ESP8266使用详解(AT,LUA, SDK)(171303)
2. 1-安装MQTT服务器(Windows),并连接测试(94478)
3. ESP8266刷AT固件与node mcu固件(62926)
4. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(60806)
5. 有人WIFI模块使用详解(37737)
6. (一)基于阿里云的MQTT远程控制(Android 连接MQTT服务器,ESP8266连接MQTT服务器实现远程通信控制----简单的连接通信)(34782)
7. 关于TCP和MQTT之间的转换(31109)
8. android服务端+eps8266+单片机+路由器之远程控制系统(30969)
9. android 之TCP客户端编程(30720)
10. C#中public与private与static(29991)

推荐排行榜

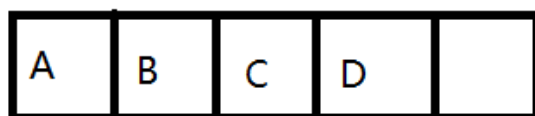
1. C#委托+回调详解(9)
2. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(8)
3. ESP8266使用详解(AT,LUA, SDK)(6)
4. 关于TCP和MQTT之间的转换(5)
5. 1-安装MQTT服务器(Windows),并连接测试(5)



然后再调用控制程序往里面存一个字符B,B便会存储到数组的第二个位置



然后再调用控制程序往里面存两个字符C和D,C,D便会存储到数组的第三,四的位置

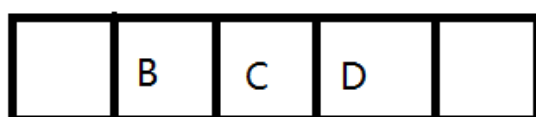


然后再调用控制程序让里面存储的时候,不能再存了,因为控制

程序默认已经满了.

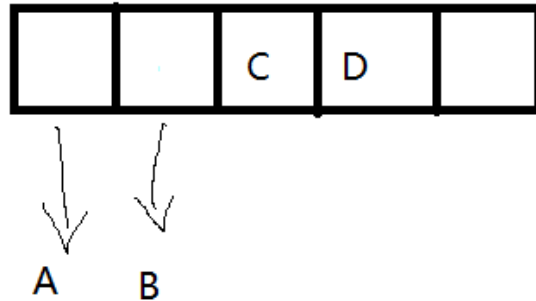
然后调用控制程序从里面取一个数据

第一个存进去的 A 便会被取出来,然后第一个位置就代表可以再存数据了

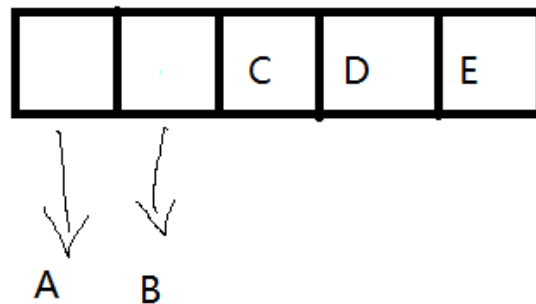


然后调用控制程序再从里面**取**一个数据

第二个存进去的 B 便会被取出来,然后第二个位置就代表可以再存数据了

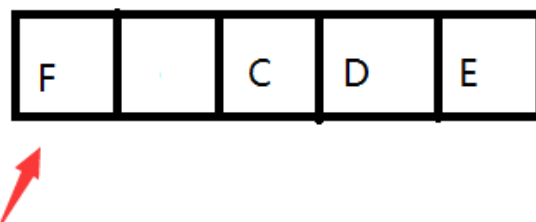


调用控制程序往里面**存**一个字符E,E便会存储到数组的第五个位置

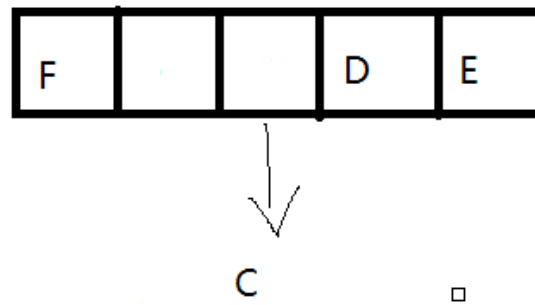


注意前方高能!

然后再调用控制程序往里面**存**一个字符F,F便会存储到数组的第一个位置



然后因为现在控制程序又认为满了,我就再调用控制程序
再从里面取一个数据



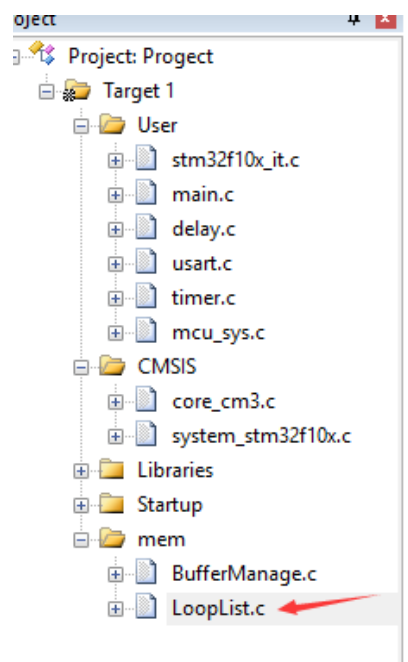
再调用控制程序往里面存一个字符G,G便会存储到数组
的第二个位置



然后就是这样子循环.

现在看看实际的

1.环形队列管理程序





```
/*
V1.0.2:
1.屏蔽printf打印
2.设置不同的返回值,以确定具体错误
*/

#define LOOPLIST_C_
#include "LoopList.h"
#include <string.h>
#include <stdio.h>

//创建或者说初始化环形缓冲区
void rbCreate(rb_t* rb, void *Buff, uint32_t BuffLen)
{
    if(NULL == rb)
    {
        //          printf("ERROR: input rb is NULL\n");
        return;
    }
    rb->rbCapacity = BuffLen;
    rb->rbBuff = Buff;
    rb->rbHead = rb->rbBuff; //头指向数组首地址
    rb->rbTail = rb->rbBuff; //尾指向数组首地址
}

//删除一个环形缓冲区
void rbDelete(rb_t* rb)
{
    if(NULL == rb)
    {
        //          printf("ERROR: input rb is NULL\n");
        return;
    }

    rb->rbBuff = NULL; //地址赋值为空
    rb->rbHead = NULL; //头地址为空
    rb->rbTail = NULL; //尾地址为空
    rb->rbCapacity = 0; //长度为空
}

//获取链表的长度
int32_t rbCapacity(rb_t *rb)
{
    if(NULL == rb)
    {
        //          printf("ERROR: input rb is NULL\n");
        return -51;
    }

    return rb->rbCapacity;
}

//返回能读的空间
int32_t rbCanRead(rb_t *rb)
{
    if(NULL == rb)
    {
        //          printf("ERROR: input rb is NULL\n");

```

```

        return -31;
    }

    if (rb->rbHead == rb->rbTail) //头与尾相遇
    {
        return 0;
    }

    if (rb->rbHead < rb->rbTail) //尾大于头
    {
        return rb->rbTail - rb->rbHead;
    }

    return rbCapacity(rb) - (rb->rbHead - rb->rbTail); //头大于尾
}

//返回能写入的空间
int32_t rbCanWrite(rb_t *rb)
{
    if(NULL == rb)
    {
        // printf("ERROR: input rb is NULL\n");
        return -41;
    }

    return rbCapacity(rb) - rbCanRead(rb); //总的减去已经写入的空间
}

/*
rb--要读的环形链表
data--读出的数据
count--读的个数
*/
int32_t rbRead(rb_t *rb, void *data, uint32_t count)
{
    int copySz = 0;

    if(NULL == rb) // printf("ERROR: input rb is NULL\n");
    {
        return -21;
    }

    if(NULL == data) // printf("ERROR: input data is NULL\n");
    {
        return -22;
    }

    if (rb->rbHead < rb->rbTail) //尾大于头
    {
        copySz = min(count, rbCanRead(rb)); //查看能读的个数
        memcpy(data, rb->rbHead, copySz); //读出数据到data
        rb->rbHead += copySz; //头指针加上读取的个数
        return copySz; //返回读取的个数
    }
    else //头大于等于了尾
    {
        if (count < rbCapacity(rb) - (rb->rbHead - rb->rbBuff)) //读的个数小于头上面
        {
            copySz = count; //读出的个数
            memcpy(data, rb->rbHead, copySz); //
            rb->rbHead += copySz;
            return copySz;
        }
    }
}

```

```

    }
    else//读的个数大于头上面的数据量
    {
        copySz = rbCapacity(rb) - (rb->rbHead - rb->rbBuff);//先读出来头上面
        memcpy(data, rb->rbHead, copySz);
        rb->rbHead = rb->rbBuff;//头指针指向数组的首地址

        //还要读的个数
        copySz += rbRead(rb, (char*)data+copySz, count-copySz);//接着读剩余
        return copySz;
    }
}

int32_t rbWrite(rb_t *rb, const void *data, uint32_t count)
{
    int tailAvailSz = 0;

    if(NULL == rb)
    {
        // printf("ERROR: rb is empty \n");
        return -11;
    }

    if(NULL == data)
    {
        // printf("ERROR: data is empty \n");
        return -12;
    }

    if (count >= rbCanWrite(rb))//如果剩余的空间不够
    {
        // printf("ERROR: no memory \n");
        return -13;
    }

    if (rb->rbHead <= rb->rbTail)//头小于等于尾
    {
        tailAvailSz = rbCapacity(rb) - (rb->rbTail - rb->rbBuff);//查看尾上面剩
        if (count <= tailAvailSz)//个数小于等于尾上面剩余的空间
        {
            memcpy(rb->rbTail, data, count);//拷贝数据到环形数组
            rb->rbTail += count;//尾指针加上数据个数
            if (rb->rbTail == rb->rbBuff+rbCapacity(rb))//正好写到最后
            {
                rb->rbTail = rb->rbBuff;//尾指向数组的首地址
            }
            return count;//返回写入的数据个数
        }
        else
        {
            memcpy(rb->rbTail, data, tailAvailSz);//填入尾上面剩余的空间
            rb->rbTail = rb->rbBuff;//尾指针指向数组首地址
            //剩余空间 剩余数据的首地址 剩余数据的个
            return tailAvailSz + rbWrite(rb, (char*)data+tailAvailSz, count-t
        }
    }
    else //头大于尾
    {
        memcpy(rb->rbTail, data, count);
        rb->rbTail += count;
        return count;
    }
}

```



```

}

/**@} */

/**
 * @brief 往环形队列里面写入数据
 * @param rb 环形队列管理变量
 * @param USARTx 控制打开某个串口发送中断
 * @param EnabledUsart 控制打开中断
 * @param buf 发送的数据
 * @param len 数据长度
 * @retval 负数:错误 正数:写入的数据长度
 * @warning
 * @example
 */
int32_t PutData(rb_t *rb, void *buf, uint32_t len)
{
    int32_t count = 0;

    if(NULL == buf)
    {
        // printf("ERROR: gizPutData buf is empty \n");
        return -1;
    }

    count = rbWrite(rb, buf, len);
    if(count != len)
    {
        //printf("ERROR: Failed to rbWrite \n");
        return -2;
    }
    return count;
}

```



```

#ifndef LOOPLIST_H_
#define LOOPLIST_H_

#ifndef LOOPLIST_C_
#define LOOPLIST_Ex_ extern
#else
#define LOOPLIST_Ex_
#endif

#include <stm32f10x.h>

#define min(a, b) (a)<(b)?(a):(b) //< 获取最小值

/** 环形缓冲区数据结构 */
typedef struct {
    uint32_t rbCapacity; //空间大小
    char *rbHead; //头
    char *rbTail; //尾
    char *rbBuff; //数组的首地址
}rb_t;

```

```

void rbCreate(rb_t *rb, void *Buff, uint32_t BuffLen); //创建或者说初始化环形缓冲区
void rbDelete(rb_t *rb);
int32_t rbCapacity(rb_t *rb); //得到环形大小
int32_t rbCanRead(rb_t *rb); //能读出数据的个数
int32_t rbCanWrite(rb_t *rb); //还剩余的空间
int32_t rbRead(rb_t *rb, void *data, uint32_t count); //读取数据
int32_t rbWrite(rb_t *rb, const void *data, uint32_t count);
int32_t PutData(rb_t *rb, void *buf, uint32_t len);

#endif

```



2.创建

```

main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c timer.c
11 #include "main.h"
12 #include <stdio.h>
13 #include <string.h>
14 #include <stdlib.h>
15 #include "mcu_sys.h"
16 #include "delay.h"
17 #include "usart.h"
18 #include "LoopList.h"
19
20 #define temp_len 10
21 char temp[temp_len]; //定义一个数组
22 rb_t rb_t_temp; //定义一个环形队列管理变量
23
24 int main(void)
25 {
26     NVIC_Configuration();
27     usart_init(115200, 115200, 115200); //串口初始化
28     delay_init();
29
30     //创建环形队列 (把数组交给环形队列管理函数)
31     rbCreate(&rb_t_temp, temp, temp_len);
32
33
34     while(1)
35     {
36
37     }
38 }
39

```

3.通过环形队列函数往数组里面存数据

```

6  /**
7   * @brief  往环形队列里面写入数据
8   * @param  rb      环形队列管理变量
9   * @param  buf      数据地址
10  * @param  len      数据长度
11  * @retval  负数:错误   正数:写入的数据长度
12  * @warning
13  * @example
14  */
15  int32_t PutData(rb_t *rb ,void *buf, uint32_t len)
16  {
17      int32_t count = 0;
18
19      if(NULL == buf)
20      {
21          //      printf("ERROR: gizPutData buf is empty \n");
22          return -1;
23      }
24
25      count = rbWrite(rb, buf, len);
26      if(count != len)
27      {
28          //printf("ERROR: Failed to rbWrite \n");
29          return -2;
30      }
31      return count;
32  }
33

```

```

main.c  startup_stm32f10x_hd.s  LoopList.c  mcu_sys.c  usart.c  timer.c
11  #include "main.h"
12  #include <stdio.h>
13  #include <string.h>
14  #include <stdlib.h>
15  #include "mcu_sys.h"
16  #include "delay.h"
17  #include "usart.h"
18  #include "LoopList.h"
19
20  #define temp_len 10
21  char temp[temp_len]; //定义一个数组
22  rb_t rb_t_temp; //定义一个环形队列管理变量
23
24  char buff[5]={1,2,3,4,5};
25  int main(void)
26  {
27      NVIC_Configuration();
28      usart_init(115200,115200,115200); //串口初始化
29      delay_init();
30
31      //创建环形队列 (把数组交给环形队列管理函数)
32      rbCreate(&rb_t_temp,temp,temp_len);
33
34      //利用环形队列函数把数据存储到数组
35      PutData(&rb_t_temp,buff,5);
36
37      while(1)
38      {
39
40      }
41  }
42

```

main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c

```
16 #include "delay.h"
17 #include "usart.h"
18 #include "LoopList.h"
19
20 #define temp_len 10
21 char temp[temp_len]; //定义一个数组
22 rb_t rb_t_temp; //定义一个环形队列管理变量
23
24 char buff[5]={1,2,3,4,5};
25 int main(void)
26 {
27     NVIC_Configuration();
28     usart_init(115200,115200); //串口初始
29     delay_init();
30
31     //创建环形队列 (把数组交给环形队列管理函数)
32     rbCreate(&rb_t_temp,temp,temp_len);
33
34     //利用环形队列函数把数据存储到数组
35     PutData(&rb_t_temp,buff,5);
36
37     while(1)
38     {
```

Watch 1

Name	Value	Type
temp	0x20000058 temp[] "D...	char[10]
[0]	0x01	char
[1]	0x02	char
[2]	0x03	char
[3]	0x04	char
[4]	0x05	char
[5]	0x00	char
[6]	0x00	char
[7]	0x00	char
[8]	0x00	char
[9]	0x00	char

<Enter expression>

4.通过环形队列函数往数组里面存数据

main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c

```
22 rb_t rb_t_temp; //定义一个环形队列管理变量
23
24 char buff[5]={1,2,3,4,5};
25 char buff1[4]={6,7,8,9};
26 int main(void)
27 {
28     NVIC_Configuration();
29     usart_init(115200,115200); //串口初始
30     delay_init();
31
32     //创建环形队列 (把数组交给环形队列管理函数)
33     rbCreate(&rb_t_temp,temp,temp_len);
34
35     //利用环形队列函数把数据存储到数组
36     PutData(&rb_t_temp,buff,5);
37     //利用环形队列函数把数据存储到数组
38     PutData(&rb_t_temp,buff1,4);
39
40     while(1)
41     {
42     }
43 }
44 }
```

Watch 1

Name	Value	Type
temp	0x2000005C temp[] "D...	char[10]
[0]	0x01	char
[1]	0x02	char
[2]	0x03	char
[3]	0x04	char
[4]	0x05	char
[5]	0x06	char
[6]	0x07	char
[7]	0x08	char
[8]	0x09	char
[9]	0x00	char

<Enter expression>

5.取出来几个数据

```

18 #include "LoopList.h"
19
20 #define temp_len 10
21 char temp[temp_len]; //定义一个数组
22 rb_t rb_t_temp; //定义一个环形队列管理变量
23
24 char buff[5]={1,2,3,4,5};
25 char buff1[4]={6,7,8,9};
26
27 int main(void)
28 {
29     char buff_read[20];
30     NVIC_Configuration();
31     usart_init(115200,115200); //串口初始化
32     delay_init();
33
34     //创建环形队列 (把数组交给环形队列管理函数)
35     rbCreate(&rb_t_temp,temp,temp_len);
36
37     //利用环形队列函数把数据存储在数组
38     PutData(&rb_t_temp,buff,5);
39     //利用环形队列函数把数据存储在数组
40     PutData(&rb_t_temp,buff1,4);
41
42     /*取出来六个数据*/
43     //一般都是先判断一下里面有多少数据之后再读取
44     if(rbCanRead(&rb_t_temp) >= 6)
45     {
46         rbRead(&rb_t_temp,buff_read,6);
47     }
48
49     printf("\r\n data:%d %d %d %d %d %d",buff_read[0],buff_read[1],buff_read[2],buff_read[3],buff_read[4],buff_read[5]);
50 }

```

ATK XCOM V2.0

```

data:1 2 3 4 5 6

```

咱存储数据的时候存储的顺序是 1,2,3,4,5,6依次存进去的.

取数据的时候也是先取1 然后取2 然后... 最后取6

其实就是先进先出的原则.

6.通过环形队列函数往数组里面存数据

```
main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c timer.c mcu_sys.h LoopList.h
18 #include "LoopList.h"
19
20 #define temp_len 10
21 char temp[temp_len]; //定义一个数组
22 rb_t rb_t_temp; //定义一个环形队列管理变量
23
24 char buff[5]={1,2,3,4,5};
25 char buff1[4]={6,7,8,9};
26 char buff2[3]='A','B','C';
27 int main(void)
28 {
29     char buff_read[20];
30     NVIC_Configuration();
31     usart_init(115200,115200,115200); //串口初始化
32     delay_init();
33
34     //创建环形队列 (把数组交给环形队列管理函数)
35     rbCreate(&rb_t_temp,temp,temp_len);
36
37     //利用环形队列函数把数据存储到数组
38     PutData(&rb_t_temp,buff,5);
39     //利用环形队列函数把数据存储到数组
40     PutData(&rb_t_temp,buff1,4);
41
42     /*取出来六个数据*/
43     //一般都是先判断一下里面有多少数据之后再读取
44     if(rbCanRead(&rb_t_temp) >= 6)
45     {
46         rbRead(&rb_t_temp,buff_read,6);
47     }
48
49     printf("\r\n data:%d %d %d %d %d %d",buff_read[0],buff_read[1],buff_read[2],buff_read[3],buff_read[4],buff_read[5]);
50
51     //利用环形队列函数把数据存储到数组
52     PutData(&rb_t_temp,buff2,3);
53
54     while(1)
55     {
56
57     }
```

```
0x080001FA: NOP
main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c
37 //利用环形队列函数把数据存储到数组
38 PutData(&rb_t_temp,buff,5);
39 //利用环形队列函数把数据存储到数组
40 PutData(&rb_t_temp,buff1,4);
41
42 /*取出来六个数据*/
43 //一般都是先判断一下里面有多少数据之后再读取
44 if(rbCanRead(&rb_t_temp) >= 6)
45 {
46     rbRead(&rb_t_temp,buff_read,6);
47 }
48
49 printf("\r\n data:%d %d %d %d %d %d",buff_read[0],buff_read[1],buff_read[2],buff_read[3],buff_read[4],buff_read[5]);
50
51 //利用环形队列函数把数据存储到数组
52 PutData(&rb_t_temp,buff2,3);
53
54 while(1)
55 {
56
57
58
59 }
```

Name	Value	Type
temp	0x2000005C temp[] 'B...	char[10]
[0]	0x42 'B'	char
[1]	0x43 'C'	char
[2]	0x03	char
[3]	0x04	char
[4]	0x05	char
[5]	0x06	char
[6]	0x07	char
[7]	0x08	char
[8]	0x09	char
[9]	0x41 'A'	char

咱再接着存的时候是不是形成了一个环形的结构了.转着圈的存数据.

注意上面的数组黄框位置,黄框位置咱已经调用了取数据函数把里面的数据读取了.

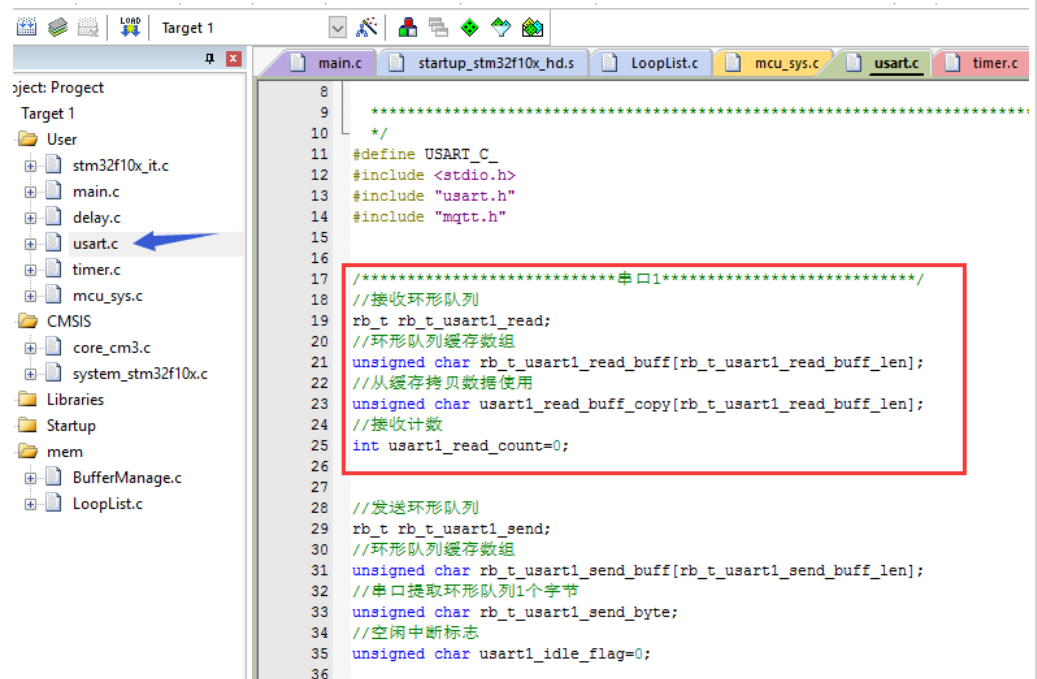
其实黄框位置在环形队列管理函数里面认为是空位置.

现在看典型应用

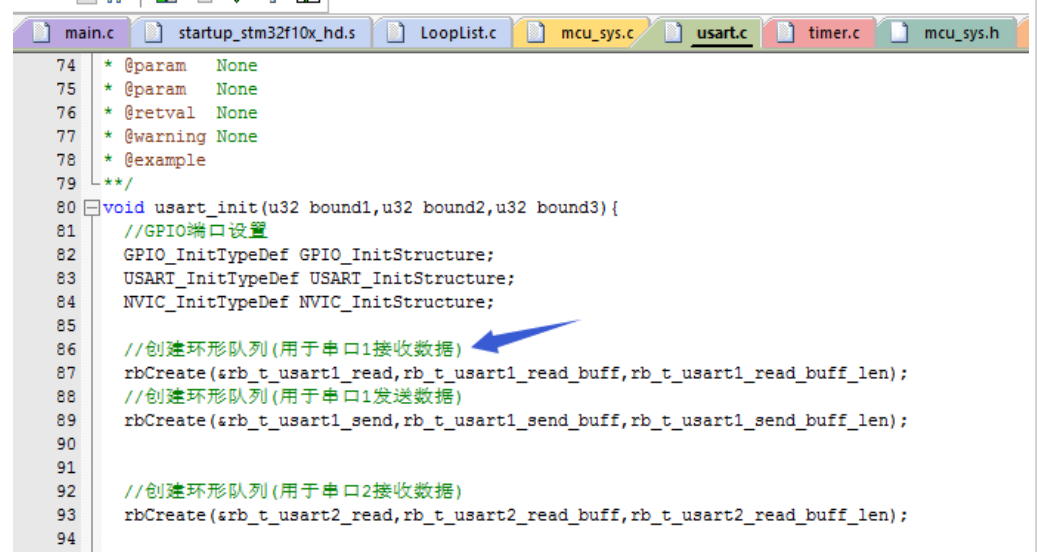
1,说明

首先环形队列适用于很多场合,尤其是一边存数据一边处理数据的场合.

2.使用环形队列缓存串口数据



```
8
9
10  */
11  #define USART_C_
12  #include <stdio.h>
13  #include "usart.h"
14  #include "mqtt.h"
15
16
17  /******串口1******/
18  //接收环形队列
19  rb_t rb_t_usart1_read;
20  //环形队列缓存数组
21  unsigned char rb_t_usart1_read_buff[rb_t_usart1_read_buff_len];
22  //从缓存拷贝数据使用
23  unsigned char usart1_read_buff_copy[rb_t_usart1_read_buff_len];
24  //接收计数
25  int usart1_read_count=0;
26
27
28  //发送环形队列
29  rb_t rb_t_usart1_send;
30  //环形队列缓存数组
31  unsigned char rb_t_usart1_send_buff[rb_t_usart1_send_buff_len];
32  //串口提取环形队列1个字节
33  unsigned char rb_t_usart1_send_byte;
34  //空闲中断标志
35  unsigned char usart1_idle_flag=0;
36
```



```
74  * @param None
75  * @param None
76  * @retval None
77  * @warning None
78  * @example
79  **/
80  void usart_init(u32 bound1,u32 bound2,u32 bound3){
81  //GPIO端口设置
82  GPIO_InitTypeDef GPIO_InitStructure;
83  USART_InitTypeDef USART_InitStructure;
84  NVIC_InitTypeDef NVIC_InitStructure;
85
86  //创建环形队列(用于串口1接收数据)
87  rbCreate(&rb_t_usart1_read,rb_t_usart1_read_buff,rb_t_usart1_read_buff_len);
88  //创建环形队列(用于串口1发送数据)
89  rbCreate(&rb_t_usart1_send,rb_t_usart1_send_buff,rb_t_usart1_send_buff_len);
90
91
92  //创建环形队列(用于串口2接收数据)
93  rbCreate(&rb_t_usart2_read,rb_t_usart2_read_buff,rb_t_usart2_read_buff_len);
94
95
```

```
main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c time
290 }
291
292
293 //串口中断服务程序
294 void USART1_IRQHandler(void)
295 {
296     u8 Res;
297     int value;
298     if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
299     {
300         Res =USART_ReceiveData(USART1); //读取接收到的数据
301
302         PutData(&rb_t_usart1_read,&Res,1);
303         usart1_read_count++;
304     }
305     else if(USART_GetITStatus(USART1,USART_IT_IDLE) == SET)//空闲中断
306     {
307         USART1->DR; //清除USART_IT_IDLE标志
308         usart1_idle_flag = 1;
309     }
310 }
```

主循环读取缓存的数据,并使用串口1发送出去

```
main.c startup_stm32f10x_hd.s LoopList.c mcu_sys.c usart.c time
10
11 #include "main.h"
12 #include <stdio.h>
13 #include <string.h>
14 #include <stdlib.h>
15 #include "mcu_sys.h"
16 #include "delay.h"
17 #include "usart.h"
18 #include "LoopList.h"
19
20 int main(void)
21 {
22     char data_read;
23     NVIC_Configuration();
24     usart_init(115200,115200,115200); //串口初始化
25     delay_init();
26
27     while(1)
28     {
29         //如果串口缓存里面有数据
30         if(rbCanRead(&rb_t_usart1_read)>0)
31         {
32             //读取一个数据
33             rbRead(&rb_t_usart1_read,&data_read,1);
34             //使用串口1发送出去
35             USART_SendData(USART1, data_read);
36         }
37     }
38 }
```




3.可能用户会想就这?

我的所有的项目都是使用的环形队列做数据处理.

更加典型的应该看下面的链接(里面的代码开源):单片机IAP升级程序

我使用环形队列接收程序文件,定义的数组只用了 5字节

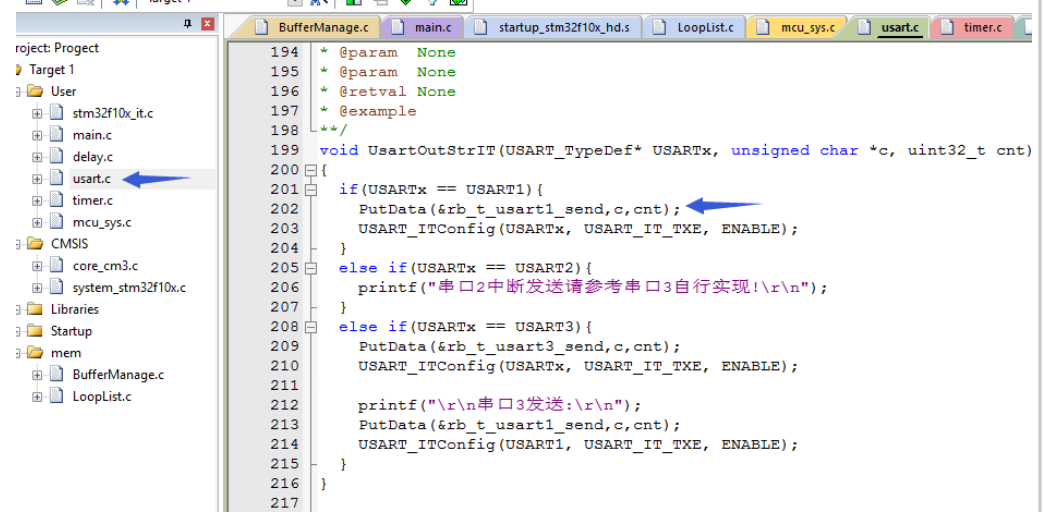
也就是说就用了5字节大小的数组就完成了升级单片机程序

<https://www.cnblogs.com/yangfengwu/p/14620102.html>

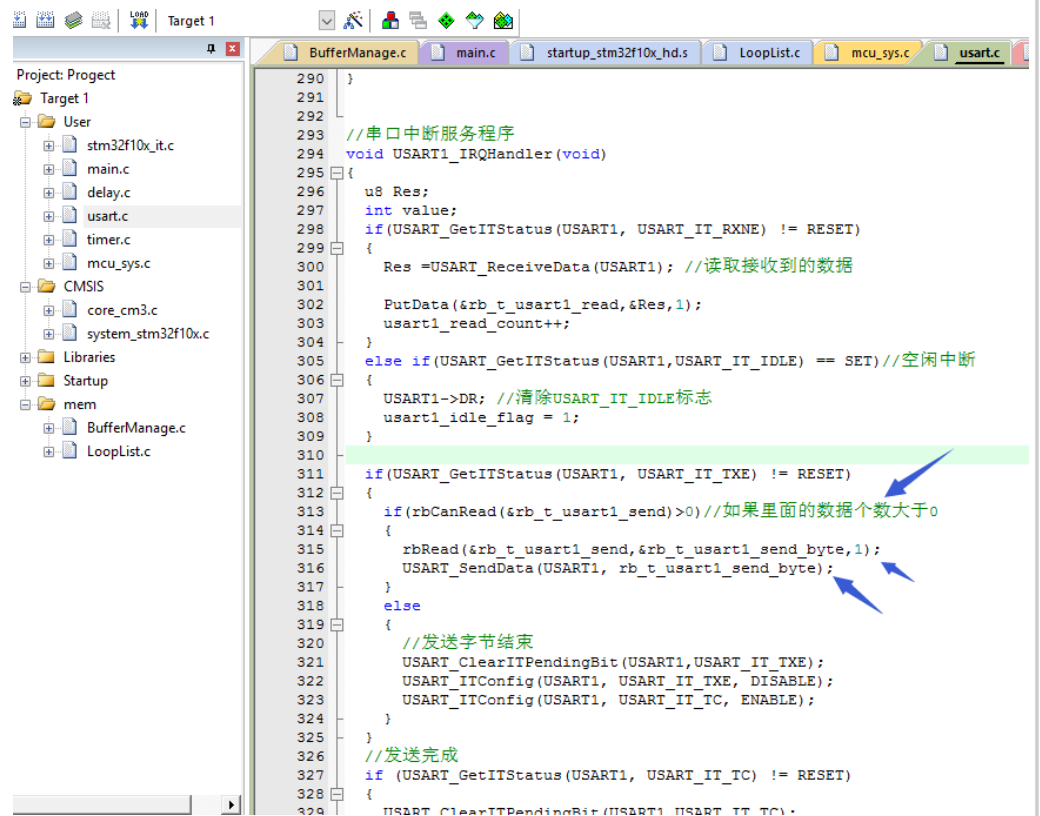
4.用户只需要知道,环形队列就是一个缓存数据的方式

此节代码中还有使用中断发送数据,缓存也是使用的环形队列

其实就是把数据放到环形队列,然后打开中断发送,

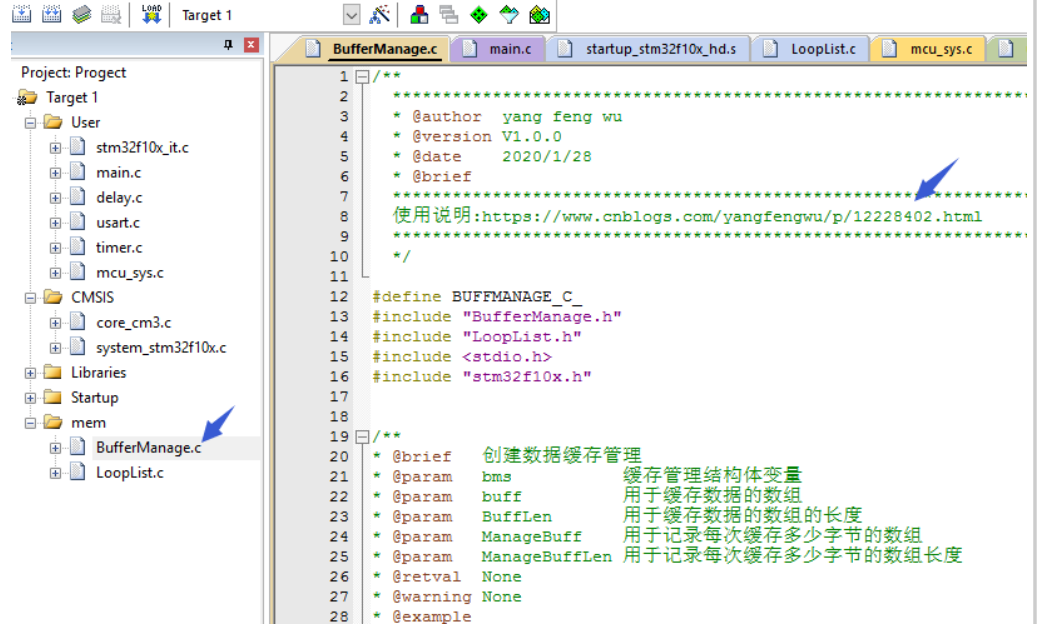


然后在中断函数里面读取数据,发送出去



5.还有我使用环形队列再次封装的一套缓存

<https://www.cnblogs.com/yangfengwu/p/12228402.html>



结语

切莫眼高手低!!!!

分类: [数据处理思想和程序架构](#)



0

0

« 上一篇: [硬件基础知识和典型应用-关于OVXXXX 系列摄像头使用说明](#)

posted on 2021-05-06 14:14 杨奉武 阅读(0) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

[编辑](#) [预览](#)

B [🔗](#) [</>](#) [“”](#) [🖼](#)

支持 Markdown

[提交评论](#)[退出](#)

[Ctrl+Enter快捷键提交]

【推荐】玩转开发板：旧键盘+OpenHarmony 变身蓝牙键盘 v0.1

【推荐】大型组态、工控、仿真、CAD\GIS 50万行VC++源码免费下载!

【推荐】阿里云爆品销量榜单，精选爆款产品低至0.55折

【推荐】限时秒杀！国云大数据魔镜，企业级云分析平台

园子动态：

- 致园友们的一封检讨书：都是我们的错
- 数据库实例 CPU 100% 引发全站故障
- 发起一个开源项目：博客引擎 fluss

最新新闻：

- YouTube Shorts正式登陆美国 和TikTok正面竞争
 - 报道称索尼已放弃数码单反产品线 重心转移至无反相机新品
 - 法院裁定Snap可因其速度过滤器助长致命车祸而被起诉
 - 苹果为一名9岁自闭症患者在App Store的不知情消费退款并道歉
 - Apple Card持卡户数在2020年底增长到约640万
- » 更多新闻...

Powered by:

博客园

Copyright © 2021 杨奉武

Powered by .NET 5.0 on Kubernetes



单片机,物联网,上位机,...

扫一扫二维码, 加入群聊。