创e联

CREELINKS

——数据打包及解包操作

版本：V1.0

日期：2017-04-14

编写：北京大信科技有限公司

在通过串口、WIFI、蓝牙、2.4G 射频等等需要数据传输的场合，我们常会定义两者之间的通讯协议，如帧头+包长+数据+校验合等等。

但是我们经常作用这种方式进行数据打包及解包：

```
sendBuf[0] = val & 0xFF;
sendBuf[1] = (val >> 8) & 0xFF;
```

这种会导致一个弊端，就是当添加或删除一个变量时，就需要修改协议结构，有时候修改过程中，还总是会出 BUG，有没有一种方法或函数，能够将一堆数据打包成 byte 数组，并能够将 byte 数组解包成一堆数据呢？并且添加删除一些参数时，能够保证不用修改太多的代码？

在设计 CREELINKS 小四轴无人机数据通讯过程中，考虑到模块化、对象化的设计思想，故设计了库 CePackage，用于解决以上问题。查看代码：

CePackage.h：

```
#ifndef __CE_PACKAGE_H__
#define __CE_PACKAGE_H__
#ifdef __cplusplus
  extern "C" {
#endif //__cplusplus
#include "Creelinks.h"

#define    CE_PACKAGE_PACK_SIZE        32    /*!<数据传输模块发送一次包的长度 */

#define                        CE_PACKAGE_SEND_BUF_SIZE                        (CE_PACKAGE_PACK_SIZE                        *
(sizeof(CePackageRecv)/sizeof(uint32)*4/(CE_PACKAGE_PACK_SIZE-6)                                                        +
((sizeof(CePackageRecv)/sizeof(uint32)*4)%(CE_PACKAGE_PACK_SIZE-6)  ==  0  ?0:1)))    /*!<  根据发送结构体中的参数个数和包头等信息，计算出发送缓存长度*/
#define                        CE_PACKAGE_RECV_BUF_SIZE                        (CE_PACKAGE_PACK_SIZE                        *
(sizeof(CePackageRecv)/sizeof(uint32)*4/(CE_PACKAGE_PACK_SIZE-6)                                                        +
((sizeof(CePackageRecv)/sizeof(uint32)*4)%(CE_PACKAGE_PACK_SIZE-6)  ==  0  ?0:1)))    /*!<  根据接收结构体中的参数个数和包头等信息，计算出接收缓存长度*/

/**
   * @brief   结构体，需要发送的数据
   */
typedef struct
{
      int32 sendVal1;                /*!<需要接收的参数 1*/
      int32 sendVal2;                /*!<需要接收的参数 2*/
        int32 sendVal3;                  /*!<需要接收的参数 3*/
       //直接添加其它参数
}CePackageSend;
/**
   * @brief   结构体，需要接收的数据
   */
typedef struct
{
      int32 recvVal1;                /*!<需要发送的参数 1*/
      int32 recvVal2;                /*!<需要发送的参数 2*/
        int32 recvVal3;                  /*!<需要发送的参数 3*/
       //直接添加其它参数
}CePackageRecv;
/*
  *CePackage 操作对像
  */
typedef struct
{

      /*!< @brief cePackageSend 模块初始化，对结构体中的数据进行清 0 操作*/
      CE_STATUS     (*initialSend)(CePackageSend* cePackageSend);

      /*!< @brief cePackageRecv 模块初始化，对结构体中的数据进行清 0 操作*/
      CE_STATUS     (*initialRecv)(CePackageRecv* cePackageRecv);

      /*!< @brief  对 cePackageSend 结构体进行打包，反回打包后可直接发送 byte 数组*/
      uint8*              (*dealSend)(CePackageSend* cePackageSend);

      /*!< @brief  对 recvBuf 中的数据进行拆包解析处理，解析后的结果更新到结构体 cePackageRecv */
      CE_STATUS     (*dealRecv)(CePackageRecv* cePackageRecv, uint8* recvBuf,uint16 recvCount);
}CePackageOp;
```

```
/*
  *CePackage 操作对象实例
  */
extern const CePackageOp cePackageOp;

#ifdef __cplusplus
  }
#endif //__cplusplus
#endif //__CE_PACKAGEH__
```

CePackage.c

```
#include "CePackage.h"
#include "CeTMU.h"
#ifdef __cplusplus
extern "C" {
#endif //__cplusplus

uint8 cePackageSendBuf[CE_PACKAGE_SEND_BUF_SIZE];
/**
   * @brief    cePackageSend 模块初始化，对结构体中的数据进行清 0 操作
   * @param cePackageSend:CePackageSend 属性对象指针
   */
CE_STATUS      cePackage_initialSend(CePackageSend* cePackageSend)
{
      int i;
      int32* temp = (int32*)(cePackageSend);
      for(i=0;i<sizeof(CePackageRecv)/sizeof(uint32);i++)
      {
            *temp = 0;
            temp++;
      }

      for(i=0;i<CE_PACKAGE_SEND_BUF_SIZE;i++)
            cePackageSendBuf[i] = 0x00;

      return CE_STATUS_SUCCESS;
}
/**
   * @brief cePackageRecv 模块初始化，对结构体中的数据进行清 0 操作
   * @param cePackageRecv:CePackageRecv 属性对象指针
   */
CE_STATUS      cePackage_initialRecv(CePackageRecv* cePackageRecv)
{
      int i;
      int32* temp = (int32*)(cePackageRecv);
      for(i=0;i<sizeof(CePackageRecv)/sizeof(uint32);i++)
      {
            *temp = 0;
            temp++;
      }
      return CE_STATUS_SUCCESS;
}
/**
   * @brief  对 cePackageSend 结构体进行打包，反回打包后可直接发送 byte 数组
   * @param cePackageSend:CePackageSend 属性对象指针
   * @return  打包后可直接发送的 byte 数组，长度为 CE_PACKAGE_SEND_BUF_SIZE
   */
uint8*    cePackage_dealSend(CePackageSend* cePackageSend)
{
      int i,j;
      int32* temp = (int32*)(cePackageSend);
      int32 structIndex = 0;
      uint8 sum=0;

      //for(j=0;j< (CE_PACKAGE_SEND_BUF_SIZE / CE_PACKAGE_PACK_SIZE);j++)
      for(j=0;j< 1;j++)//由于遥控器无需发送调试参数，故只需发送 1 包数据即可
      {
            cePackageSendBuf[0+j*CE_PACKAGE_PACK_SIZE] = 0x55;    //帧头 0x55
            cePackageSendBuf[1+j*CE_PACKAGE_PACK_SIZE] = j;          //第 N 包
            sum = 0x55+j;
            for(i=2;i< CE_PACKAGE_PACK_SIZE-2;i+=4)
            {
                  if(structIndex >=    sizeof(CePackageRecv)/sizeof(uint32))
                        break;

                  cePackageSendBuf[i+0+j*CE_PACKAGE_PACK_SIZE] = (*temp)&0xFF;
```

```
                cePackageSendBuf[i+1+j*CE_PACKAGE_PACK_SIZE] = ((*temp) >> 8)&0xFF;
                cePackageSendBuf[i+2+j*CE_PACKAGE_PACK_SIZE] = ((*temp) >> 16)&0xFF;
                cePackageSendBuf[i+3+j*CE_PACKAGE_PACK_SIZE] = ((*temp) >> 24)&0xFF;
                temp++;
                structIndex++;
                sum = (sum + cePackageSendBuf[i+j*CE_PACKAGE_PACK_SIZE] +
cePackageSendBuf[i+1+j*CE_PACKAGE_PACK_SIZE]) +
cePackageSendBuf[i+2+j*CE_PACKAGE_PACK_SIZE]+cePackageSendBuf[i+3+j*CE_PACKAGE_PACK_SIZE] ;//检验合累加
            }
            cePackageSendBuf[CE_PACKAGE_PACK_SIZE+j*CE_PACKAGE_PACK_SIZE-2] = sum;//校验数据的正确性
            cePackageSendBuf[CE_PACKAGE_PACK_SIZE+j*CE_PACKAGE_PACK_SIZE-1] = 0xFE;//帧尾
        }
        return cePackageSendBuf;
}
/**
  * @brief  对 recvBuf 中的数据进行拆包解析处理，解析后的结果更新到结构体 cePackageRecv
  * @param cePackageRecv:CePackageRecv 属性对象指针
  * @param recvBuf:接收数据缓存地址
  * @param recvCount:接收数据缓存长度
  * @return  返回 CE_STATUS_SUCCESS 则解析成功，否则失败
  */
CE_STATUS       cePackage_dealRecv(CePackageRecv* cePackageRecv, uint8* recvBuf, uint16 recvCount)
{
        int i=0,j;
        int packIndex = 0;
        int32 structIndex = 0;
        int32* temp = (int32*)(cePackageRecv);
        uint8 sum=0;

        if( recvCount % CE_PACKAGE_PACK_SIZE != 0 )
                return CE_STATUS_FAILE;

        for(j=0;j< recvCount/CE_PACKAGE_PACK_SIZE;j++)
        {
                if(recvBuf[0+j*CE_PACKAGE_PACK_SIZE] != 0x55 ||
recvBuf[CE_PACKAGE_PACK_SIZE-1+j*CE_PACKAGE_PACK_SIZE] != 0xFE)//检测帧头与帧尾
                        return CE_STATUS_FAILE;//帧头不符合，直接返回，不进行处理理

                sum = 0;
                for(i=j*CE_PACKAGE_PACK_SIZE;i<j*CE_PACKAGE_PACK_SIZE+CE_PACKAGE_PACK_SIZE-2;i++)//检验合检查
                        sum += recvBuf[i];

                if(sum != recvBuf[j*CE_PACKAGE_PACK_SIZE+CE_PACKAGE_PACK_SIZE-2])
                        return CE_STATUS_FAILE;//校验合不符合，直接返回，不进行处理

                packIndex = recvBuf[j*CE_PACKAGE_PACK_SIZE+1];

                if(packIndex >= CE_PACKAGE_RECV_BUF_SIZE/CE_PACKAGE_PACK_SIZE)
                        break;

                temp = (int32*)(cePackageRecv) + packIndex * (CE_PACKAGE_PACK_SIZE-4)/4;

                structIndex = packIndex * (CE_PACKAGE_PACK_SIZE-4)/4;

                for(i=j*CE_PACKAGE_PACK_SIZE+2;i<j*CE_PACKAGE_PACK_SIZE+CE_PACKAGE_PACK_SIZE-2;i+=4)
                {
                        if(structIndex > sizeof(CePackageRecv)/sizeof(uint32)-1)
                                break;

                        if(temp >= (int32*)(cePackageRecv) + sizeof(CePackageRecv)/sizeof(uint32))
                                break;

                        *temp = (int32)(((0x000000FF & recvBuf[i+3]) << 24) |((0x000000FF & recvBuf[i+2]) << 16) |((0x000000FF &
recvBuf[i+1]) << 8) | (0x000000FF & recvBuf[i]));
                        temp++;
                        structIndex++;
                }
        }
        return CE_STATUS_SUCCESS;
        }
/**
  * @brief    CePackage 模块操作对象定义
  */
const CePackageOp cePackageOp = {cePackage_initialSend,cePackage_initialRecv,cePackage_dealSend,cePackage_dealRecv};


#ifdef __cplusplus
}
#endif //__cplusplus
```

使用方法：

根据自己的需求，先定义包长度：在 CePackage.h 文件中，配置 CE_PACKAGE_PACK_SIZE 的长度，即包长度，其它参数不需要修改。如果参数打包完成后长度小于 CE_PACKAGE_PACK_SIZE-4，则其它部分补 0，如果大于 CE_PACKAGE_PACK_SIZE－4，则打成两个包。

再定义发送端参数：在 CePackageSend 与 CePackageRecv 两个结构体中，添加自己的需要发送和接收的参数。同时定义接收端：将发送端 CePackageSend 与 CePackageRecv 两个结构体中参数互相换一下即可。

注意：添加的所有参数类型一定要为 int32 型，也就是说必须占 4 个字节，不然拆包过程中会出现错误。如果代码中用到了多个 float 型，可放大 1000 位后，再转换为 int32 型。

在你的 main 函数中添加如下代码即可：

```
//定义发送及接收结构体对象
CePackageSend    mySend;
CePackageRecv    myRecv;
//初始化发送及接收结构体对象
cePackageOp.initialSend(&mySend);
cePackageOp.initialRecv(&myRecv);
//将 mySend 发送结构体中的参数进行打包：帧头(0x55)+包序号+数据+校验合+帧尾(0xFE)。打包完成后，直接返回发送 byte 数据缓存的地址。
sendBuf    = cePackageOp.dealSend(&mySend);
//当接收到数据后，进行传入接收 byte 数组缓存，并提供接收数据的长度，即可解释包数据，并把解释的新数据更新到 myRecv 中。
cePackageOp.dealRercv(&myRecv, recvBuf, recvBufCount);
```

驱动下载地址：

CREELINKS 开源载人机源代码完整下载：可直接访问 http://www.creelinks.com/uav 下载完整的 CREELINKS 无源小四轴无人机源代码，内包含最新的 CePackage 驱动文件。