[避免全局变量漫天飞 - 寻觅左岸 - 博客园 (cnblogs.com)]

先谈谈全局变量的特点

全局变量（Global Variables）：在计算机编程语言中，所谓全局变量是指具有全局作用域的变量，这意味着它在整个程序中是可见的，因此是可访问的。所谓可访问，是指全局可读、全局可写。在编译语言中，全局变量通常是静态变量，其范围(生命周期)是程序的整个运行时。**当然解释性语言除外，解释性语言包括命令行解释器（比如python, Java script,shell等）中，全局变量通常在声明时由解释器动态分配，这是由于解释性语言是读取>解释>执行模式，不像编译性语言，运行前可预知变量属性，解释性语言读取解释前无从获取变量属性。**

在C/C++编程语言中，全局变量的这种全局可见性特点，滥用全局变量会让代码表现当相当邪恶！如果使用全局变量，就意味着下面这些场景的存在：

- 实际代码可能有很多地方在读、在写全局变量
- 全局变量在多线程或多任务间共享
- 全局变量在常规代码和中断服务程序间共享

为啥说全局变量很邪恶？

单片机裸机编程

或许你会说，我就这样用？咋了？软件也跑的很好啊？来看看这个场景：

> 一个超字宽的变量（比如16位单片机，字宽即为16位），正被一个常规代码在写变量数据域时且还没写完，啪叽，来了个中断！中断一来，CPU赶紧把手里的活儿停下来，奔过去处理中断了，不巧在中断函数里，该变量因业务需求有需要写这个变量有经验的不这么写，仅为了方便说明：

举个栗子,还是以之前文章的传感器为例，实际应用中传感器可能是下面这样的数据结构来描述：

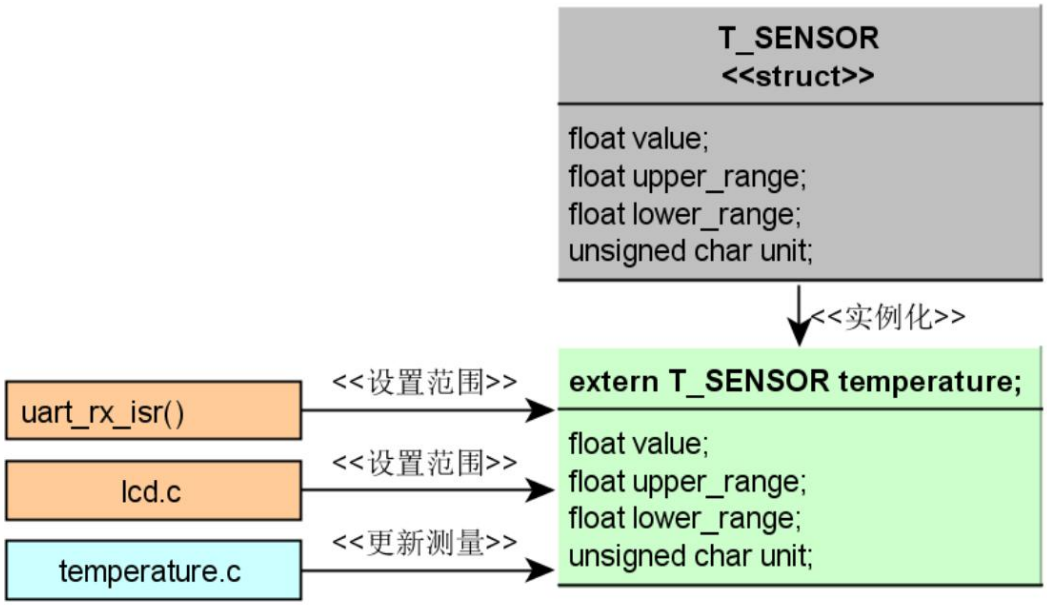
```
#ifndef _SENSOR_H_
#define _SENSOR_H_
typedef struct _t_sensor{
   /* 测量值与测量范围及单位有关 */
   float value;

   /* 测量范围，根据采样值映射  */
   float upper_range;
   float lower_range;
   /* 温度单位 */
   unsiged char unit;
}T_SENSOR;
/*假定是一个温度测量产品*/
extern T_SENSOR temperature;

#endif _SENSOR_H_
```

假定这个传感器数据结构有这样一些被访问的可能：

1. 上位机会改写测量数据的范围及单位，串口通信中断服务程序直接写这个全局变量中的上下限数据域
2. LCD操作界面可改写温度上下限范围。
3. 测量更新模块根据当前范围及单位配置，将传感器采集到的数据映射为测量值。

这些需求用例，用图描述一下：

这些需求用例，用图描述一下：



比如用户操作HMI界面正改写温度范围，而此时远程上位机也正改写温度范围，按上面这个做法，可能出现哪些邪恶的后果呢?

1. 通过LCD界面写入上限为300.5（假定原下限为0），此时远程串口报文收到，程序直接在中断服务程序将范围修改为（-100，200.5），此时中断返回，用户可能接着修改下限为-200，则最终设备内的温度范围可能既不是（-100，200.5）也不是（-200，300.5），而可能是（-200，200.5）。这是一个易理解的数据混乱的场景。
2. 现实中如果使用的单片机是8位/16位单片机，一条指令无法完成操作一个32位立即数，有可能才完成一个浮点数中某几个字节，此时就被中断打断写入200，然后中断返回后继续写入剩下字节，数据可能会变得非常诡异！利用http://www.speedfly.cn/tools/hexconvert/ 在线工具转换浮点数到16进制：

```
0x43964000 /* 浮点数300.5的16进制*/
0x43488000 /* 浮点数200.5的16进制*/
```

假定中断进入时，HMI界面程序写入了0x4396前两个字节，中断返回时，上限改写为200.5(0x43488000),此时继续执行后面两个字节写入，则上限变成为(0x43484000),来看看这个数是多大？变成了200.25，这是不是很邪恶?

或许有的朋友会说，可以在LCD写范围时关中断嘛。诚然，可以这么做：

```
void hmi_operate()
{
    /*关中断*/
    _disable_interrupt();
    /*改写温度范围*/
    ....
    /*开中断*/
    _enable_interrupt();
}
```

但是如果这个全局变量有很多地方在改写，为了数据安全，势必就到处开/关中断，这样做的坏处：

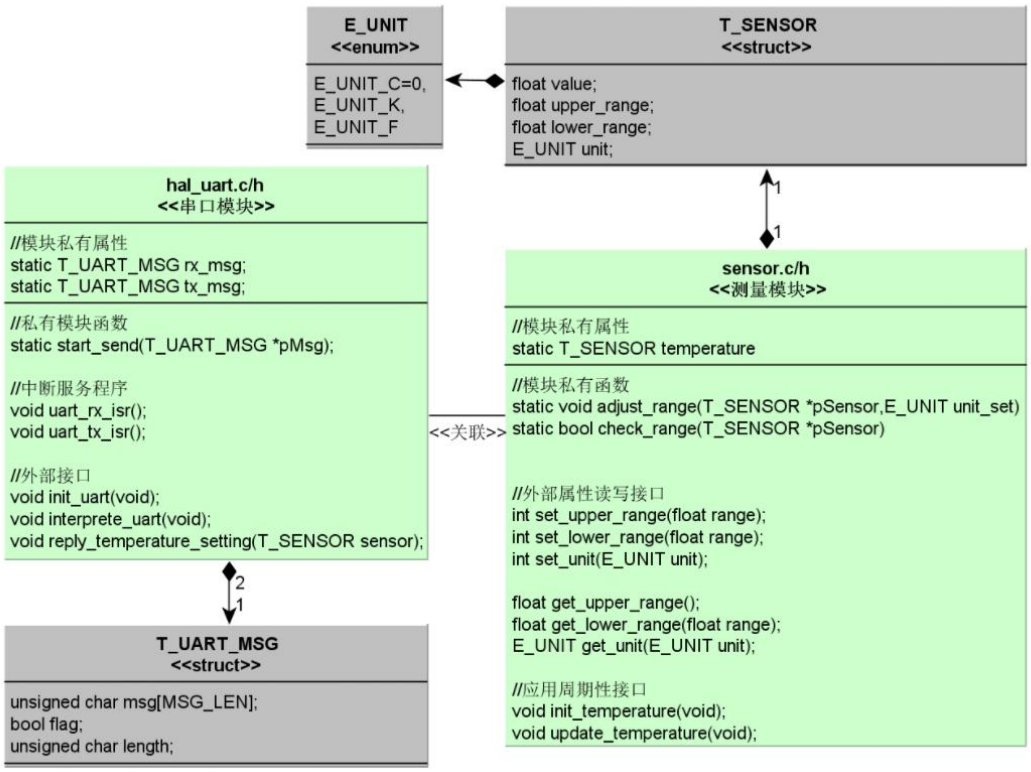
- 经常开关中断，势必影响中断响应，会有概率丢失异步中断处理(比如串口按字节接收中断，可能就会漏收字节)，程序不健壮，工作不稳定。
- 到处访问改写，不易调试，群魔乱舞，代码也不易维护。想加点东西，改点东西可能哪处都是坑，一不小心就掉坑里去了！
- 初学者甚至不会用struct将相关的数据包在一起，其结果是代码里到处都是基本类型的全局变量。一些简单的业务逻辑实现变成一个复杂的代码，数据信息流向一团乱麻。

3. Re:lir

参考资料

裸机程序策略

对于上面这样一个应用场景，怎么解决这种混乱的现象呢。这里分享一下我的思路，这里将主要的串口以及测量模块的设计思路用UML图描述一下大体思路：



如此一来，外部就看不到全局变量了，只需要调用对应的set/get方法即可实现读写访问，由于是裸机前后台程序，数据流向就变的非常清晰了。main函数的主循环大致就可能是这样：

```
void main(void)
{
   /*模块初始化*/
   init_uart();
   init_temperature();
   ....

   while(1)
   {
       interprete_uart();
       /*可能是周期性调用*/
       if(timer_100ms)
       {
          timer_100ms = 0;
          update_temperature();
       }

       ....
   }
}
```

那么uart协议解析要怎么做呢?

```
void interprete_uart(void)
{
    if(rx_msg.flag)
    {
        rx_msg.flag = false;
        /*报文完整性检查*/
        ...

        /*设置温度配置*/
        set_upper_range(xxx);
        set_lower_range(xxx);
        set_unit(xxx);
    }

    if(tx_msg.flag)
    {
        tx_msg.flag = false;
        start_send();
    }
}

static start_send(T_UART_MSG *pMsg)
{
    /*负责底层操作，启动中断传输*/
}
```

```
/*提供应答数据接口*/
void reply_temperature_setting(T_SENSOR sensor)
{
    /*解析传入参数并封装应答报文*/
}
```

如此一来，数据流向将变得很清晰，串口接收到数据更新范围配置时，也无需开关中断了，从应用角度几乎见不到全局变量。当然这样做的代价就是会增加一些栈开销。但是这种代价还是值得的。

对于测量模块的set函数思路稍做说明：

```
int set_upper_range(float range)
{
    T_SENSOR temp = temperature;
    temp.upper_range = range;
    /*实现范围合理性检查*/
    if(check_range(temp))
    {
        /*两个结构体变量可以直接赋值*/
        temperature = temp;
        return 0;
    }
    else
    {
       return -1;
    }
}

int set_unit(E_UNIT unit)
{
    if(unit>E_UNIT_F)
        return -1;
    adjust_range(&temperature,unit);
    temperature.unit = unit;
}
```

上述代码旨在分享个人的一些思路，其中或有不够严谨的地方，但通过这样的设计思路，应能大幅度远离满天飞的全局变量。

多任务/多线程环境

上面描述其实本质上描述了裸机程序里，普通模式运行程序与中断服务程序对于临界资源的竞争。事实上现在不管是单片机，还是处理器，大多都是基于一个操作系统进行应用开发。甚至还可能是多核芯片，这里就存在并发竞争访问资源的问题。

> **临界资源**：各任务/线程采取互斥的方式，实现共享的资源称作临界资源。属于临界资源的硬件串口打印、显示等,软件有消息缓冲队列、变量、数组、缓冲区等。多任务/线程间应采取互斥方式，从而实现对这种资源的共享。

多任务/多线程情况下在写模块时，只需要封装进保护机制即可。常见的保护机制有关中断、信号量、互斥锁等。在Linux内核中为应对多核并发访问还有自旋锁机制。由于篇幅所限，本文就不做展开了，先挖个坑，以后有机会再分享吧。