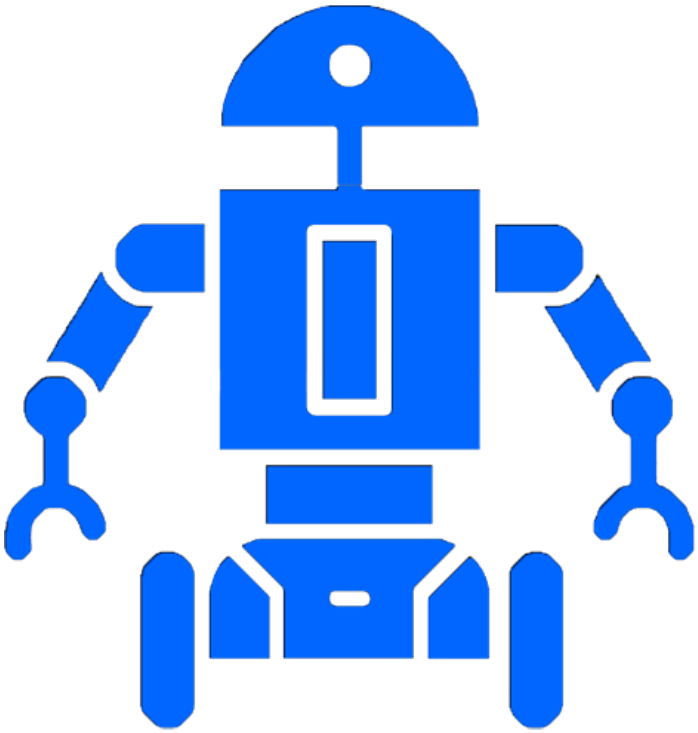
全网最实用的 STM32 和 ROS 机器人的串口通信方案

```
/*--------------------------------发送协议--------------------------------
//----------------55 aa size 00 00 00 00 00 crc8 0d 0a----------------------
//数据头55aa + 数据字节数size + 数据（利用共用体） + 校验crc8 + 数据尾0d0a
//注意：这里数据中预留了一个字节的控制位，其他的可以自行扩展，更改size和数据
------------------------------------------------------------------------*/

/*--------------------------------接收协议--------------------------------
//----------------55 aa size 00 00 00 00 00 crc8 0d 0a----------------------
//数据头55aa + 数据字节数size + 数据（利用共用体） + 校验crc8 + 数据尾0d0a
//注意：这里数据中预留了一个字节的控制位，其他的可以自行扩展，更改size和数据
------------------------------------------------------------------------*/
```

源码文件和文档电子版获取方式：进入下面公众号，发送：串口通信。

下面说明，和具体的程序代码有些出入，但是完全不影响理解。



本方案解决的问题：解决以 STM32 做 ROS 机器人底层驱动的串口通信问题。

为什么要写本次博客？：

最近发现越来越多的小伙伴走入 ROS 机器人的领域，而 ROS 机器人与底层驱动的串口通信问题，是大家学习路上的一个难题。很多小伙伴对 STM32 单片机并不熟悉，对串口通信的理解并不透彻，自己去解决这个问题，费时费力，最后也可能没有好的结果，并且这又不是大多数学习 ROS 机器人的重点。最后发现网上也没有很好的教程（也可能是我没找到），所以，这里根据本人的开发经历，给大家提供一种高效、稳定、易用的 ROS 机器人与 STM32 串口通信的常用方案。

如果不想了解细节的朋友，可以只看下面方案介绍和方案快速使用部分，关

于方案的原理就不用花时间思考了，但是还是希望大家对细节有敬畏之心。

本方案提供的 API：

STM32 向 ROS 发送左轮实时轮速、右轮实时轮速、实时角度、预留控制。

ROS 向 STM32 发送左轮设定速度、右轮设定速度、预留控制

本方案优势：

经测试，长期稳定

保证数据准确率极高

频率 50HZ 左右

易用，引入相关头文件即可，低耦合

----------华丽丽的分界线----------

方案介绍：

本方案将数据开头加入数据头，数据尾部加入数据循环冗余校验和数据尾，将数据打包发送，确保数据的正确性，避免出现一些无法察觉的问题。同样根据 STM32 和 Linux 系统配备相应的数据解析协议。

本方案 STM32 下位机依托 USART1 编写的收发协议，ROS 上位机依托 boost::asio 编写的收发协议。串口并不一定是串口 1，可以更改，但是需要程序变更一些内容（很容易，程序中有标记共三处）。

本方案巧妙的使用共用体的特性，进行数据解析，（也就是无需使用数据分离技术解析数据）。关于共用体，你只需要知道以下几点：

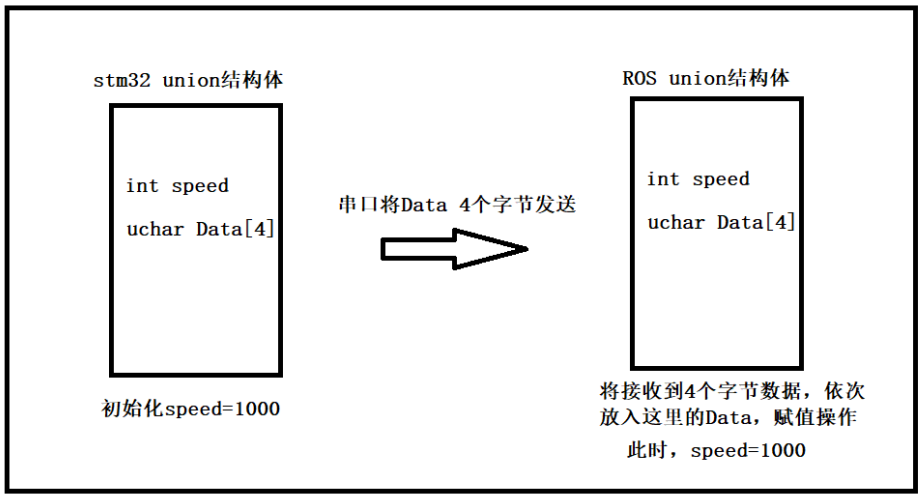
（1） C 语言的一种机制，结构体内不同成员共享内存的机制，（即内存地址一致）

（2） 同一时刻，只能访问其中的一个成员

（3） 不同成员，按照成员类型的性质进行内存访问

不理解的小伙伴，可以看下图，有直观的理解即可。有图有真相

温馨提示：从左向右读图。



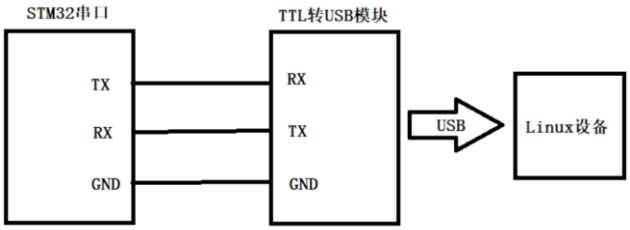
----------华丽丽的分界线----------

方案快速使用：

硬件环境准备：

STM32 串口+TTL 转 USB 模块（CH340）+Linux 硬件设备

线路连接：（有图有真相）



STM32 下位机软件使用介绍：

首先是 STM32 串口参数的配置：（这里配置代码和相关例程都一致）

（1） 波特率=115200

（2） 数据长度=8 位

（3） 停止位=1 个

（4） 奇偶校验位=无

（5） 硬件数据流控制=无

其次是函数使用说明，封装函数如下图：

```
//从linux接收并解析数据到参数地址中
extern int usartReceiveOneData(int *p_leftSpeedSet,int *p_rightSpeedSet,unsigned char *p_crtlFlag);
//封装数据，调用USART1_Send_String将数据发送给linux
extern void usartSendData(short leftVel, short rightVel,short angle,unsigned char ctrlFlag);
//发送指定字符数组的函数
void USART_Send_String(unsigned char *p,unsigned short sendSize);
//计算八位循环冗余校验，得到校验值，一定程度上验证数据的正确性
unsigned char getCrc8(unsigned char *ptr, unsigned short len);
```

这里做简单说明：

函　　数　usartReceiveOneData(int　*p_leftSpeedSet,int *p_rightSpeedSet,unsigned　char　*p_crtlFlag)，填入地址参数作数据获取，使用时放在相应串口的中断服务函数中即可。如这里使用的串口 1，如下图所示：

```
//中断服务函数
void USART1_IRQHandler()
{
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
     {
         USART_ClearITPendingBit(USART1,USART_IT_RXNE);//首先清除中断标志位
         usartReceiveOneData(&leftSpeedSet,&rightSpeedSet,&ctrlFlag);
     }
}
```

函　数　usartSendData(short　leftVel,　short　rightVel,short angle,unsigned char ctrlFlag)，填入需要发送的数据变量作发送，使用时放入指定频率的循环里使用，每次发送一次数据，最好延时 10-15ms，下位机发送和上位机接收都需要时间（时间和串口波特率有关）。我使用的如下图所示：

```
while(1)
{
    //蓝牙调试时用，不调试注释
    //pcShow();
    getAngle();
    //给树莓派发送速度，角度,这里速度已经乘以1000
    usartSendData((short)left_speed,(short)right_speed,(short)yaw,15);
    //发送需要一定的延时
    delay_ms(10);
}
```

其余的两个函数是被上面两个函数调用的，这里就不多说了。

注意：在外部引用函数时，注意引用头文件#include "mbotLinuxUsart.h"

ROS 上位机软件使用介绍：

看图说话，下面共有四个函数，看函数名和参数就可以理解用途。

```
extern void serialInit();
extern void writeSpeed(double RobotV, double YawRate,unsigned char ctrlFlag);
extern bool readSpeed(double &vx,double &vth,double &th,unsigned char &ctrlFlag);
unsigned char getCrc8(unsigned char *ptr, unsigned short len);
```

首先是调用头文件#include "mbot_linux_serial.h"，然后进行初始化串口在程序初始化的时候调用 serialInit()函数，内置串口参数和下位机一致。

然后就是在调用 writeSpeed(double RobotV, double YawRate, unsigned char ctrlFlag)函数，参数是机器人线速度和角速度，也就是/cmd_vel 的数据。将机器人的需要设定的速度下发到下位机。

最后就是调用 readSpeed(double &vx,double &vth,double &th, unsigned char &ctrlFlag)函数，这里使用的引用，输入存放机器人线速度、角速度、角

度的变量即可。

注意：

这里需要两个参数根据自己的机器人进行更改，ROBOT_LENGTH 机器人真实轮间距（从左侧轮子中心到右侧轮子中心的距离），ROBOT_RADIUS 机器人轮间距的一半。

文中 boost::asio::serial_port sp(iosev, "/dev/mbot");的设备名字是我的串口的设备名字，小伙伴可以根据自己的进行更改，例如，/dev/ttyUSB0。

到这里大家肯定都可以愉悦的使用了。如果想知道细节，请往下看。

----------华丽丽的分界线----------

方案的原理解释：

此方案用的是共用体的思路，上面小伙伴们也都对共用有个大致的了解。这是一种按照共用体内成员的数据类型进行内存访问的特性，不同数据类型按照自己的类型访问内存。上位机和下位机的原理是一致的。都定义了数据头、数据尾的常量，和收发共用体。

下位机发送的数据协议：上位机发送的数据协议：

```
/*--------------------------------发送协议-------------------------------------
//----------------55 aa size 00 00 00 00 00 crc8 0d 0a-----------------------
//数据头55aa + 数据字节数size + 数据（利用共用体） + 校验crc8 + 数据尾0d0a
//注意：这里数据中预留了一个字节的控制位，其他的可以自行扩展，更改size和数据
-----------------------------------------------------------------------------*/
```

STM32 的串口接收原理：每接收到一个字节就会触发一次中断，我这里采用在串口的中断服务函数中进行数据接收的解析，具体函数体现在 receiveTo103()根据上位机发送的协议进行判断解析，详见代码，注释清晰。

Linux 上位机采用 ASIO，ASIO 不仅支持网络通信，还能支持串口通信。

这里采用 boost::asio::write(sp, boost::asio::buffer(buf));发送数据

使用 boost::asio::read_until(sp, response, "\r\n",err);

```
    copy(istream_iterator<unsigned
char>(istream(&response)>>noskipws),
          istream_iterator<unsigned char>(),
          buf);
```

获取数据，再具体的细节需要见源码解释了。脑子里的思想就是把相应的数据放到相应的位置，没有数据解析概念，对应的字符数据存好后，就可以通过另

外一个成员访问了。

如果有什么问题可以加我微信讨论：