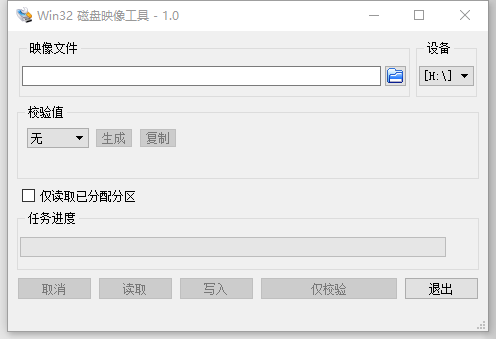
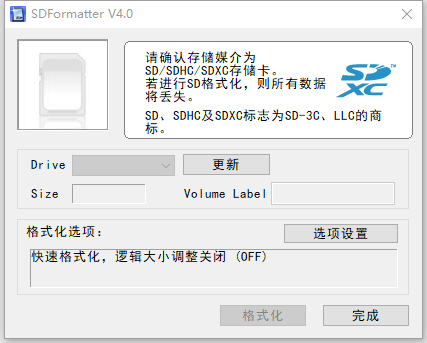
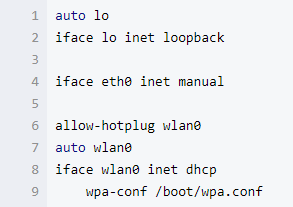
# 树莓派zero w 搭建ROS与基础通讯

## 下载系统

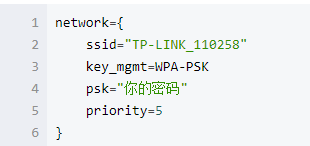
1. 在官网下载最新版本系统 此处下载的为2018-06-27-raspbian-stretch-lite版本，lite为无桌面图象的版本
2. 使用win32磁盘映像工具向SD卡中下载程序，该软件如下
3. 如果需要删除系统，建议使用sdfomatter进行SD卡的格式化，使用windows自带的磁盘工具格式化则需要重新合并，同时可能有其他隐患。当出现SD卡名乱码时，可先重命名后格式化。且第一次格式化基本失败，再次格式化就好。

注：以上步骤均在window系统下进行，完成后转至linux系统下进行系统文件的修改。

1. 系统修改参照链接https://blog.csdn.net/px\_528/article/details/71725251
2. 在sd卡boot目录下新建空白文件ssh（不加任何后缀）
3. 在sd卡etc/network/目录下修改interfaces文件，此处修改需要root权限，使用sudo gedit进行修改，内容如下：

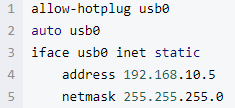


1. 在sd卡的boot/目录下新建wpa.conf文件，内容如下：



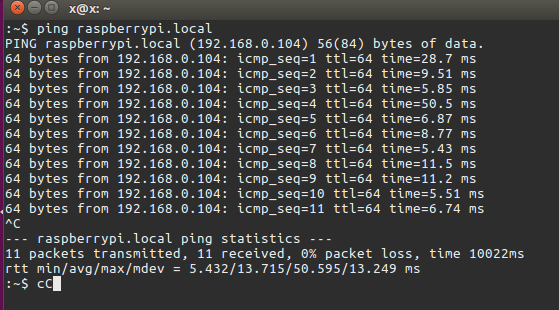
\*注意此处若无其他备用wifi,建议去掉priority这一行，账号密码别错，区分大小写

1. 此处顺便修改USB网卡配置，在sd的boot分区下找到config.txt文件，在最后加一行代码dtoverlay=dwc2
2. 在cmdline.txt中找到rootwait，后面空一格，加上modules-load=dwc2,g\_ether
3. 在sd卡etc/network目录下的interfaces最后面加上一段代码

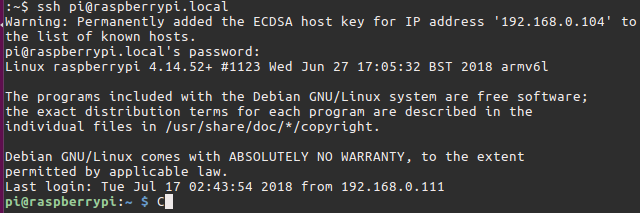


此处使用静态IP，不使用dhcp方式

1. 经过前面的步骤，使用usb连接线连接树莓派板上的PWR口，而不是USB口
2. 若绿灯LED不闪烁，可能问题出在下载的系统上，经测试jessie版本的系统绿灯不亮，而stretch版本绿灯会亮。若绿灯闪烁，则等其稳定（大概1分钟）之后，再进行下一步。
3. 在linux终端使用命令行ping raspberrypi.local 测试连通性（均位于同一子网下），结果如下



1. 连通之后，使用ssh命令连接，ssh [pi@raspberrypi.local](mailto:pi@raspberrypi.local) ，提示输入密码，初始密码为raspberry 结果如下



如果IP改变，则会在ssh连接时提示新的IP，如上图warning

注意：如果之前已经成功ssh连接，而后又重刷了sd卡的系统，则在登陆时会报错，需要删除当前主机linux上~/.ssh下的known\_hosts文件 即rm known\_hosts

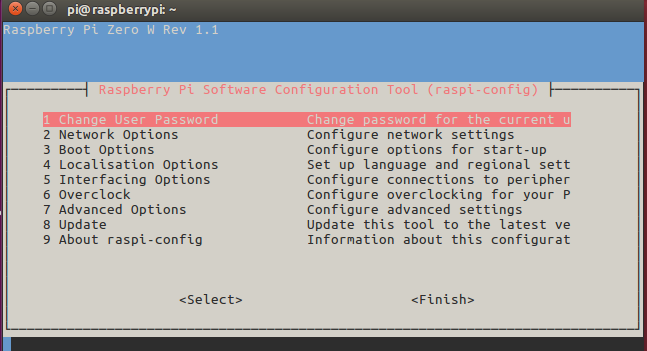
至此，完成了linux系统对树莓派的远程控制。

## 搭建ROS

0.搭建ros之前需要做一些前期工作

1）改密码

输入 sudo raspi-config 可进入系统设置



选择第一项修改密码，改为简单易输入的

1. 更换镜像源

到树莓派 /etc/apt文件下下，使用sudo vim.tiny sources.list 打开文件，修改内部的网址

此处修改为

deb http://mirrors.ustc.edu.cn/raspbian/raspbian/ stretch main contrib non-free rpi

deb-src http://mirrors.ustc.edu.cn/raspbian/raspbian/ stretch main contrib non-free rpi

为中科大的源，其余源可自行寻找

1. 换源后执行

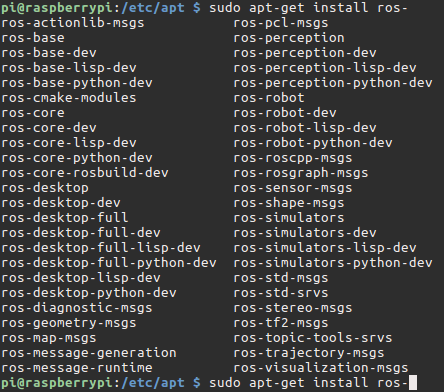
sudo apt-get update

sudo apt-get upgrade

确保整个系统为最新的

1.安装ROS

输入命令行sudo apt-get install ros- 使用tab键可查询到与ros的安装包，如下所示。如果并未出现，则是源和系统的问题，需要多次更新直至有以下选项出现。



1. 此处选择ros-desktop-full 以及 ros-desktop-full-dev安装（下载过程可能出现中断的问题，可以多重复下载几次，如果仍不可以，可以更换新的源，此处的ros版本为debian）
2. 下载之后执行rosdep init 与 rosdep upgrade 不需要添加环境变量
3. 按照以往建立工作空间即可，此处需要在.bashrc中添加工作空间路径
4. 创建工作包的命令有所不同，为

catkin\_creat --rosdistro debian pkg\_name roscpp std\_msgs

1. 创建cpp文件命令为 touch xxx.cpp
2. 编译过程中可能出现有关string类型而不能通过的提示，猜测与gcc版本有关，更新gcc至最新版本即可完成编译
3. 运行多个ros节点时，需要开启多个ssh服务

## 建立通讯

1. 硬件连接

树莓派的硬件引脚图如下

此处的TX RX电压为3.3V

1. 串口软件设置

在linux端可以使用minicom作为串口调试助手

Sudo apt-get install minicom下载

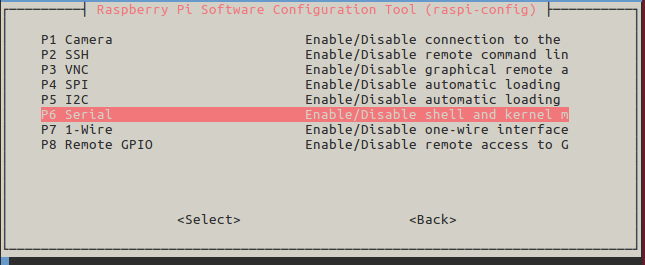
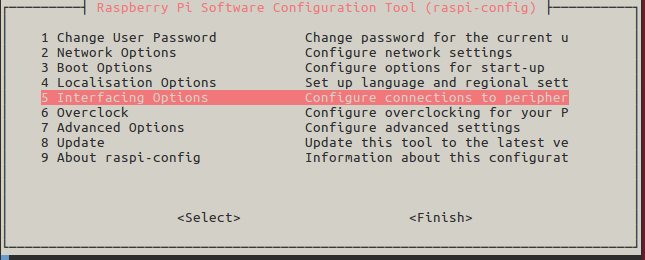
Sudo minicom -s 修改设置 serial port setup 完成后选择save setup as dfl

Sudo minicom 打开minicom

1. 系统软件设置

初始的系统是不可以进行Usart串口通讯,原因是蓝牙、mini串口、硬件串口均使用的是这对TX RX,所以在系统设置中要屏蔽蓝牙以及mini串口的使用，以下的步骤为我尝试过后可以正常通讯的步骤，但是不确定每一步都是必要的。

1. 运行sudo raspi-config 选择interfacing options 选择serial打开使能



1. 下载一个"pi3-miniuart-bt-overlay"文件（自行搜索下载）并将其拷贝至/boot/overlays文件夹中
2. sudo systemctl disable hciuart //关闭蓝牙
3. 修改cmdline.txt文件如下（ttyAMA0是硬件串口名称，而mini串口名称为ttyS0）

dwc\_otg.lpm\_enable=0 console=ttyAMA0,115200 console=tty1 root=PARTUUID=f029fa11-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait modules-load=dwc2,g\_ether

1. 修改config.txt文件如下

dtparam=audio=on

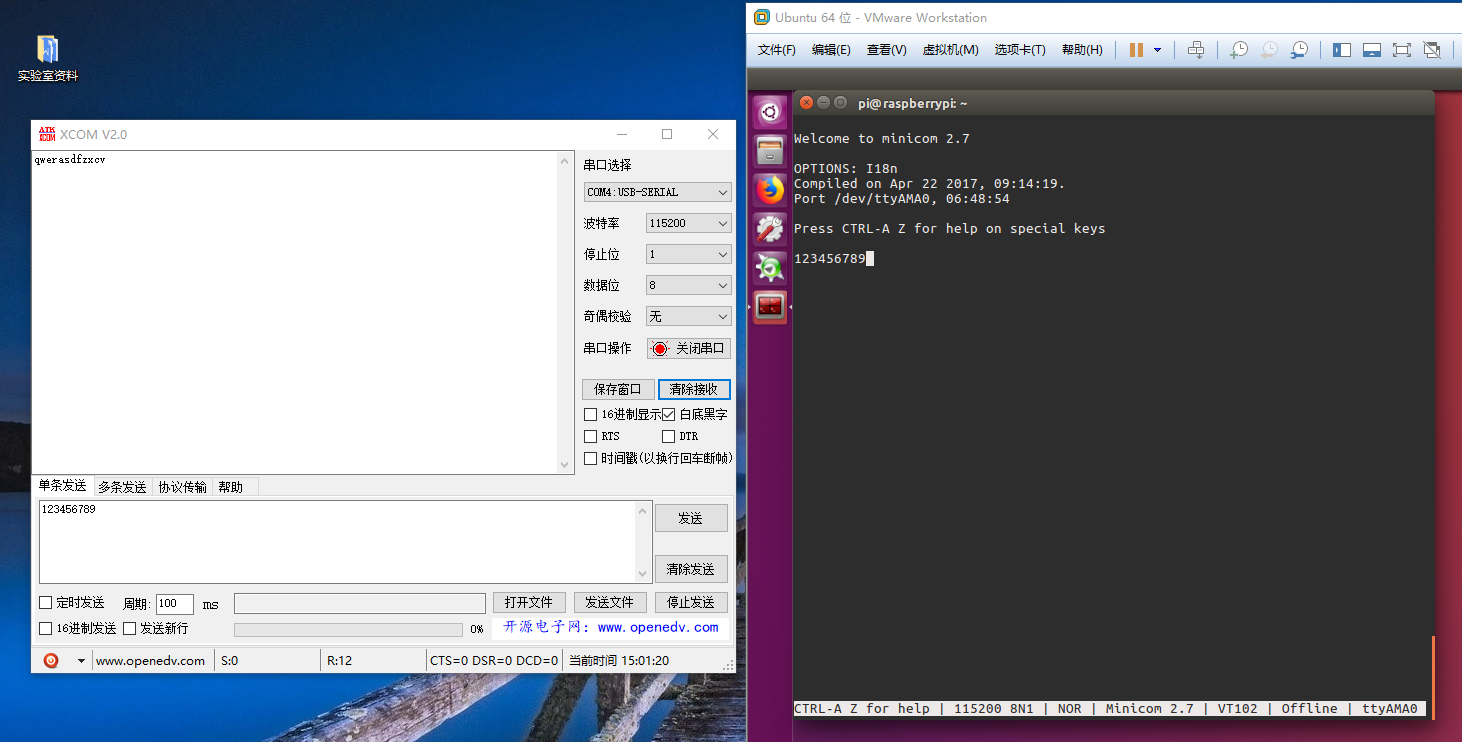
dtoverlay=dwc2

dtoverlay=pi3-miniuart-bt-overlay

enable\_uart=1

6）修改minicom工具的串口名以及波特率等等

7）使用串口转usb（3.3v） 将信息传到windows的串口调试助手上，测试通讯如下



1. 软件程序编写

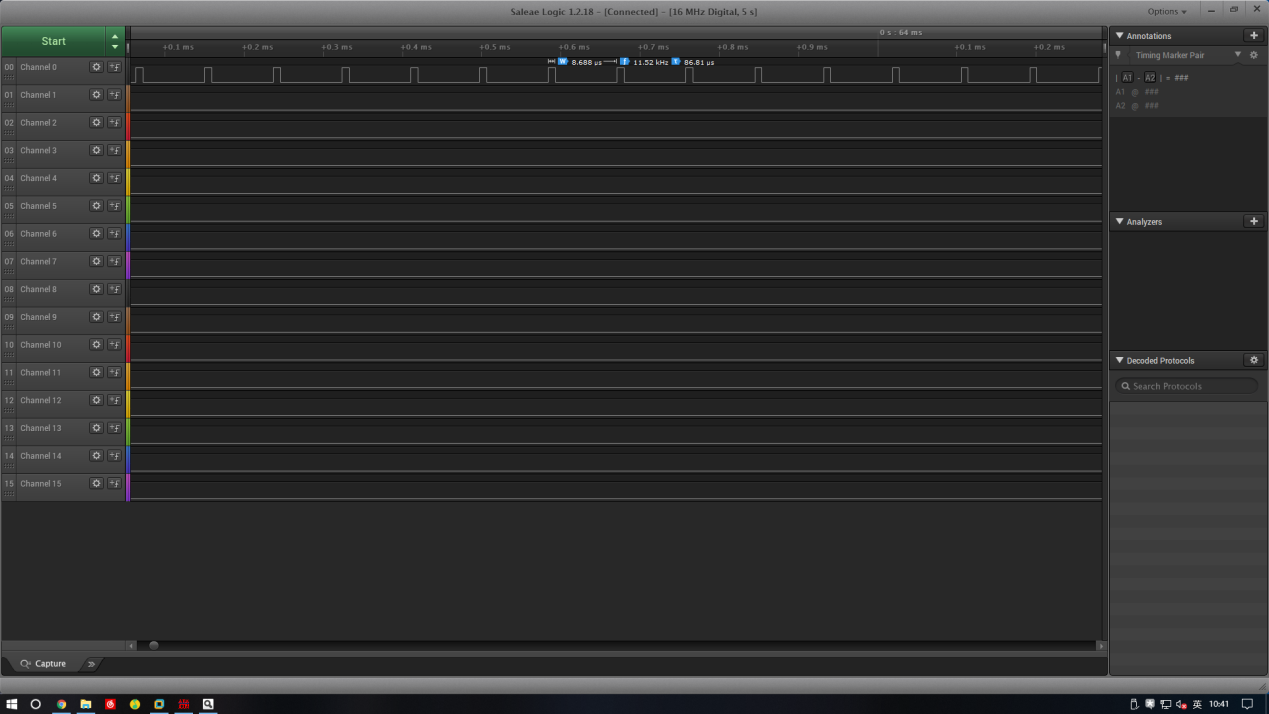
由于是在树莓派上进行编译，所以ros自带的那个串口包并无debian版本，这里采用最为基础的方式编写程序

需要注意：

1. 串口打开时的方式需选用非阻塞的方式，否则会导致程序堵死
2. 程序包含两个包 com 和 rt,其中com为主要的读写串口程序，然后通过话题的形式发布消息，而rt为简单的话题消息发布与订阅，用于测试com的正确性
3. 延时测试

使用逻辑分析仪，修改com函数，注释while内主体，循环发送固定的1024字节数组test,

中间不加任何延时函数，其结果如下：



结果表示，几乎为实时处理

附：com.cpp

#include <sys/types.h> //自带的库

#include <sys/stat.h>

#include <fcntl.h>

#include <termios.h>

#include <stdio.h>

#include <unistd.h>

#include <cstdlib>

#include <ros/ros.h> //ros库

#include <std\_msgs/String.h>

#include <std\_msgs/Empty.h>

#include <string.h>

using namespace std;

//常量定义

#define BAUDRATE B115200

#define UART\_DEVICE "/dev/pts/26"

#define FALSE -1

#define TRUE 0

//变量定义

int fd; //串口句柄

int t\_len,r\_len; //写入\读取字符串长度

string t\_msg; //发送字符串

char t\_buff[256]; //发送数组

char r\_buff[256]; //接收数组

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//回调函数

void write\_callback(const std\_msgs::String::ConstPtr& msg)

{

ROS\_INFO\_STREAM("CALLBACK");

int i,result;

string t\_msg;

t\_msg= msg->data;

t\_len = t\_msg.size();

t\_msg.copy(t\_buff,t\_len);

write(fd,t\_buff,t\_len);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\*@brief 设置串口通信速率

\*@param fd 类型 int 打开串口的文件句柄

\*@param speed 类型 int 串口速度

\*@return void

\*/

int speed\_arr[] = {B115200, B38400, B19200, B9600, B4800, B2400, B1200, B300,

B115200, B38400, B19200, B9600, B4800, B2400, B1200, B300, };

int name\_arr[] = {115200, 38400, 19200, 9600, 4800, 2400, 1200, 300,

115200, 38400, 19200, 9600, 4800, 2400, 1200, 300, };

void set\_speed(int fd, int speed){

int i;

int status;

struct termios Opt;

tcgetattr(fd, &Opt);

for ( i= 0; i < sizeof(speed\_arr) / sizeof(int); i++) {

if (speed == name\_arr[i]) {

tcflush(fd, TCIOFLUSH);

cfsetispeed(&Opt, speed\_arr[i]);

cfsetospeed(&Opt, speed\_arr[i]);

status = tcsetattr(fd, TCSANOW, &Opt);

if (status != 0) {

perror("tcsetattr fd1");

return;

}

tcflush(fd,TCIOFLUSH);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\*@brief 设置串口数据位，停止位和效验位

\*@param fd 类型 int 打开的串口文件句柄

\*@param databits 类型 int 数据位 取值 为 7 或者8

\*@param stopbits 类型 int 停止位 取值为 1 或者2

\*@param parity 类型 int 效验类型 取值为N,E,O,,S

\*/

int set\_Parity(int fd,int databits,int stopbits,int parity)

{

struct termios options;

if ( tcgetattr( fd,&options) != 0) {

perror("SetupSerial 1");

return(FALSE);

}

options.c\_cflag &= ~CSIZE; //对数据不使用掩码

switch (databits) /\*设置数据位数\*/

{

case 7:

options.c\_cflag |= CS7;

break;

case 8:

options.c\_cflag |= CS8;

break;

default:

fprintf(stderr,"Unsupported data size\n"); return (FALSE);

}

switch (parity)

{

case 'n':

case 'N':

options.c\_cflag &= ~PARENB; /\* 不使用偶校验 \*/

options.c\_iflag &= ~INPCK; /\* \*/

break;

case 'o':

case 'O':

options.c\_cflag |= (PARODD | PARENB); /\* 设置为奇效验\*/

options.c\_iflag |= INPCK; /\* Disnable parity checking \*/

break;

case 'e':

case 'E':

options.c\_cflag |= PARENB; /\* Enable parity \*/

options.c\_cflag &= ~PARODD; /\* 转换为偶效验\*/

options.c\_iflag |= INPCK; /\* Disnable parity checking \*/

break;

case 'S':

case 's': /\*as no parity\*/

options.c\_cflag &= ~PARENB;

options.c\_cflag &= ~CSTOPB;break;

default:

fprintf(stderr,"Unsupported parity\n");

return (FALSE);

}

/\* 设置停止位\*/

switch (stopbits)

{

case 1:

options.c\_cflag &= ~CSTOPB;

break;

case 2:

options.c\_cflag |= CSTOPB;

break;

default:

fprintf(stderr,"Unsupported stop bits\n");

return (FALSE);

}

/\* Set input parity option \*/

if (parity != 'n')

options.c\_iflag |= INPCK;

tcflush(fd,TCIFLUSH);

options.c\_cc[VTIME] = 10; /\* 设置超时10 seconds\*/

options.c\_cc[VMIN] = 0; /\* Update the options and do it NOW \*/

if (tcsetattr(fd,TCSANOW,&options) != 0)

{

perror("SetupSerial 3");

return (FALSE);

}

options.c\_iflag &=~(IXON | IXOFF | IXANY);

options.c\_iflag &=~(INLCR | IGNCR | ICRNL); //此处用于关掉读取回显

options.c\_lflag &= ~(ICANON | ECHO | ECHOE| ISIG); /\*Input\*///

options.c\_oflag &= ~OPOST; /\*Output\*/

return (TRUE);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[])

{

ros::init(argc, argv, "serial\_example\_node"); //初始化节点

ros::NodeHandle nh; //声明节点句柄

ros::Subscriber write\_sub = nh.subscribe("write", 10000, write\_callback); //订阅主题，并配置回调函数

ros::Publisher read\_pub = nh.advertise<std\_msgs::String>("read", 10000); //发布主题

ros::Rate r(10); //设置循环频率为1s一次

std\_msgs::String r\_msg; //ros 收到信息的变量

std\_msgs::String t\_msg; //ros 收到发送的变量

printf("Start...\n"); //开启

fd = open(UART\_DEVICE, O\_RDWR|O\_NOCTTY|O\_NDELAY); //打开串口 获取句柄 O\_RDWR读写模式 O\_NDELAY非阻塞模式

//打开串口失败

if (fd < 0) {

perror(UART\_DEVICE);

exit(1);

}

//打开串口成功

printf("Open success\n");

set\_speed(fd,115200); //设定传输波特率

//设定传输参数

if (set\_Parity(fd,8,1,'N') == FALSE){

printf("Set Parity Error\n");

exit (0);

}

printf("Reading...\n"); //参数设置成功

while(ros::ok())

{

ROS\_INFO\_STREAM(r\_len);

r\_len = read(fd,r\_buff, 255); //串口数据读取

if(r\_len > 0) //读取为空 返回第一句

{

r\_buff[r\_len+1]=0; //写入0结束读取

r\_msg.data = r\_buff; //数组赋值给对应节点消息对象

read\_pub.publish(r\_msg);

}

//tcflush(fd, TCIOFLUSH); //清掉串口缓存

ros::spinOnce(); //调用回调函数，处理写入数据

r.sleep(); //频率控制

}

printf("Close...\n"); //结束代码

close(fd); //释放串口

return 0;

}

附：rt.cpp

#include <ros/ros.h> //ros库

#include <std\_msgs/String.h>

#include <std\_msgs/Empty.h>

//回调函数

void read\_callback(const std\_msgs::String::ConstPtr& msg)

{

ROS\_INFO\_STREAM("CALLBACK");

ROS\_INFO\_STREAM(msg->data);

}

int main(int argc, char \*argv[])

{

//初始化节点

ros::init(argc, argv, "rt\_test");

//声明节点句柄

ros::NodeHandle nh;

//订阅主题，并配置回调函数

ros::Subscriber r\_s = nh.subscribe("read", 10000, read\_callback);

//发布主题

ros::Publisher w\_p = nh.advertise<std\_msgs::String>("write", 10000);

ros::Rate r(2);

std\_msgs::String r\_msg; //ros 收到信息的变量

std\_msgs::String w\_msg; //ros 收到发送的变量

w\_msg.data = "open ros!";

printf("Start...\n"); //开启

while(ros::ok())

{

w\_p.publish(w\_msg);

//ROS\_INFO\_STREAM("Send");

//ROS\_INFO\_STREAM(w\_msg.data);

ros::spinOnce(); //调用订阅回调函数

r.sleep();

}

printf("Close...\n");

return 0;

}