Ubuntu串口通讯

查看串口设备

1. 打开终端，插入串口设备，输入下面指令

```
ls /dev/tty*
```

2. 通过对比分配设备前后/dev/ 目录下的 tty* 文件，可以了解到插入的 USB 转串口线对应的是哪个设备文件。

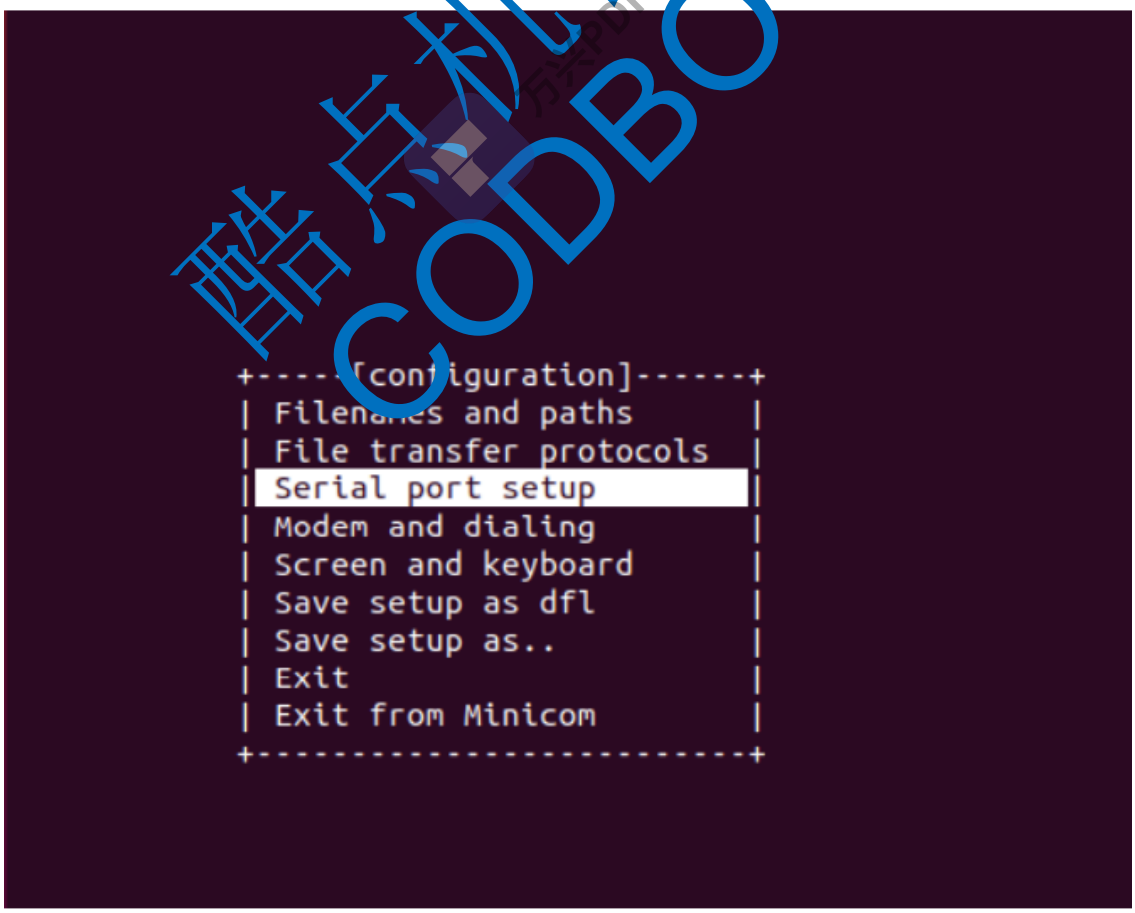
安装和配置 minicom

1. 输入以下指令，安装minicom

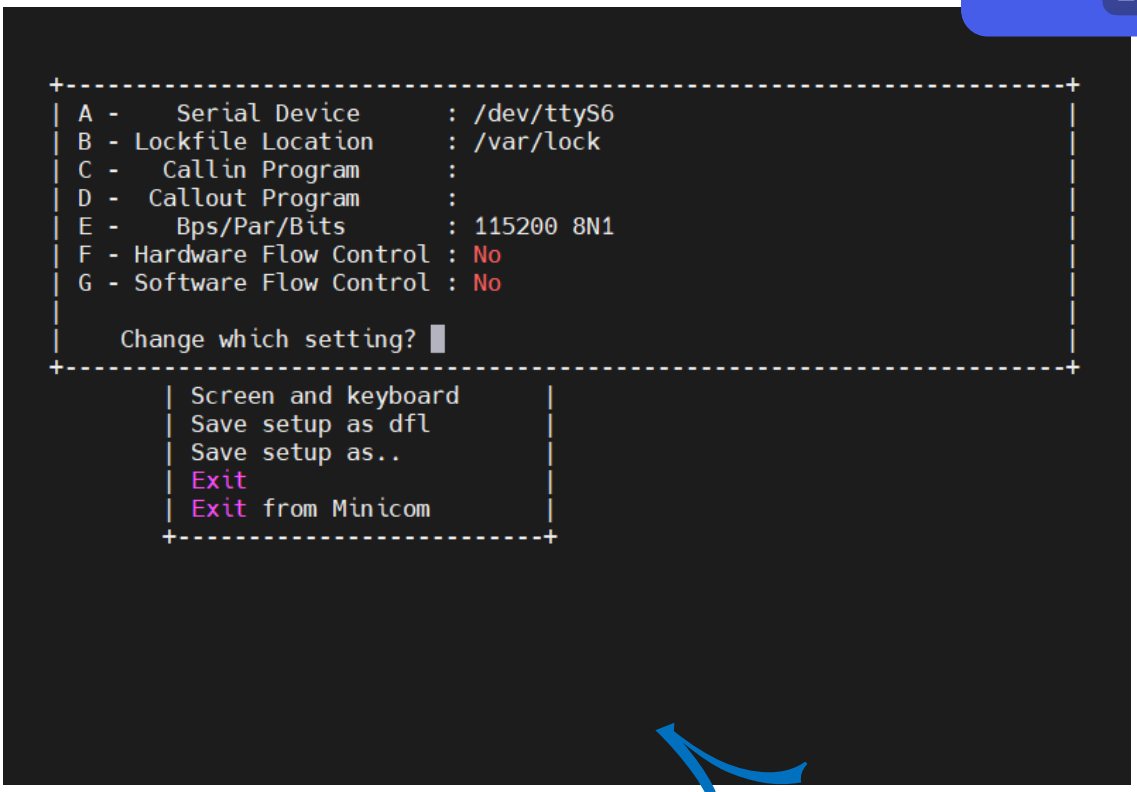
```
 sudo apt install minicom
```

2. 安装成功后，赋予root权限, 进入minicom配置界面

```
sudo minicom -s
```



上图是 minicom 运行时的配置界面，注意执行 minicom 命令时需要使用 sudo 获取权限，否则无法修改设备的参数。在该界面中使用键盘的上下方向键和回车键可以进入菜单进行配置。下面我们选择“Serial port setup”菜单配置串口参数，如下图所示：



在配置串口参数页面中根据提示的按键“A”、“E”、“F”配置串口设备为“/dev/ttyS6”（根据 自己的电脑设备选择）、波特率为“115200”、以及不使用硬件流控“No”，配置完成后按回车键退出当前菜单。然后再选择“Save setup as dfl”菜单保存配置，见下图（若提示无法保存，请确保 前面是使用“sudo”权限运行 minicom 的），保存完成后选择“Exit”菜单或按键盘的“Esc”键即 可进入终端界面。如下图所示：

在 minicom 的终端界面中，按下 Ctrl+A 键再按下 Z 键可以查看帮助，按下 Ctrl+A 键再按下 X 键可以退出

串口通讯

ubuntu上串口通讯一共分为三个部分，终端设备配置、串口发送数据、串口读取数据。

1. 终端串口配置

   在终端串口配置中，主要用到一个结构体termios,这个结构体用于设置终端设备的参数，包括波特率、数据 位数、校验位等，具体配置使用如下：

```c
int fd;
/*获取串口文件描述符,并使用读写权限打开串口*/
fd = open(path, O_RDWR);

if(fd < 0)
{
    printf("fail to open %s device \n", path);
return 0;
}

/*用于串口波特率、数据 位数、校验位配置*/
struct termios opt;

/*清空串口接收数据缓冲区*/
tcflush(fd, TCIOFLUSH);

//获取串口参数 opt
tcgetattr(fd, &opt);

//设置串口输出波特率为115200
cfsetospeed(&opt, B115200);

//设置串口输入波特率为115200
cfsetispeed(&opt, B115200);

//设置串口数据位
opt.c_cflag &= ~CSIZE;
opt.c_cflag |= CS8;             /*设置数据位为8位*/

//校验位
opt.c_cflag &= ~PARENB;   /*不使用奇偶检验*/
 opt.c_cflag &= ~INPCK;    /*禁止输入奇偶检测*/

//设置停止位
opt.c_cflag &= ~CSTOPB; /*设置停止位为1位*/

opt.c_iflag &= ~(INLCR);/*禁止将输入中的换行符NL映射为回车-换行CR*/
opt.c_iflag &= ~(IXON | IXOFF | IXANY); //不要软件流控制
opt.c_oflag &= ~OPOST;
opt.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //原始模式
//更新配置
tcsetattr(fd, TCSANOW, &opt);
```

以上代码就是对终端串口的参数配置

2. 串口写数据

串口写数据主要使用 write()，该函数原型如下：

```c
ssize_t write(int fd, const void *buf, size_t nbyte)

fd：文件描述符，
buf：指定的缓冲区，即指针，指向一段内存单元；
nbyte：要写入文件指定的字节数
返回值：写入文档的字节数（成功）；-1（出错）

该函数在使用时，一定要打开串口设备。
```

3. 串口读数据

串口读取数据主要使用read(),该函数原型如下：

```
ssize_t read(int fd, void *buf, size_t count);

参数
count
是请求读取的字节数，读上来的数据保存在缓冲区buf中，同时文件的当前读写位置向后移。注意这个读写位置和使用C标准I/O库时的读写位置有可能不同，这个读写位置是记在内核中的，而使用C标准I/O库时的读写位置是用户空间I/O缓冲区中的位置。比如用fgetc读一个字节，fgetc有可能从内核中预读1024个字节到I/O缓冲区中，再返回第一个字节，这时该文件在内核中记录的读写位置是1024，而在FILE结构体中记录的读写位置是1。注意返回值类型是ssize_t，表示有符号的size_t，这样既可以返回正的字节数、0（表示到达文件末尾）也可以返回负值-1（表示出错）。
read函数返回时，返回值说明了buf中前多少个字节是刚读上来的。有些情况下，实际读到的字节数（返回值）会小于请求读的字节数count，例如：读常规文件时，在读到count个字节之前已到达文件末尾。例如，距文件末尾还有30个字节而请求读100个字节，则read返回30，下次read将返回0。

返回值
成功返回读取的字节数，出错返回-1并设置errno，如果在调read之前已到达文件末尾，则这次read返回0
```

串口完整数据发送接收程序

下面是完整串口数据接收发送程序，可以直接在Ubuntu上运行使用，运行之前请给串口设备赋予读写权限。

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <string.h>
#include <sys/ioctl.h>
#include <stdint.h>

/*根据串口对应的设备，进行修改*/
const char default_path[] = "/dev/ttyS6";
int main(int argc, char *argv[])
{
    int fd;
    int res;
    char *path;
    char buf[1024] = "Embedfire tty send test\n";

    if(argc > 1)
        path = argv[1];
    else
        path = (char *)default_path;

    printf("this is tty/usart demo\n");
```

```
27
28      /*获取串口文件描述符*/
29      fd = open(path, O_RDWR);
30
31      if(fd < 0)
32      {
33          printf("fail to open %s device \n", path);
34      return 0;
35      }
36
37      struct termios opt;
38
39      /*清空串口接收数据缓冲区*/
40      tcflush(fd, TCIOFLUSH);
41
42      //获取串口参数 opt
43      tcgetattr(fd, &opt);
44
45    //设置串口输出波特率
46    cfsetospeed(&opt, B115200);
47
48    //设置串口输入波特率
49    cfsetispeed(&opt, B115200);
50
51    //设置串口数据位
52    opt.c_cflag &= ~CSIZE;
53    opt.c_cflag |= CS8;       /*设置数据位为8位*/
54
55    //校验位
56    opt.c_cflag &= ~PARENB;   /*不使用奇偶校验*/
57    opt.c_iflag &= ~INPCK;    /*禁止输入奇偶检测*/
58
59    //设置停止位
60    opt.c_cflag &= ~CSTOPB;   /*设置停止位为1位*/
61
62    opt.c_iflag &= ~(INLCR);/*禁止将输入中的换行符NL映射为回车-换行CR*/
63
64    opt.c_iflag &= ~(IXON | IXOFF | IXANY); //不要软件流控制
65    opt.c_oflag &= ~OPOST;
66    opt.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //原始模式
67    //更新配置
68    tcsetattr(fd, TCSANOW, &opt);
69
70    printf("device %s is set to 115200bps, 8N1", path);
71
72    do{
73    write(fd, buf, strlen(buf));
74
75    res = read(fd, buf, 1024);
76      if(res > 0)
77    {
78     buf[res] = '\0';
79     for(int i = 0; i < res; i ++)
80     printf("%x ", buf[i]);
81     packet_unpack(buf, res);
82
83    }
84      usleep(200000);
```

```
    }while(res >= 0);

    printf("read error,res = %d",res);

    close(fd);

    return 0;

}

```

运行该程序，即可看到接收的16进制的数据

车辆底层通讯

车辆底层串口通讯主要通过循环读取一个字节数据，然后根据具体的协议，判断该字节属于那个指令，并将他放进对应指令的缓冲区中。具体解析程序如下：

数据接收部分：

```
  while(1)
    {
      /*发送控制命令，速度100，  左转向10°*/
       sendCarControlCmd( &fd, 100,  0, 10);
       // 接收字符串
        res = read(fd, &buf, 1);
        if (res > 0) {

            /*数据解析*/
            packet_unpack(buf);
        }

    }
```

数据解析部分：

packet_unpack ()

```c
/*缓存每一帧数据，并缓存下来*/
void packet_unpack(uint8_t _buf)
{
    static uint8_t uart_flag = 1;
    static uint8_t s_uartBuf[100];
    static uint8_t s_len = 0;
    if (_buf == 0xFD)
    {
        s_uartBuf[0] = 0xFD;
        uart_flag = 1;
        s_len++;
    }
    else
    {
        if (uart_flag == 1)
        {
            if (s_len > s_uartBuf[1] + 1)
            {
                s_uartBuf[s_len] = _buf;
                s_len++;

                carInfoParse(s_uartBuf, s_len);
                uart_flag = 0;
                s_len = 0;
                return;
            }
            else
            {
                s_uartBuf[s_len] = _buf;
                s_len++;
            }
        }
    }
}
```

carInfoParse()

```c
/*根据缓存的数据，分别存入各个缓冲区*/
uint8_t carInfoParse(uint8_t *_buf, uint8_t _len)
{
    //根据异或校验判断数据是否存在接收错误
    if (_buf[_len - 1] == xor_check(&_buf[2], _len - 3))
    {

        switch (_buf[2])
        {
        case 0x02: // 速度、转向

            g_tCarMoveInfo.x_dir = _buf[3];
            g_tCarMoveInfo.x_lineSpeed = _buf[4] << 8 | _buf[5];
            g_tCarMoveInfo.y_dir = _buf[6];
            g_tCarMoveInfo.y_lineSpeed = _buf[7] << 8 | _buf[8];
            g_tCarMoveInfo.steerDir = _buf[9];
            g_tCarMoveInfo.steerAngle = _buf[10] << 8 | _buf[11];

            break;
```

```
20
21          case 0x03: //电机转速
22              g_tCarMotorInfo.motor1Speed = _buf[3] << 8 | _buf[4];
23              g_tCarMotorInfo.motor2Speed = _buf[5] << 8 | _buf[6];
24              g_tCarMotorInfo.motor3Speed = _buf[7] << 8 | _buf[8];
25              g_tCarMotorInfo.motor4Speed = _buf[9] << 8 | _buf[10];
26
27              break;
28
29          case 0x04: //电压
30              g_tCarBatteryInfo.voltage = _buf[3];
31              //printf("%x\n", g_tCarBatteryInfo.voltage );
32              break;
33
34          case 0x05: //IMU pitch roll yaw
35              g_tCarImuAttitudeInfo.pitchSymbol = _buf[3];
36              g_tCarImuAttitudeInfo.pitch = _buf[4] << 8 | _buf[5];
37              g_tCarImuAttitudeInfo.rollSymbol = _buf[6];
38              g_tCarImuAttitudeInfo.roll = _buf[7] << 8 | _buf[8];
39              g_tCarImuAttitudeInfo.yawSymbol = _buf[9];
40              g_tCarImuAttitudeInfo.yaw = _buf[10] << 8 | _buf[11];
41
42          case 0x06: //imu 原始数据
43              g_tCarImuRawInfo.gyroxSymbol = _buf[3];
44              g_tCarImuRawInfo.gyrox = _buf[4] << 8 | _buf[5];
45              g_tCarImuRawInfo.gyroySymbol = _buf[6];
46              g_tCarImuRawInfo.gyroy = _buf[7] << 8 | _buf[8];
47              g_tCarImuRawInfo.gyrozSymbol = _buf[9];
48              g_tCarImuRawInfo.gyroz = _buf[10] << 8 | _buf[11];
49              g_tCarImuRawInfo.accelxSymbol = _buf[12];
50              g_tCarImuRawInfo.accelx = _buf[13] << 8 | _buf[14];
51              g_tCarImuRawInfo.accelySymbol = _buf[15];
52              g_tCarImuRawInfo.accely = _buf[16] << 8 | _buf[17];
53              g_tCarImuRawInfo.accelzSymbol = _buf[18];
54              g_tCarImuRawInfo.accelz = _buf[19] << 8 | _buf[20];
55              g_tCarImuRawInfo.quatwSymbol = _buf[21];
56              g_tCarImuRawInfo.quatw = _buf[22] << 8 | _buf[23];
57              g_tCarImuRawInfo.quatxSymbol = _buf[24];
58              g_tCarImuRawInfo.quatx = _buf[25] << 8 | _buf[26];
59              g_tCarImuRawInfo.quatySymbol = _buf[27];
60              g_tCarImuRawInfo.quaty = _buf[28] << 8 | _buf[29];
61              g_tCarImuRawInfo.quatzSymbol = _buf[30];
62              g_tCarImuRawInfo.quatz = _buf[31] << 8 | _buf[32];
63              ;
64              break;
65
66          case 0x07: //车辆类型
67              g_tCarTypeInfo.carType = _buf[3];
68
69              break;
70
71          default:
72
73              break;
74          }
75      }
76  }
77
```

```
/*缓存每一帧数据，并缓存下来*/
void packet_unpack(uint8_t _buf)
{
    static uint8_t uart_flag = 1;
    static uint8_t s_uartBuf[100];
    static uint8_t s_len = 0;
    if (_buf == 0xFD)
    {
        s_uartBuf[0] = 0xFD;
        uart_flag = 1;
        s_len++;
    }
    else
    {
        if (uart_flag == 1)
        {
            if (s_len > s_uartBuf[1] + 1)
            {
                s_uartBuf[s_len] = _buf;
                s_len++;

                carInfoParse(s_uartBuf, s_len);
                uart_flag = 0;
                s_len = 0;
                return;
            }
            else
            {
                s_uartBuf[s_len] = _buf;
                s_len++;
            }
        }
    }
}
```

小车控制部分：

```
void sendCarControlCmd(int *fd, int16_t _xLineSpeed, int16_t _yLineSpeed,
int16_t _steerAngle)
{
    uint8_t _buf[13] = {0};
    _buf[0] = 0xCD;
    _buf[1] = 0x0a;
    _buf[2] = 0x01;

    if (_xLineSpeed > 0)
    {
        _buf[3] = 0x00;
    }
    else
    {
        _buf[3] = 0x01;
    }

    _buf[4] = (abs(_xLineSpeed) & 0xff00) >> 8;
    _buf[5] = (abs(_xLineSpeed) & 0x00ff);

```

```
    if (_yLineSpeed > 0)
    {
        _buf[6] = 0x00;
    }
    else
    {
        _buf[6] = 0x01;
    }

    _buf[7] = (abs(_yLineSpeed) & 0xff00) >> 8;
    _buf[8] = (abs(_yLineSpeed) & 0x00ff);

    if (_steerAngle > 0)
    {
        _buf[9] = 0x00;
    }
    else
    {
        _buf[9] = 0x01;
    }

    _buf[10] = (abs(_steerAngle) & 0xff00) >> 8;
    _buf[11] = (abs(_steerAngle) & 0x00ff);

    _buf[12] = xor_check(&_buf[2], _buf[1]);

    write(*fd, _buf, 13);
}
```

完整代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <string.h>
#include <sys/ioctl.h>
#include <stdint.h>

typedef struct
{
    uint8_t head;
    uint8_t frame_len;
    uint8_t frame_code;
    uint8_t *frame_data;
    uint8_t frame_crc;

} uart_frame_t;

typedef struct
{

    uint8_t x_dir;         //x轴方向
    uint16_t x_lineSpeed; //x轴线速度
```

```
    uint8_t y_dir;         //Y轴方向
    uint16_t y_lineSpeed; //y轴线速度
    uint8_t steerDir;      //转向方向
    uint16_t steerAngle;  //转向角度

} carMoveInfo_t;

typedef struct
{
    uint16_t motor1Speed; // 电机1转速
    uint16_t motor2Speed; // 电机2转速
    uint16_t motor3Speed; // 电机3转速
    uint16_t motor4Speed; // 电机4转速

} carMotorInfo_t;

typedef struct
{
    uint8_t voltage; //电压大小
} carBatteryInfo_t;

typedef struct
{

    uint8_t pitchSymbol; //picth 正负
    uint16_t pitch;
    uint8_t rollSymbol; //roll 正负
    uint16_t roll;
    uint8_t yawSymbol; //yaw 正负
    uint16_t yaw;

} carImuAttitude_t;

typedef struct
{
    uint8_t gyroxSymbol;
    uint8_t gyrox;

    uint8_t gyroySymbol;
    uint8_t gyroy;

    uint8_t gyrozSymbol;
    uint8_t gyroz;

    uint8_t accelxSymbol;
    uint8_t accelx;

    uint8_t accelySymbol;
    uint8_t accely;

    uint8_t accelzSymbol;
    uint8_t accelz;

    uint8_t quatwSymbol;
    uint8_t quatw;

    uint8_t quatxSymbol;
    uint8_t quatx;
```

```
85
86      uint8_t quatySymbol;
87      uint8_t quaty;
88
89      uint8_t quatzSymbol;
90      uint8_t quatz;
91
92  } carImuRaw_t;
93
94  typedef struct
95  {
96
97      uint8_t carType;
98
99  } carTypeInfo_t;
100
101 /*底盘反馈数据缓冲区*/
102 carMoveInfo_t g_tCarMoveInfo;
103 carMotorInfo_t g_tCarMotorInfo;
104 carBatteryInfo_t g_tCarBatteryInfo;
105 carImuAttitude_t g_tCarImuAttitudeInfo;
106 carImuRaw_t g_tCarImuRawInfo;
107 carTypeInfo_t g_tCarTypeInfo;
108
109 /*根据串口对应的设备，进行修改*/
110 const char default_path[] = "/dev/ttyS6";
111 /*串口解析*/
112 void packet_unpack(uint8_t buf);
113
114 int main(int argc, char *argv[])
115 {
116
117     int fd;
118     int res;
119     char *path;
120     char buf;
121
122     //若无输入参数则使用默认终端设备
123     if (argc > 1)
124         path = argv[1];
125     else
126         path = (char *)default_path;
127
128     //获取串口设备描述符
129     printf("This is tty/usart demo.\n");
130     fd = open(path, O_RDWR);
131     if (fd < 0)
132     {
133         printf("Fail to Open %s device\n", path);
134         return 0;
135     }
136
137     struct termios opt;
138
139     //清空串口接收缓冲区
140     tcflush(fd, TCIOFLUSH);
141     // 获取串口参数 opt
142     tcgetattr(fd, &opt);
```

```
143
144     //设置串口输出波特率
145     cfsetospeed(&opt, B115200);
146     //设置串口输入波特率
147     cfsetispeed(&opt, B115200);
148     //设置数据位数
149     opt.c_cflag &= ~CSIZE;
150     opt.c_cflag |= CS8;
151     //校验位
152     opt.c_cflag &= ~PARENB;
153     opt.c_iflag &= ~INPCK;
154     //设置停止位
155     opt.c_cflag &= ~CSTOPB;
156
157     //更新配置
158     tcsetattr(fd, TCSANOW, &opt);
159
160     opt.c_iflag &= ~(INLCR); /*禁止将输入中的换行符NL映射为回车-换行CR*/
161
162     opt.c_iflag &= ~(IXON | IXOFF | IXANY); //不要软件流控制
163     opt.c_oflag &= ~OPOST;
164     opt.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //原始模式
165
166     tcsetattr(fd, TCSANOW, &opt); //更新终端配置
167
168     printf("device %s is set to 115200bps, 8N1\n", path);
169
170     while (1)
171     {
172         /*发送控制命令，速度100，左转向100*/
173         sendCarControlCmd(&fd, 100, 0, 100);
174         // 接收字符串
175         res = read(fd, &buf, 1);
176         if (res > 0)
177         {
178
179             /*数据解析*/
180             packet_unpack(buf);
181         }
182     }
183
184     printf("read error,res = %d", res);
185
186     close(fd);
187     return 0;
188 }
189
190 //异或校验
191 uint8_t xor_check(uint8_t *_buf, uint8_t len)
192 {
193     uint8_t xorTemp = _buf[0];
194
195     for (int i = 1; i < len; i++)
196     {
197         xorTemp ^= (*(_buf + i));
198     }
199     return xorTemp;
200 }
```

```
201
202 /*根据缓存的数据，分别存入各个缓冲区*/
203 uint8_t carInfoParse(uint8_t *_buf, uint8_t _len)
204 {
205     //根据异或校验判断数据是否存在接收错误
206     if (_buf[_len - 1] == xor_check(&_buf[2], _len - 3))
207     {
208
209         switch (_buf[2])
210         {
211         case 0x02: // 速度、转向
212
213             g_tCarMoveInfo.x_dir = _buf[3];
214             g_tCarMoveInfo.x_lineSpeed = _buf[4] << 8 | _buf[5];
215             g_tCarMoveInfo.y_dir = _buf[6];
216             g_tCarMoveInfo.y_lineSpeed = _buf[7] << 8 | _buf[8];
217             g_tCarMoveInfo.steerDir = _buf[9];
218             g_tCarMoveInfo.steerAngle = _buf[10] << 8 | _buf[11];
219
220             break;
221
222         case 0x03: //电机转速
223             g_tCarMotorInfo.motor1Speed = _buf[3] << 8 | _buf[4];
224             g_tCarMotorInfo.motor2Speed = _buf[5] << 8 | _buf[6];
225             g_tCarMotorInfo.motor3Speed = _buf[7] << 8 | _buf[8];
226             g_tCarMotorInfo.motor4Speed = _buf[9] << 8 | _buf[10];
227
228             break;
229
230         case 0x04: //电压
231             g_tCarBatteryInfo.voltage = _buf[3];
232             //printf("%x\n", g_tCarBatteryInfo.voltage );
233             break;
234
235         case 0x05: //IMU pitch roll yaw
236             g_tCarImuAttitudeInfo.pitchSymbol = _buf[3];
237             g_tCarImuAttitudeInfo.pitch = _buf[4] << 8 | _buf[5];
238             g_tCarImuAttitudeInfo.rollSymbol = _buf[6];
239             g_tCarImuAttitudeInfo.roll = _buf[7] << 8 | _buf[8];
240             g_tCarImuAttitudeInfo.yawSymbol = _buf[9];
241             g_tCarImuAttitudeInfo.yaw = _buf[10] << 8 | _buf[11];
242
243         case 0x06: //imu 原始数据
244             g_tCarImuRawInfo.gyroxSymbol = _buf[3];
245             g_tCarImuRawInfo.gyrox = _buf[4] << 8 | _buf[5];
246             g_tCarImuRawInfo.gyroySymbol = _buf[6];
247             g_tCarImuRawInfo.gyroy = _buf[7] << 8 | _buf[8];
248             g_tCarImuRawInfo.gyrozSymbol = _buf[9];
249             g_tCarImuRawInfo.gyroz = _buf[10] << 8 | _buf[11];
250             g_tCarImuRawInfo.accelxSymbol = _buf[12];
251             g_tCarImuRawInfo.accelx = _buf[13] << 8 | _buf[14];
252             g_tCarImuRawInfo.accelySymbol = _buf[15];
253             g_tCarImuRawInfo.accely = _buf[16] << 8 | _buf[17];
254             g_tCarImuRawInfo.accelzSymbol = _buf[18];
255             g_tCarImuRawInfo.accelz = _buf[19] << 8 | _buf[20];
256             g_tCarImuRawInfo.quatwSymbol = _buf[21];
257             g_tCarImuRawInfo.quatw = _buf[22] << 8 | _buf[23];
258             g_tCarImuRawInfo.quatxSymbol = _buf[24];
```

```
            g_tCarImuRawInfo.quatx = _buf[25] << 8 | _buf[26];
            g_tCarImuRawInfo.quatySymbol = _buf[27];
            g_tCarImuRawInfo.quaty = _buf[28] << 8 | _buf[29];
            g_tCarImuRawInfo.quatzSymbol = _buf[30];
            g_tCarImuRawInfo.quatz = _buf[31] << 8 | _buf[32];
            ;
            break;

        case 0x07: //车辆类型
            g_tCarTypeInfo.carType = _buf[3];

            break;

        default:

            break;
        }
    }
}

/*缓存每一帧数据，并缓存下来*/
void packet_unpack(uint8_t _buf)
{
    static uint8_t uart_flag = 1;
    static uint8_t s_uartBuf[100];
    static uint8_t s_len = 0;
    if (_buf == 0xFD)
    {
        s_uartBuf[0] = 0xFD;
        uart_flag = 1;
        s_len++;
    }
    else
    {
        if (uart_flag == 1)
        {
            if (s_len > s_uartBuf[1] + 1)
            {
                s_uartBuf[s_len] = _buf;
                s_len++;

                carInfoParse(s_uartBuf, s_len);
                uart_flag = 0;
                s_len = 0;
                return;
            }
            else
            {
                s_uartBuf[s_len] = _buf;
                s_len++;
            }
        }
    }
}

/*小车控制函数，
    fd :  串口设备的文件描述符
    _xLineSpeed: 小车x轴的线速度
```

```
    _yLineSpeed: 小车y轴的线速度
    _steerAngle: 小车的转向速度

*/
void sendCarControlCmd(int *fd, int16_t _xLineSpeed, int16_t _yLineSpeed,
int16_t _steerAngle)
{
    uint8_t _buf[13] = {0};
    _buf[0] = 0xCD;
    _buf[1] = 0x0a;
    _buf[2] = 0x01;

    if (_xLineSpeed > 0)
    {
        _buf[3] = 0x00;
    }
    else
    {
        _buf[3] = 0x01;
    }

    _buf[4] = (abs(_xLineSpeed) & 0xff00) >> 8;
    _buf[5] = (abs(_xLineSpeed) & 0x00ff);

    if (_yLineSpeed > 0)
    {
        _buf[6] = 0x00;
    }
    else
    {
        _buf[6] = 0x01;
    }

    _buf[7] = (abs(_yLineSpeed) & 0xff00) >> 8;
    _buf[8] = (abs(_yLineSpeed) & 0x00ff);

    if (_steerAngle > 0)
    {
        _buf[9] = 0x00;
    }
    else
    {
        _buf[9] = 0x01;
    }

    _buf[10] = (abs(_steerAngle) & 0xff00) >> 8;
    _buf[11] = (abs(_steerAngle) & 0x00ff);

    _buf[12] = xor_check(&_buf[2], _buf[1]);

    write(*fd, _buf, 13);
}
```