# 1.PID

直接贴出代码，应该不难理解 pid.h

```
#ifndef PIBOT_PID_H_
#define PIBOT_PID_H_

class PID{
    public:
        PID(float* input, float* feedback, float kp, float ki, float kd, unsigned
short max_output);
        short compute(float interval);
        void clear();
    private:
        float kp;
        float ki;
        float kd;
        float max_output;
        float* input;
        float* feedback;

        float error;
        float integra;
        float derivative;

        float previous_error;
};

#endif
```

pid.cpp

```
#include "pid.h"

#include "board.h"
#include <stdio.h>
```

```
PID::PID(float* _input, float* _feedback, float _kp, float _ki, float _kd,
unsigned short _max_output)
    :input(_input), feedback(_feedback), kp(_kp), ki(_ki), kd(_kd),
max_output(_max_output*1.0){

    clear();

        printf("pid=%ld %ld %ld\r\n", long(kp*1000), long(ki*1000),
long(kd*1000));
}

void PID::clear(){
    error = integra = derivative = previous_error =0;
}

short PID::compute(float interval){
    error = *input - *feedback;

        integra = integra + error*interval;
        derivative = (error - previous_error) / interval;

        previous_error = error;


        if (ki != 0)
        #ifdef PID_DEBUG_OUTPUT
                printf("integra=%ld max_output=%ld %ld\r\n", long(integra*1000),
long(-(max_output/ki*1000)), long(max_output/ki*1000));
        #endif
                if (integra < -(max_output/ki))
                {
                        printf("integra clear-\r\n");
                        integra = -(max_output/ki);
                }
                if (integra > max_output/ki)
                {
                        printf("integra clear+\r\n");
                        integra = max_output/ki;
                }

        float val = error*kp + integra*ki + derivative*kd;

        if (val < -max_output)
                val = -max_output+1;
        else if (val > max_output)
                val = max_output-1;

#ifdef PID_DEBUG_OUTPUT
        printf("error=%ld integra=%ld derivative=%ld val=%ld\r\n",
long(error*1000), long(integra*1000), long(derivative*1000), long(val*1000));
#endif

        return val;
}
```

# 2.通讯部分

## 2.1Transport接口

```
#ifndef PIBOT_TRANSPORT_H_
#define PIBOT_TRANSPORT_H_

class Transport{
    public:
        virtual bool init()=0;
        virtual bool read(unsigned char& ch)=0;
        virtual void write(unsigned char* data, unsigned char len) = 0;

};
#endif
```

## 2.2数据报接口

```
#ifndef PIBOT_DATA_FRAME_H_
#define PIBOT_DATA_FRAME_H_

enum MESSAGE_ID{
    ID_GET_VERSION = 0,
    ID_SET_ROBOT_PARAMTER = 1,
    ID_GET_ROBOT_PARAMTER = 2,
    ID_CLEAR_ODOM = 3,
    ID_SET_VELOCITY = 4,
    ID_GET_ODOM = 5,
    ID_GET_PID_DATA = 6,
    ID_MESSGAE_MAX
};

//消息关注回调接口
class Notify{
    public:
        virtual void update(const MESSAGE_ID id, void* data) = 0;
};

class Dataframe{
    public:
        virtual bool init()=0;
        virtual void register_notify(const MESSAGE_ID id, Notify* _nf)=0;//注册对某个消息的关注
        virtual bool data_recv(unsigned char c)=0;//处理接收数据，返回true 表示接收到完整的数据包
        virtual bool data_parse()=0;//解析包
```

```
        virtual bool interact(const MESSAGE_ID id)=0;//master调用用来发送数据、接收
响应
};

#endif
```

# 3.bsp接口及实现

基本根据接口名字定义就知道功能，不再赘述

```
#ifndef PIBOT_BOARD_H_
#define PIBOT_BOARD_H_

class Queue;

enum USART_NUMBER
{
    USART_0 = 0,
    USART_1,
    USART_2,
    USART_3,
    USART_4,
    USART_5,
    USART_6,
};

#define PIN_MODE_INPUT 0x0
#define PIN_MODE_OUTPUT 0x1
#define PIN_MODE_INPUT_PULLUP 0x2

class Board{
  public:
    virtual void enable_irq()=0;
    virtual void disable_irq()=0;
    virtual void usart_debug_init()=0;
    virtual void usart_init(unsigned char num, unsigned long buad)=0;
    virtual Queue* usart_getDataQueue(unsigned char num)=0;

    virtual void usart_write(unsigned char num, unsigned char ch)=0;
    virtual void usart_write(unsigned char num, unsigned char* data, unsigned char
len)=0;

    virtual void set_config(unsigned char* data, unsigned short len)=0;
    virtual void get_config(unsigned char* data, unsigned short len)=0;

    virtual void pin_init(unsigned char pin, unsigned char mode)=0;
    virtual void pin_write(unsigned char pin, unsigned char level)=0;
    virtual unsigned char pin_read(unsigned char pin)=0;

    virtual void pwm_init(unsigned char khz)=0;
```

```
    virtual void pwm_output(unsigned char pin, unsigned short pwm)=0;

    virtual unsigned long get_tick_count()=0;

    static Board* get();
};

#endif
```

附上Mega2560的实现board_mega2560

```
#include "board_mega2560.h"
#include "Arduino.h"
#include "EEPROM.h"

#include "TimerOne.h"
Board_Mega2560 Board_Mega2560::board;

#define CONFIG_EEPROM_BASE 0

int serial_puts(char c, struct __file*){
  Serial3.write(c);
  return c;
}

void printf_begin(void){
  fdevopen(&serial_puts, 0);
}

Board* Board::get(){
    return &Board_Mega2560::board;
}


Board_Mega2560::Board_Mega2560(){

}

Board_Mega2560::~Board_Mega2560(){

}

void Board_Mega2560::enable_irq(){
  interrupts();
}

void Board_Mega2560::disable_irq(){
  noInterrupts();
}

void Board_Mega2560::usart_debug_init()
{
```

```
  usart_init(USART_3, 115200);
  printf_begin();
}

void Board_Mega2560::usart_init(unsigned char num, unsigned long buad){
  if (num == (unsigned char)USART_0){
    printf("uart0 start\r\n");
    Serial.begin(buad);
  }else if (num == (unsigned char)USART_3){
    Serial3.begin(buad);
  }
}


Queue* Board_Mega2560::usart_getDataQueue(unsigned char num){
  if (num == (unsigned char)USART_0){
    return &usart1_queue;
  }

  return 0;
}

void Board_Mega2560::usart_write(unsigned char num, unsigned char ch){
  if (num == (unsigned char)USART_0){
    Serial.write(ch);
  }
}

void Board_Mega2560::usart_write(unsigned char num, unsigned char* data, unsigned
char len){
  if (num == (unsigned char)USART_0){
    Serial.write((char*)data, len);
  }else if (num == (unsigned char)USART_3){
    Serial3.write((char*)data, len);
  }
}


void serialEvent(){
  if (Serial.available()){
    if (!Board::get()->usart_getDataQueue(USART_0)->put(Serial.read())){
        //err
    }
  }
}

void Board_Mega2560::set_config(unsigned char* data, unsigned short len){
  for(unsigned short i=0; i<len;i++){
    EEPROM.write(CONFIG_EEPROM_BASE+i, data[i]);
    delayMicroseconds(2);
  }
}

void Board_Mega2560::get_config(unsigned char* data, unsigned short len){
```

```
  for(unsigned short i=0; i<len;i++){
    data[i] = EEPROM.read(CONFIG_EEPROM_BASE+i);
    delayMicroseconds(2);
  }
}

void Board_Mega2560::pin_init(unsigned char pin, unsigned char mode){
  pinMode(pin, mode);
}

void Board_Mega2560::pin_write(unsigned char pin, unsigned char level){
  digitalWrite(pin, level);
}

unsigned char Board_Mega2560::pin_read(unsigned char pin){
  return digitalRead(pin);
}

void Board_Mega2560::pwm_init(unsigned char khz){
  Timer1.initialize(1000.0/khz);
}

void Board_Mega2560::pwm_output(unsigned char pin, unsigned short value){
  if (value== 0)
    Timer1.disablePwm(pin);
    Timer1.pwm(pin, value);
}

unsigned long Board_Mega2560::get_tick_count(){
  return millis();
}
```

# 4.通讯实现类

下面看下通讯类的具体实现`usart_transport`，实现了`init read write`接口 `usart_transport.h`

```
#ifndef PIBOT_USART_TRANSPORT_H_
#define PIBOT_USART_TRANSPORT_H_

#include "transport.h"

class USART_transport:public Transport{
  public:
    USART_transport(unsigned char num, unsigned long buad);
    bool init();
    bool read(unsigned char& ch);
    void write(unsigned char* data, unsigned char len);
  private:
    unsigned char usart_num;
    unsigned long usart_buad;
```

```
};
#endif
```

usart_transport.cpp

```
#include "usart_transport.h"

#include "board.h"
#include <stdio.h>

#include "queue.h"

USART_transport::USART_transport(unsigned char num, unsigned long
buad):usart_num(num), usart_buad(buad){

}

bool USART_transport::init()
{
    //printf("port=%d %ld\r\n", usart_num, usart_buad);
    Board::get()->usart_init(usart_num, usart_buad);//这里根据bsp的库来实现串口读写
    return true;
}

bool USART_transport::read(unsigned char& ch){
    return Board::get()->usart_getDataQueue(usart_num)->get(ch);
}

void USART_transport::write(unsigned char* data, unsigned char len){
    Board::get()->usart_write(usart_num, data, len);
}
```

# 5.协议实现类

继续看下dataframe的实现simple_dataframe simple_dataframe.h定义了simple_dataframe的共用的数据结构

```
#ifndef PIBOT_SIMPLE_DATAFRAME_H_
#define PIBOT_SIMPLE_DATAFRAME_H_

#include "dataframe.h"
#include <string.h>

static const unsigned short MESSAGE_BUFFER_SIZE = 255;

#define FIX_HEAD 0x5A

struct Head{
```

```
    unsigned char flag;// 头部标记,固定值:0X5A
    unsigned char msg_id;// 消息ID,表示消息具体作用,决定消息体具体格式
    unsigned char length;// 消息体长度
};


struct Message{
    struct Head head;
    unsigned char data[MESSAGE_BUFFER_SIZE];
    unsigned char check;
    unsigned char recv_count;//已经接收的字节数

    Message(){}
    Message(unsigned char msg_id, unsigned char* data=0,unsigned char len=0){
        head.flag = FIX_HEAD;
        head.msg_id = msg_id;
        head.length = recv_count = len;
        check = 0;

        if (data != 0 && len !=0)
            memcpy(this->data, data, len);

        unsigned char* _send_buffer = (unsigned char*)this;

        unsigned int i = 0;
        for (i = 0; i < sizeof(head)+head.length; i++)
            check += _send_buffer[i];

        _send_buffer[sizeof(head)+head.length] = check;
    }
};

enum RECEIVE_STATE{
    STATE_RECV_FIX=0,
    STATE_RECV_ID,
    STATE_RECV_LEN,
    STATE_RECV_DATA,
    STATE_RECV_CHECK
};

#endif
```

# 6.slave的实现

```
#ifndef PIBOT_SIMPLE_DATAFRAME_SLAVE_H_
#define PIBOT_SIMPLE_DATAFRAME_SLAVE_H_

#include "simple_dataframe.h"

class Transport;
```

```
class Simple_dataframe : public Dataframe{
    public:
        Simple_dataframe(Transport* trans=0);

        void register_notify(const MESSAGE_ID id, Notify* _nf){
            if (id >= ID_MESSGAE_MAX)
                return;
            nf[id] = _nf;
        }

        bool init();
        bool data_recv(unsigned char c);
        bool data_parse();
        bool interact(const MESSAGE_ID id){return true;};
    private:
        bool send_message(const MESSAGE_ID id);
        bool send_message(const MESSAGE_ID id, unsigned char* data, unsigned char
len);
        bool send_message(Message* msg);
    private:
        Notify* nf[ID_MESSGAE_MAX];

        Message active_rx_msg;

        RECEIVE_STATE recv_state;
                Transport* trans;
};
#endif
```

# 7.总结

这里使用的通讯接口及数据接口类使得更换通讯口或者通讯协议变得简单，只按照接口实现`transport`和`dataframe`即可替换目前使用的串口及目前简单的通讯协议。同时底层的接口也解耦，更换STM32或者启动的板子也变得极为容易