**DHBLAB**

**代码设计规范**

V1.0

2025年5月

# 命名规范

## 文件命名

（1）头文件使用 .h 后缀，源文件使用 .c 后缀

（2）文件名使用小写字母，单词间用下划线分隔

（3）文件名应反映其包含的主要内容

**正确示例：**

uart\_driver.c/cpp

protocol\_handler.h

## 变量命名

（1）局部变量使用小驼峰命名法（首字母小写，后续单词首字母大写）

（2）全局变量添加 g\_ 前缀

（3）静态变量添加 s\_ 前缀

（4）常量使用全大写，单词间用下划线分隔

（5）变量名应清晰表达其用途

**正确示例：**

// 局部变量

uint8\_t bufferSize;

float sensorValue;

// 全局变量

uint8\_t g\_systemState;

UART\_HandleTypeDef g\_huart1;

// 静态变量

static uint32\_t s\_errorCount;

// 常量

#define MAX\_BUFFER\_SIZE 256

const uint8\_t FRAME\_HEADER = 0x55;

## 函数命名

（1）使用动词+名词的形式

（2）采用大驼峰命名法（每个单词首字母大写）

（3）函数名应清晰表达其功能

**正确示例：**

void InitSystem(void);

uint8\_t CalculateChecksum(uint8\_t \*data, uint32\_t length);

void HandleReceivedData(Frame\_t \*frame);

## 类型定义命名

（1）结构体、枚举和联合使用大驼峰命名法

（2）类型定义添加 \_t 后缀

**正确示例：**

typedef struct {

uint8\_t id;

uint8\_t length;

uint8\_t \*data;

} Message\_t;

typedef enum {

STATE\_IDLE,

STATE\_BUSY,

STATE\_ERROR

} SystemState\_t;

## 宏定义命名

（1）使用全大写字母，单词间用下划线分隔

**正确示例：**

#define MAX\_RETRY\_COUNT 3

#define ENABLE\_DEBUG\_OUTPUT

# 代码格式

## 缩进与空格

（1）使用4个空格进行缩进，不使用制表符

（2）运算符两侧添加空格

（3）逗号后添加空格

（4）函数名与左括号之间不加空格

（5）关键字与左括号之间加空格

**正确示例：**

if (condition) {

value = (a + b) \* c;

Function(param1, param2);

}

## 大括号

（1）左大括号与控制语句在同一行

（2）右大括号单独占一行

（3）即使只有一条语句，也使用大括号

**正确示例：**

if (condition) {

statement;

}

else {

statement;

}

while (condition) {

statement;

}

## 行长度

（1）每行代码不超过80个字符

（2）超过限制时进行适当换行，换行后进行两级缩进

**正确示例：**

void LongFunctionName(uint8\_t param1, uint8\_t param2,

uint8\_t param3, uint8\_t param4);

**错误示例：**

void LongFunctionName(uint8\_t param1, uint8\_t param2, uint8\_t param3, uint8\_t param4);

# 注释规范

## 文件头注释

每个源文件和头文件开头必须包含以下注释：

/\*\*

\* @file file\_name.c

\* @brief 简要描述文件功能

\* @details 详细描述文件功能和实现

\*

\* @author 作者名

\* @date 创建日期

\* @version 版本号

\*

\* @copyright Copyright (c) 年份 DHBLAB

\*/

## 函数注释

每个函数定义前必须包含以下注释：

/\*\*

\* @brief 简要描述函数功能

\* @param param1 参数1说明

\* @param param2 参数2说明

\* @return 返回值说明

\* @note 注意事项

\*/

## 变量和宏注释

重要的变量和宏定义应添加注释：

/\* 最大缓冲区大小 \*/

#define MAX\_BUFFER\_SIZE 256

/\* 系统状态枚举 \*/

typedef enum {

STATE\_IDLE, /\* 空闲状态 \*/

STATE\_BUSY, /\* 忙碌状态 \*/

STATE\_ERROR /\* 错误状态 \*/

} SystemState\_t;

## 代码内注释

（1）复杂的代码段应添加注释说明

（2）注释应说明"为什么"而不是"做什么"

（3）单行注释使用 // ，多行注释使用 /\* \*/

**正确示例：**

// 检查接收到的数据帧是否有效

if (frame->length > 0 && frame->checksum == calculatedChecksum) {

/\*

\* 数据帧有效，进行处理

\* 1. 解析功能码

\* 2. 执行对应操作

\*/

ProcessFrame(frame);

}

# 函数设计规范

## 函数长度

（1）函数不超过50行代码

（2）复杂的函数应拆分为多个子函数

## 参数数量

（1）函数参数不超过5个

（2）参数过多时考虑使用结构体封装

## 返回值

（1）函数返回值应明确表示执行结果

（2）错误处理应返回错误码或状态枚举

**正确示例：**

typedef enum {

RET\_OK,

RET\_ERROR,

RET\_TIMEOUT,

RET\_INVALID\_PARAM

} ReturnCode\_t;

ReturnCode\_t SendMessage(Message\_t \*message) {

if (message == NULL) {

return RET\_INVALID\_PARAM;

}

// 发送消息

return RET\_OK;

}

## 函数复杂度

（1）控制函数的循环嵌套层数不超过3层

（2）控制条件分支复杂度，避免过多的if-else嵌套

# 错误处理

## 错误检查

（1）函数开始处检查参数有效性

（2）对可能失败的操作进行错误检查

**正确示例：**

ReturnCode\_t ProcessData(uint8\_t \*data, uint32\_t length) {

// 参数检查

if (data == NULL || length == 0) {

return RET\_INVALID\_PARAM;

}

// 处理数据

ReturnCode\_t ret = DoSomething(data, length);

if (ret != RET\_OK) {

// 错误处理

return ret;

}

return RET\_OK;

}

## 资源释放

（1）确保在函数退出前释放所有分配的资源