研发管理规范

IMYFIT
吾 控 健 康

专注 • 专业 • 开放 • 共赢

挖掘物联网核心价值

我们专注于人员健康与位置管理领域

版本	日期	作者	审核人	备注
1.0	2019/07/16	赖正喜	赖正喜	初稿形成
2.0	2021/02/28	赖正喜 彭华 舒德军 刘远帆	康海强	去掉冗余要求，添加新要求 完善软件命名、增加 git 分支命名规则 新增 java、kotlin、js 编程规范
2.1	2022/07/14	赖正喜	康海强	修改函数命名规范和注释规范 （使用 VSCode 的扩展 Doxygen Documentation Generator 默认注释风格）

目 录

第 1 章 文档撰写规范 4
第 2 章 文件夹命名规范 4
第 3 章 文件命名规范 5
第 4 章 Git 版本管理规范 5
4.1 仓库命名规范 5
4.2 嵌入式软件分支规范 6
4.2.1 分支概述 6
4.2.2 分支命名 6
4.2.3 标签命名 7
4.3 系统软件分支规范 7
4.3.1 分支概述 7
4.3.2 分支命名 8
第 5 章 软件命名规范 8
第 6 章 C 编程规范 9
6.1 概述 9
6.2 命名规范 9
6.2.1 源文件 9
6.2.2 头文件 9
6.2.3 宏/枚举 10
6.2.4 变量 10
6.2.5 结构与联合体 11
6.2.6 函数 11
6.3 内容排版与注释 12
6.3.1 源文件 12
6.3.2 头文件 13
第 7 章 C++编程规范 16
7.1 概述 16
7.2 类命名规范 16

第 8 章 Java 编程规范 17
8.1 类命名规范 17
8.2 抽象类命名 17
8.3 接口命名 17
8.4 方法命名 18
8.5 属性命名 18
第 9 章 Kotlin 编程规范 19
9.1 类命名规范 19
9.2 抽象类命名 20
9.3 接口命名 20
9.4 方法命名 20
9.5 属性命名 21
第 10 章 JavaScript 编程规范 22
10.1 类命名 22
10.2 函数 22
10.3 Vue 视图文件 23
10.3.1 Vue 组件文件 23
10.4 变量命名 24
10.5 常量 24

第 1 章 文档撰写规范

所有相对正式的文档均按照如下要求撰写，可参阅文档模板《010_文档模板_内部资料_研发部使用_销售部建议使用_v2.0_20210214》撰写：

1、 段落行距 1.5 倍；

2、 正文宋体小四；

3、 题注宋体五号；图片标注在下方；表格标注在上方；

4、 数字英文采用 TIMES NEW ROMAN；

5、 一级标题：黑体小三居中；二级标题：黑体小四左对齐；三级标题：黑体小四左对齐。

第 2 章 文件夹命名规范

编号_项目名称[_其它信息][_时间]

命名采用英文小写字母（特殊名词除外）、汉字、阿拉伯数字和下划线混合方式，其中各个字段的含义如下所述：

a) 编号：由三位数组成，默认中间位数递增，例如：020、030。需要在 030 文件夹后面添加文件夹则编号变更为 031，依次类推。

b) 项目名称：采用项目编号、产品型号定义或者××系统，例如：a100、b10、患者时间轴系统。

c) 其它信息：该条目为可选，例如：需方公司名称、应用场景等。

d) 时间：该条目为可选，创建时间，例如：20190512。

示例：

《010_智能床带_huawei_20190825》

《230_智慧精神病院_猫度云科_20190825》

《891_uwb_location_algorithm_m100_imyfit_20210116》

《030_原理图》

《030_source》

第 3 章 文件命名规范

编号_文档用途_项目名称[_其它信息]_版本号_时间[_流水号]

命名采用英文小写字母（特殊名词除外）、汉字、阿拉伯数字和下划线混合方式，其中各个字段的含义如下所述：

a) 编号：由三位数组成，默认中间位数递增，例如：020、030。需要在 030 文档后面添加文档则编号变更为 031，依次类推。

b) 文档用途：表明该文档的作用，例如：详细设计说明书、解决方案。

c) 项目名称：采用项目编号、产品型号定义或者××系统，例如：a100、b10。

d) 其它信息：该条目为可选，例如：需方公司名称。

e) 版本号：文档需要多人修改，修改后必要时版本递增，例如：v1.0。

f) 时间：创建时间，例如：20190512。

g) 流水号：该条目为可选，防止名称相同被覆盖。每次修改后该编号可递增。

示例：

《011_详细设计说明书_智能床带 m11_大客户 huawei_v1.2.0_ 20190825_5.pdf》

《160_立项说明书_b10_v1.1_20190825_8.docx》

《070_解决方案_患者时间轴系统_绵阳中心医院&长虹_v1.0_20210210.docx》

《010_文档模板_内部资料_研发部使用_销售部建议使用_v2.0_20210214.docx》

第 4 章 Git 版本管理规范

4.1 仓库命名规范

历史项目仓库：

项目名称_平台名称

命名采用拼音/英文小写字母（特殊名词除外）、汉字、阿拉伯数字和下划线混合方式，其中各个字段的含义如下所述：

a) 项目名称：英文或者拼音。

b) 平台名称：表明所使用的平台或其它信息。

例如：b10_nrf52840、zhongchuan_fangcang

新项目仓库：

项目名称_时间

命名采用拼音/英文小写字母（特殊名词除外）、汉字、阿拉伯数字和下划线混合方式，其中各个字段的含义如下所述：

a) 项目名称：英文或者拼音。

b) 时间：表明项目创建时间。

例如：m11_20210119、three_demension_map_20210101

4.2 嵌入式软件分支规范

4.2.1 分支概述

开发过程中采用如下系列分支：

master：主分支（默认分支）。维持初始版本。

develop：日常开发分支。该分支正常保存了开发的最新代码。

feature：功能开发分支。只与 develop 分支交互，新功能开发完成后合并到 develop 分支上。

release：发布分支。属于生产分支，只能从 develop 分支得到。发布后需要给发布版本打标签。

hotfix：线上 bug 修复分支。只能从 develop 分支得到。

style：格式分支。该分支为规范注释格式而创建。

refactor：重构分支。不是新增功能，也不是修改 bug 的代码变动。

test：测试分支。

注：需要针对某个历史版本进行小的修改，基于版本标签创建分支进行。分功能 commit 提交时有详细的文字说明，禁止无说明提交。

4.2.2 分支命名

项目名称_所属分支[_其它信息]

a) 项目名称：采用项目编号、产品型号定义等，例如：x3、b10、m11 等。

b) 所属分支：满足 4.2.1 分支概述要求。

c) 其它信息：可选，为了直观查看，遵循如下规则。

- 大写字母 E 开始和结束的字段为其它信息字段；
- 大写字母 V 开始和结束的字段为版本信息；
- 大写字母 C 表示客户；
- 大写字母 P 表示功能；
- 大写字母 O 表示其他信息;

d) 每个项目/系统固定存在分支 xx_release。

e) 对于不同客户和功能可以扩展其他分支。

示例：

x3w_release

x3w_develop

x3w_feature_EChuaweiE

r9_release

r9_develop、r9_develop_EOsupport_otaCmaoduPno_uwbE、r9_feature_ECmaoduPnoneuwbE

r9_test_Egh3011_power_testE

4.2.3 标签命名

软件发布时需要随带一个标签，方便通过软件名称快速查找对应的代码版本，追溯故障问题。标签命名规则完全遵循《第 5 章软件命名规范》命名规范。

4.3 系统软件分支规范

4.3.1 分支概述

master：主分支，主要用来版本发布。

develop：日常开发分支，该分支正常保存了开发的最新代码。

feature：具体的功能开发分支，只与 develop 分支交互，新功能开发完成后提交 develop 合并到 develop 分支上。

release：release 分支可以认为是 master 分支的未测试版。比如某一期的功能全部开发完

成，那么就将 develop 分支合并到 release 分支，测试没有问题并且到了发布日期就合并到 master 分支，进行发布。

hotfix：线上 bug 修复分支。

注：测试开发完成的新版本或者新功能，直接从 release 拉取代码打包测试；测试成功后合并到 master 分支等待发布，发布只能从 master 拉取代码打包，禁止拉取 release 等其他分支上的代码进行发布。

4.3.2 分支命名

参考嵌入式软件分支命名《4.2.2 分支命名》规范。

第 5 章 软件命名规范

编号_开发阶段_项目名称_其它信息_编译时间

命名采用英文小写字母（特殊名词除外）、阿拉伯数字和下划线混合方式，其中各个字段的含义如下所述：

a) 此部分软件主要存放在内部 wiki 上供生产（含测试）和销售人员使用，信息尽可能多，方便非开发人员快速明白意图，此部分软件可能会流向客户使用，请认真命名。

b) 编号：即流水号，由三位数组成，默认中间位数递增，例如：020、030。需要在 030 软件后面添加软件则编号变更为 031，依次类推。

c) 开发阶段：只能为版本开发的三个阶段之一 alpha、beta、release。

d) 项目名称：采用项目编号、产品型号定义或者××系统，例如：a100、b10、患者时间轴系统。

f) 其它信息：为了方便批处理使用，遵循如下规则：

- 大写字母 E 开始和结束的字段为其它信息字段；
- 大写字母 V 开始和结束的字段为版本信息；
- 大写字母 C 表示客户；
- 大写字母 P 表示功能；
- 大写字母 O 表示其他信息;

e) 编译时间：软件被编译的时间，例如：20210128T2227。

示例：

《001_alpha_x3w_EChuaweiPxnbiotV1_0_2VE_20190825T1623.hex》

《001_beta_m100_EOwinserver2008_uwb_algorithmV1_2_3VE_20190906T1423.exe》

《001_release_r9_EOentire_support_ota_produce_test_onlyV1_0_0VE_20210128T2227.hex》

第 6 章 C 编程规范

6.1 概述

规范适用于当前嵌入式项目所有的平台，包含 bootloader 程序和 app 程序。

6.2 命名规范

6.2.1 源文件

一般形式：aaa_bbb_ccc.c

aaa：表示该源程序所属的系统，如 fire 消防、uwb 定位系统、temperature 体温系统。默认可以不使用所属的系统，自研系统比较少，外包时比较实用。

bbb：表示功能，部分专用名词大小写等不做严格规定，尽量满足行业用语。

ccc：表示其它信息，（如 m 表示主程序，i 表示输入，o 表示输出等）。

例如：

fire_tcp_protocol.c 消防 TCP 通信协议、dl_lora.c 电力 LoRa 通信协议、hr_user_task_gh301x.c 心率系统下的用户线程与 gh301x 相关。

6.2.2 头文件

一般形式：aaa_bbb_ccc.h

aaa：表示该源程序所属的系统（如 xf 消防，dl 电力等），特殊情况可以省略。

bbb：表示功能，部分专用名词大小写等不做严格规定，尽量满足行业用语。

ccc：一般情况可省略。当多个源程序文件属于同一功能的分支程序，用来表示源程序文件的功能类型（如 m 表示主程序，i 表示输入，o 表示输出等）。

例如：

fire_tcp_protocol.h 消防 TCP 通信协议、dl_lora.h 电力 LoRa 通信协议、

hr_user_task_gh301x.h 心率系统下的用户线程与 gh301x 相关。

6.2.3 宏/枚举

一般形式：AAA_BBB。

AAA、BBB：表示功能的组合，可简称亦可全称。定义名称必须能反映定义功能，并确保不与操作系统或其他系统软件的宏定义重复。

- 必须使用大写字母命名、用下划线间隔语义单元。
- 必须使用 typedef。
- 宏定义：宏所属功能大类名称_宏具体功能名称。

例如：
```
#define LORA_HEAD_LEN 10
```

- 枚举定义：所属功能大类名称_具体功能名称。

例如：
```
typedef enum{FALSE = 0, TRUE = !FALSE}BOOL;
typedef enum{LORA_DATA_LOCK, LORA_DATA_UNLOCK} LORA_RECE_LOCK;
```

6.2.4 变量

一般形式：x_aaa_bbb_ccc_t。

x：表示变量作用域；函数内的局部变量则省略，全局变量用 g，静态变量用 m。

aaa：表示数据类型，根据情况可选择省略。

◆ int8（unsigned char）简为 sc；

◆ uint8 （unsigned char）简为 uc；

◆ int16 （signed short）简为 ss；

◆ uint16 （unsigned short）简为 us；

◆ int32 （signed long）简为 sl；

◆ uint32 （unsigned long）简为 ul；

◆ p 指针。

bbb、ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

t：表示类型（可选）。str 字符串，arr 数组，struct 结构体或者其它说明。

例如：

```
uint8_t g_uc_uart_receive_buffer_arr[1204] = {0};

uint8_t *p_uart_rec_buffer = osal_mem_alloc(num_bytes);
```

6.2.5 结构与联合体

一般形式：aaa_bbb_t。

aaa、bbb：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。定义名称必须能反映功能，并确保不与操作系统或其他系统软件的定义重复。

- 必须使用大写字母命名、用下划线间隔语义单元，以_t 结束。
- 必须使用 typedef。
- 结构元素：aaabbbccc。aaa、bbb、ccc 表示元素功能的组合，该成员变量也可遵循 6.2.4 变量要求。

例如：

```
typedef struct

{

    OSAL_EVENT_HDR_t  hdr;

    uint8 opcode;

    uint8 numberdevicess;

    uint8 connection_endpoint;

    GAP_DEV_REC_t *pdevlist;

} ble_dev_discovery_event_t;

ble_dev_discovery_event_t g_ble_discovery_struct;
```

6.2.6 函数

一般形式：aaa_bbb_ccc()

aaa：表示本函数所在的源程序文件的**简称**。

bbb_ccc：表示功能组合，部分专用名词大小写等不做严格规定，尽量满足行业用语。

函数形参必须严格遵循 6.2.4 变量要求，如无形参必须加入 void 关键字。

例如：

//表示 simple_ble_entral.c 文件内的函数

```
simple_ble_central_process_msg(void);
```

//表示 GATT.c 文件内的函数

```
gatt_write_char_value(uint16 us_connect_handle, ATTWRITEREQ_t *p_request);
```

6.3 内容排版与注释

6.3.1 源文件

- 一般按功能，即把属于同一方面功能的内容集中写在一个源程序文件中。如内容较多，则可以把功能细分并比较均衡的写在多个源程序文件中。
- 一个源程序文件的行数一般不超过 500 行，否则细分为多个源程序文件。
- 文件顺序依次为：源程序说明注释（**考虑到跨平台与规范性，注释尽可能的采用英文**）、头文件引入、常量/全局变量（先注释再初始化）、静态变量初始化（先注释再初始化）、函数体。
- 注释使用 Doxygen 支持的 javadoc 风格。

1、 **源程序说明注释：**初始创建信息（时间和姓名）、修改信息（时间和姓名）、版本号信息、详细描述。

例如：

```
/**
 * @file wk_heart_storage.c
 * @author laizx (laizx@cositea.com)
 * @brief heart store into flash used by 'tsdb'
 * @version 0.1
 * @date 2022-07-11
 *
 * @copyright Copyright (c) 2022
 *
 */
```

2、 **头文件引入：**include 引用头文件，系统文件用<>、用户文件用""，如需要注释则位于上一行（尽量有注释）。

例如：

```
#include <string.h>
```

```
/** 电力巡检 TCP 通信协议解析 */

include "dl_tcp_config.h"
```

3、 **常量/全局变量（先注释再初始化）：**

例如：

```
/** 蓝牙外设 MAC 地址数组 */

uint8 g_uc_ble_peripheral_mac_arr[6] = {0};

/** 蓝牙手环的 UUID 数组 */

uint8 uc_k18s_uuid_arr[2] = {0};
```

4、 **静态变量初始化（先注释再初始化）：**

例如：

```
/** BLE 特征值 1 的句柄 */

static uint16 us_ble_char1_handle = 0;
```

5、 **函数体：**

例如：

```
/**
 * @brief write data to dw1000 use spi
 *
 * @param header_length -register address length
 * @param header_buffer -register address
 * @param body_length -want to write the length
 * @param body_buffer -the point of write data
 * @return int -0,sucess;1,failed
 */
int wk_write_to_spi(uint16 header_length, const uint8 *header_buffer, uint32
body_length, const uint8 *body_buffer)
{
}
```

6.3.2 头文件

- 一个头文件的行数一般不超过 500 行，否则细分为多个。
- 文件顺序依次为：头文件说明注释、条件编译、头文件引入、宏定义、结构体、常量/全局变量（先注释再定义）、函数体（先注释再定义）。

1、 **头文件说明注释：**包含文件名、初始创建信息（时间和姓名）、修改信息（时间和姓名）、版本号信息、详细描述。

例如：

```
/**
 * @file wk_heart_storage.h
 * @author laizx (laizx@cositea.com)
 * @brief
 * @version 0.1
 * @date 2022-07-11
 *
 * @copyright Copyright (c) 2022
 *
 */
```

2、 **条件编译：**头文件的开始和结尾用＃ifdef _头文件名_ #endif 控制重复被引用

例如：

```
#ifndef DL_LORA_CONFIG_H

#define DL_LORA_CONFIG_H

……

#endif
```

3、 **头文件引入：**include 引用头文件，系统文件用<>、用户文件用""，如需要注释则位于上一行（尽量保证有注释）。

例如：

```
#include <string.h>

/** 蓝牙配置头文件 */

#include "ble_config.h"
```

4、 **宏定义：**

例如：

```
/** define lora data length is 3 oct */

#define  LORA_HEAD  0x03
```

5、 **结构体：**

例如：

```
/**

 * @brief    蓝牙事件结构体
```

```
 * @details  存储各种连接数量
 */
typedef struct
{
    OSAL_EVENT_HDR_t  hdr;  /*!< Detailed description of the member var1 */
    uint8 opcode;           /*!< Detailed description of the member var1 */
    uint8 numdevs;          /*!< Detailed description of the member var1 */
    GAP_DEV_REC_t *pdevlist; /*!< Detailed description of the member var1 */
} ble_dev_discovery_event_t;
```

6、 **常量/全局变量（先注释再定义）：**

例如：

```
/** 蓝牙外设 MAC 地址数组 */
uint8 g_uc_ble_peripheral_mac_arr[6];
/** 蓝牙手环的 UUID 数组(只在当前*.c 、*.h 里面使用的变量) */
uint8 uc_k18s_uuid_arr[2] = {0};
```

7、 **函数体（先注释再定义）：**要提供给别的*.c 使用则需要加上 extern，只在当前对应的*.c 中使用则不用添加。

例如：

```
/**
 * @brief write data to dw1000 use spi
 *
 * @param header_length -register address length
 * @param header_buffer -register address
 * @param body_length -want to write the length
 * @param body_buffer -the point of write data
 * @return int -0,sucess;1,failed
 */
extern int wk_write_to_spi(uint16 header_length, const uint8 *header_buffer, uint32
body_length, const uint8 *body_buffer);
```

第 7 章 C++编程规范

7.1 概述

与 C 语言有相似规范时，则遵循第 6 章 C 编程规范，例如变量的命名、文件的命名等。

7.2 类命名规范

一般形式：AaaBbbCcc，必须遵从驼峰形式。

Aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

例如：

```
/**
 * @author laizx
 * @description 定时器线程类
 * @time 2020-02-19 09:22
 */


class TimerThread : public QThread
{
    Q_OBJECT
private:
    bool m_stop=false;
public:
    timerThread();
    void stopThread();
    void run() Q_DECL_OVERRIDE;
signals:
    void timerSendHealthData();
};
```

第 8 章 Java 编程规范

注：下载阿里编程规范，尽量避免程序出现阿里规范报错，如果我司规范与阿里规范冲突，参照我司规范。

8.1 类命名规范

一般形式：AaaBbbCcc，必须遵从驼峰形式。

Aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

类名之前必须加上注释，相关描述以及建立时间

```
/**
 * @author shudj
 * @description redis 操作类
 * @time 2021-02-19 09:22
 */
public class RedisUtil {
}
```

8.2 抽象类命名

同类命名

8.3 接口命名

一般形式：IAaaBbbCcc，必须遵从驼峰形式。

I:表示 interface 的缩写；Aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

类名之前必须加上注释，相关描述以及建立时间

```
/**
 * @author shudj
 * @description 员工 逻辑层处理
```

```
 * @time 2021-02-19 09:22
 */
public interface IWorkerService {
}
```

8.4 方法命名

一般形式：aaaBbbCcc，必须遵从驼峰形式。

aaa 最好是动词，表示某种行为，Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

只在本类中使用的方法，需要加上 private 修饰，禁止外访。

写方法的时候需要先注释，必须注释清楚该方法的含义，以及每个传入参数的具体含义

```
/**
 * @author shudj
 * @description 发送信息
 * @time 2021-02-19 09:22
 * @param message 发送的具体消息
 */
public void sendMessage(String message) {
}
```

8.5 属性命名

一般形式：aaaBbbCcc，必须遵从驼峰形式；静态常量属性全部大写下划线（_）分割，形如：AAA_BBB_CCC

aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

属性都使用 private 修饰，外界需要方位即使用 getXxx()才能访问禁止直接调用属性访问；

静态常量属性，如本类专属需要用 private 修饰，并且通 final static 修饰禁止更改，禁止外访；如果很多类都需要访问，使用 public 修饰，并且写入专门存放共有属性的类中，并使用 final static 修饰禁止修改。

属性前加注释，表名该属性的具体含义。

```
/**
 * @author shudj
 * @description 用户类
 * @time 2021-02-19 09:22
 */
public class User {

  public final static CLASS_NAME = “0001”
  private final static USER_ADDRESS = “xxx”
   // 用户名称
   private String userName;
   // 用户年龄
   private int age;

   public setUserName(String userName) {
       this.userName = username;
   }
   public String getUserName() {
       return this.userName;
   }
   ...
}
```

第 9 章 Kotlin 编程规范

9.1 类命名规范

一般形式：AaaBbbCcc，必须遵从驼峰形式。

Aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规

定，尽量满足行业用语。

类名之前必须加上注释，相关描述以及建立时间

```
/**
 * @author shudj
 * @description redis 操作类
 * @time 2021-02-19 09:22
 */
class RedisUtil {
}
```

9.2 抽象类命名

同类命名

9.3 接口命名

一般形式：IAaaBbbCcc，必须遵从驼峰形式。

I:表示 interface 的缩写；Aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

类名之前必须加上注释，相关描述以及建立时间

```
/**
 * @author shudj
 * @description 员工 逻辑层处理
 * @time 2021-02-19 09:22
 */
interface IWorkerService {
}
```

9.4 方法命名

一般形式：aaaBbbCcc，必须遵从驼峰形式。

aaa 最好是动词，表示某种行为，Bbb、Ccc：表示功能的组合，可简称亦可全称，部分

专用名词大小写等不做严格规定，尽量满足行业用语。

只在本类中使用的方法，需要加上 private 修饰，禁止外访。

写方法的时候需要先注释，必须注释清楚该方法的含义，以及每个传入参数的具体含义

```
/**
 * @author shudj
 * @description 发送信息
 * @time 2021-02-19 09:22
 * @param message 发送的具体消息
 */
Fun sendMessage(message: String) {
}
```

9.5 属性命名

一般形式：aaaBbbCcc，必须遵从驼峰形式；静态常量属性全部大写下划线（_）分割，形如：AAA_BBB_CCC

aaa、Bbb、Ccc：表示功能的组合，可简称亦可全称，部分专用名词大小写等不做严格规定，尽量满足行业用语。

静态常量属性，如本类专属需要用 private 修饰，并且通 val 修饰禁止更改，禁止外访；如果很多类都需要访问，使用 public 修饰，并且写入专门存放共有属性的类中，并使用 val 修饰禁止修改。

属性前加注释，表名该属性的具体含义。

```
/**
 * @author shudj
 * @description 用户类
 * @time 2021-02-19 09:22
 */
class User {
  companion object {
     val CLASS_NAME: String = “0001”
```

```
    private val USER_ADDRESS: String = "xxx"
  }
  // 用户名称
  var username: String = ""
  // 用户年龄
  var age: Int = 0
  ...
}
```

第 10 章 JavaScript 编程规范

10.1 类命名

一般形式：AaaBbb.js，使用大驼峰命名规则。文件名称和类名称一致

Aaa：表示该文件主要用于哪个模块，如果用于公共模块请键入 Common，Bbb：表示该文件需要表示的类

类前加注释详尽注释

```
/**
 * @author shudj
 * @description 正则表达式使用
 * @time 2021-02-19 09:22
 */
class CommonReguler {
}
```

10.2 函数

一般形式：aaaBbb，使用小驼峰命名规则。

aaa：表示该文件主要用于哪个模块，如果用于公共模块请键入 common，Bbb: 表示该文件主要的功能

1.使用函数建立文件

```
/**
 * @author shudj
 * @description 正则表达式使用，使用函数建立文件
 * @time 2021-02-19 09:22
 * @param {aaa, bbb} [object,array] 某对象或者集合
 * @param ccc [string, number, boolean] 字符串，数字，布尔类型
 * @return any 返回的具体函数一
 */
function CommonReguler(aaa, ccc) {
}
```

2.类、函数体中的函数

```
/**
 * 测试某字段是否是空白
 * @param message 被测试的信息
 */
function testMessageIsBlank(message) {
}
```

10.3 Vue 视图文件

一般形式：aaa.vue；通过对应的路由关系,使用文件夹包裹该文件。

aaa：表示该文件展示视图的路由名。

10.3.1 Vue 组件文件

一般形式：aaa-bbb.vue、aaa-bbb-ccc.vue。

aaa：表示该组件用于哪个模块。

bbb：表示该组件的主要功能。

ccc：表示该组件在这个功能中承担的责任,没有不写。

10.4 变量命名

一般形式：aaaBbbCcc，使用小驼峰命名规则。

Note：

1、请使用 ES6 中的 let 代替 var；

2、请在初始化变量时为其赋值；

3、尽量避免使用全局变量；

```
// 注明该属性的具体含义
 let aaBbCc = “”
```

10.5 常量

一般形式：AAA_BBB_CCC，全部使用大写英文字母，单词之间使用下划线隔开。

Note:

1、禁止使用拼音；

2、禁止加入数字；

```
// 注明该属性的具体含义
const AA_BB_CC = “”
```