**编码准则**

# 文档目的

## 代码方面

1. 统一编码风格
2. 规范编码
3. 提高代码可阅读性
4. 提高代码可维护性
5. 将运行时错误遏制在编码及编译时
6. 提高代码运行效率

## 工作方面

1. 提高代码沟通效率
2. 在组内同事互助时最大程度减少阻碍

# 代码规范说明

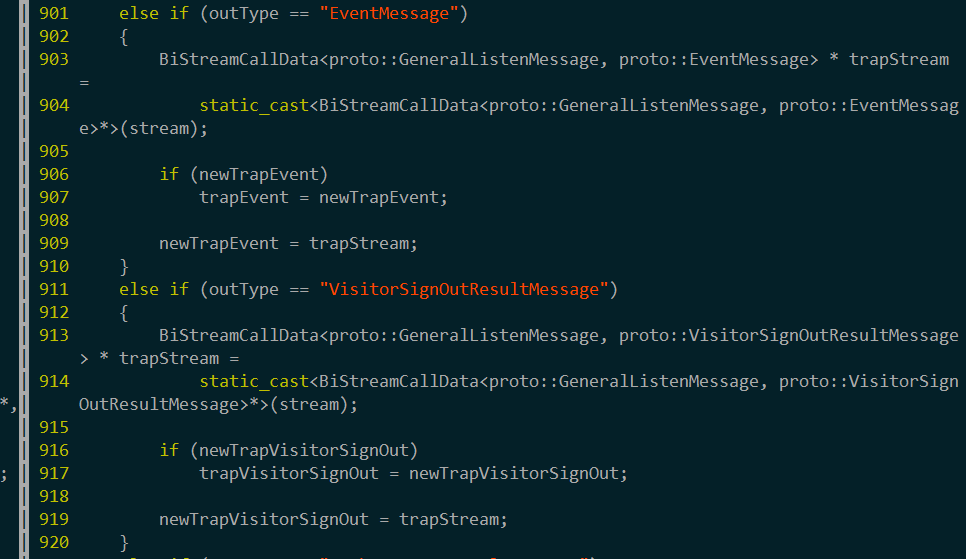
## 文件说明

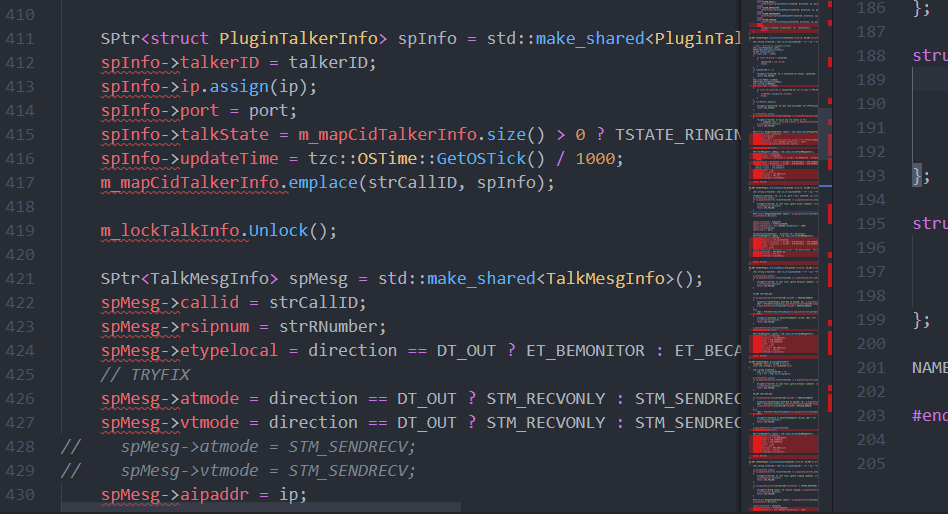
1. 代码行长度适宜，以不超过半屏为准；

导致出现这个问题的原因有命名太长、if层数过多。优化建议有使用类型别名、规范命名、优化代码逻辑。

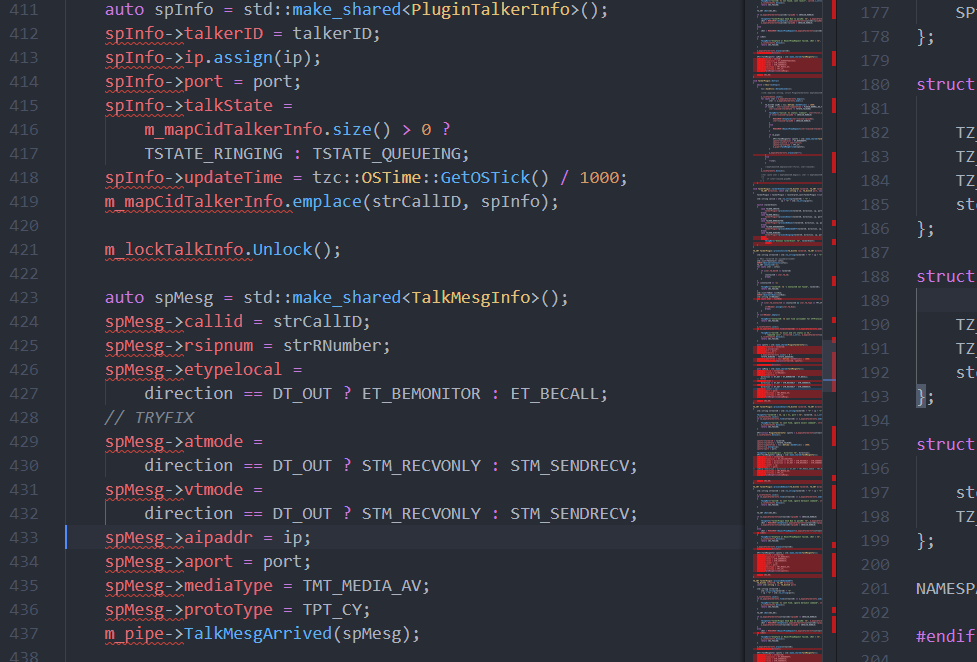
错误示例：







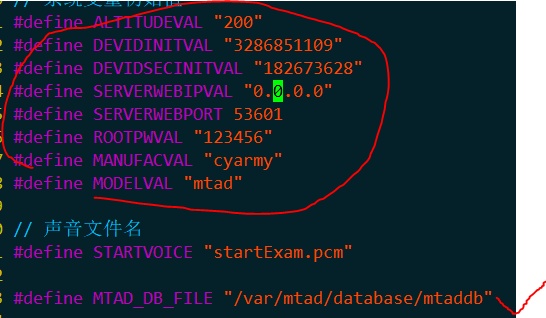
修正后：



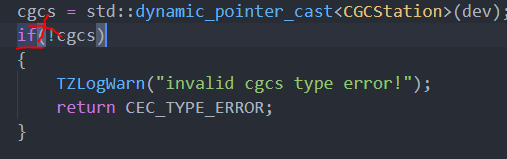
1. if、while等逻辑块后存在其他代码内容，在}后需换行和隔开；



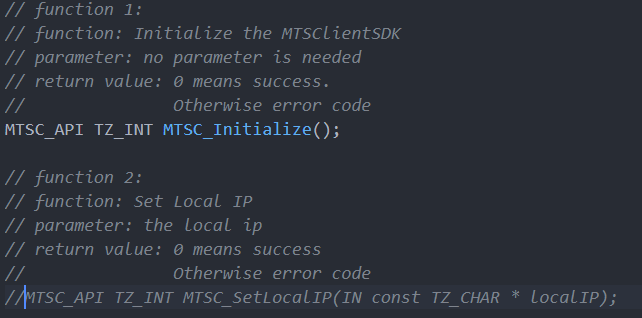
1. 非特殊情况，所有的预置基本类型需要使用common库中定义的TZ\_类型；
2. 所有文件均以空白行作为结束行
3. const值、宏定义、枚举值均采用大写字母书写，若包含多个单词，则不同单词间以"\_"隔开



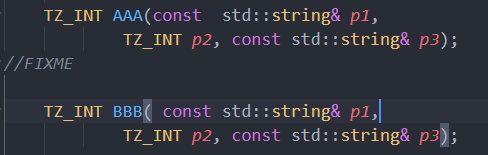
1. 函数名后紧跟圆括号；其余需要圆括号的地方空一格；括号内容紧贴括号
2. 双目运算符前后留出空格：= + - / \* | & || && == > < >= <= , ?:
3. if while for do-while 等逻辑操作与后面的括号需要以空格隔开；



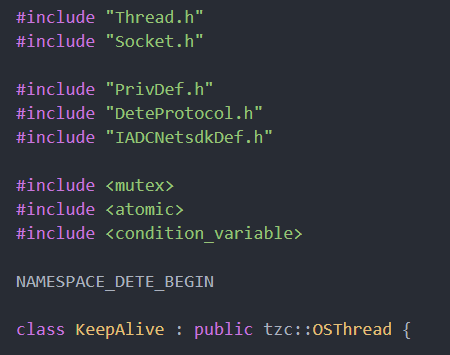
1. 代码注释说明采用双斜杠后增加空格（// ），注释代码采用双斜杠（//），便于风格统一和快速区分理解

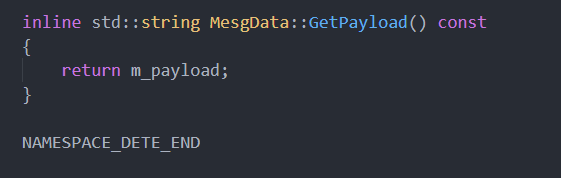


1. 长注释采用/\*\*/的方式，并且遵循1）中的规则
2. 无特殊情况，不允许出现多余空行，空格



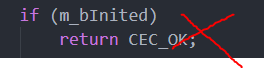
1. 命名空间采用宏定义方式，减少一行中无效空格所占比例
2. 命令空间前后需要换行



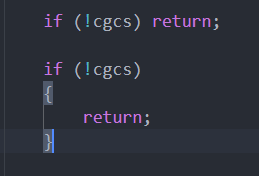


1. 条件语句后若是单行语句，需要换行编写必须增加{}，否则同行编写

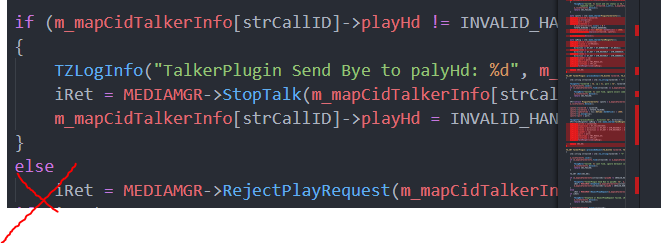
错误示例：



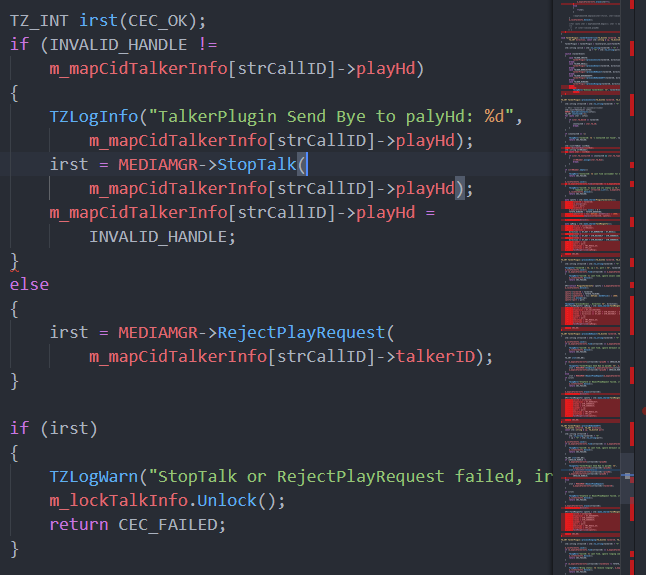
修正后：



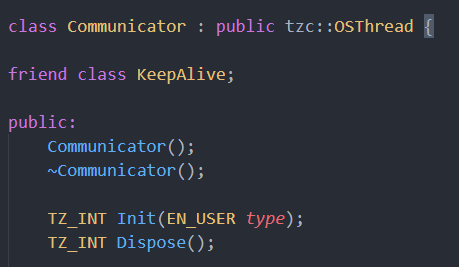
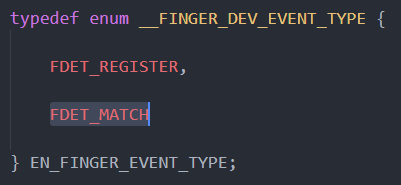
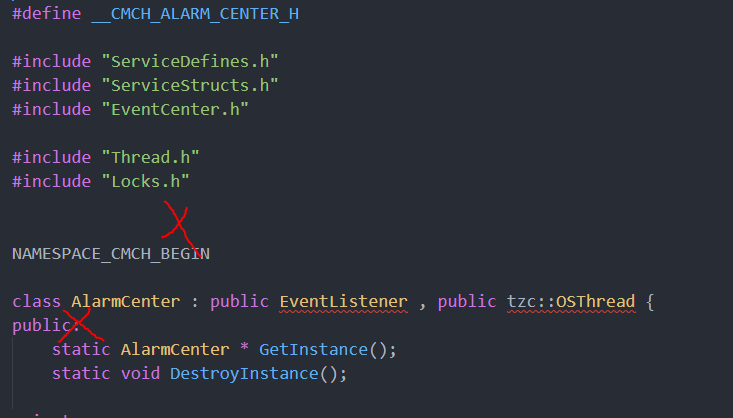
错误示例：

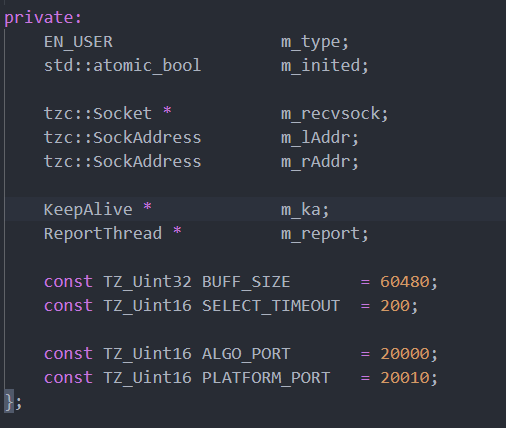


修正后：

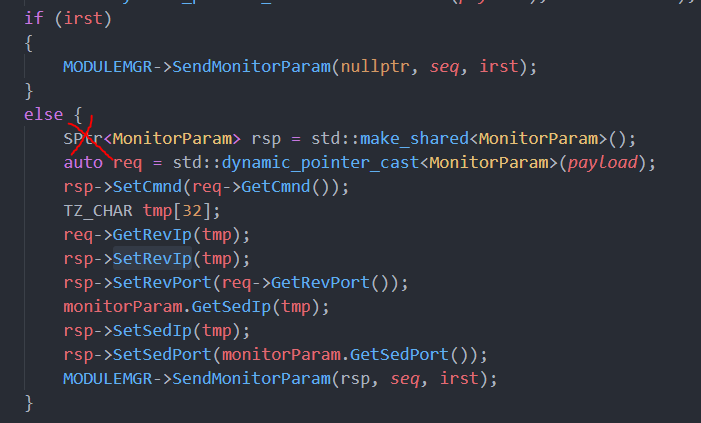


1. 类、结构体和枚举定义，括号和名称同行，且需要加上空格，并在与第一行内容之间加上换行符，表示内容开始；若是枚举定义，则最后一个枚举值和}之间需换行。



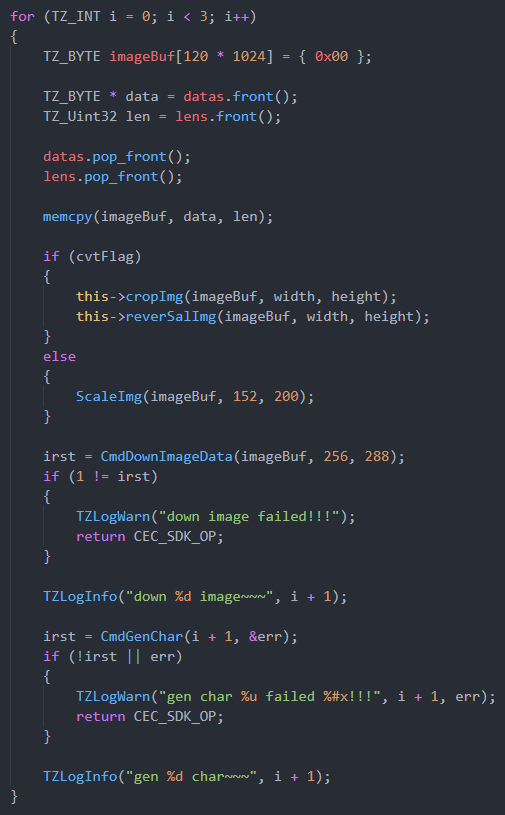
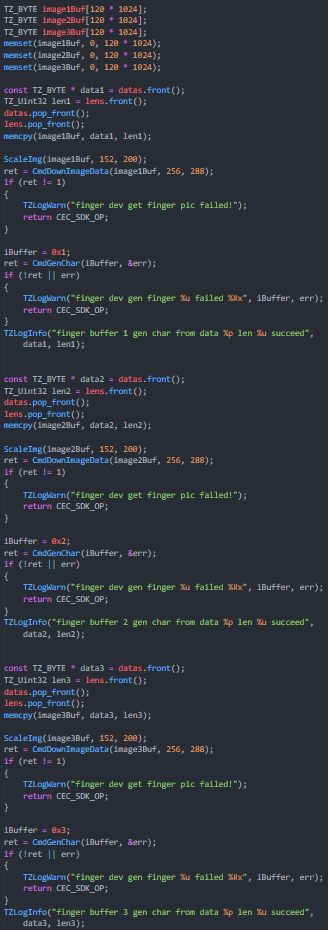


1. 函数实现，及条件语句的括号另起一行；

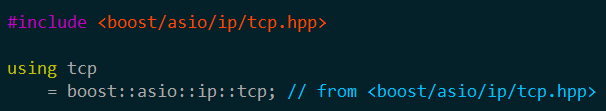


1. 变量名与函数名尽量简短，言简意赅，不允许使用a、b、c、1、2、3、old、new这种无意义的命名；较长的英文单次简写采用发音的字母；

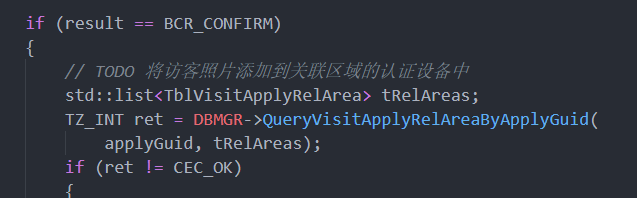




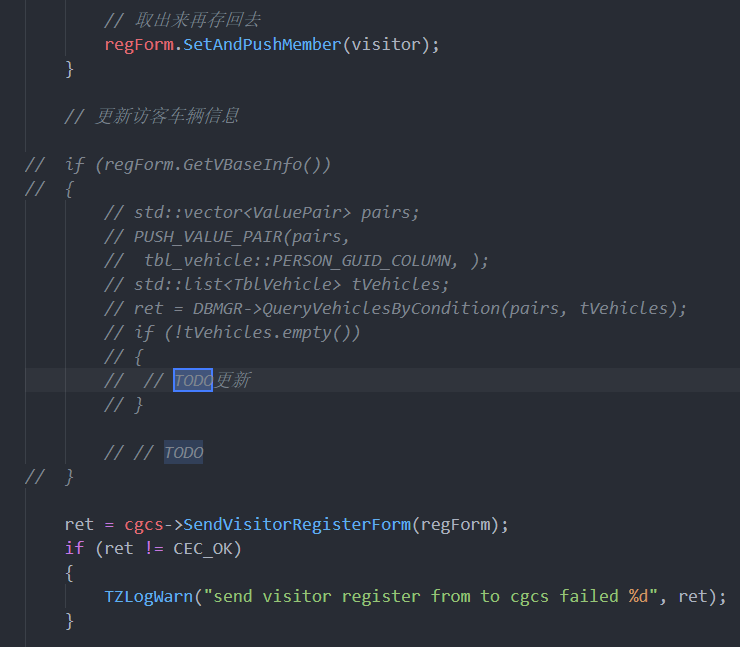
1. stl、common库等第三方库，需要带上类、枚举等所属的命名空间，不允许使用using namespace，防止出现名字相同导致的未知问题，增加代码可阅读性；命名空间过长的情况可以使用typedef、using和宏定义做别名处理；



1. 代码中未完成的部分用“// TODO”标注，并说明未完成的内容，在完成后删除“// TODO”标识；



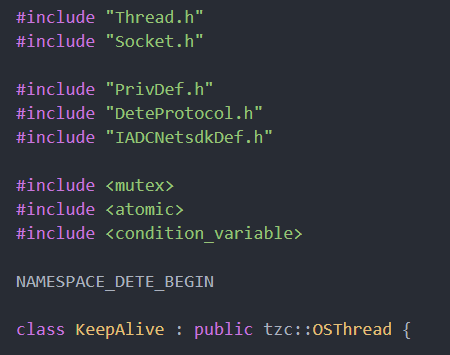
1. 注释的无用代码块在版本发布时直接删除，减少对代码阅读者的干扰；



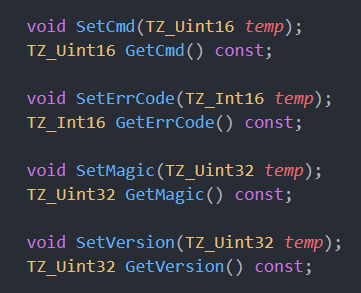
## 类、结构体、枚举说明

1. 头文件遵循最小包含原则，尽量将头文件包含在cpp中，减少相互依赖情况；
2. 头文件包含时，依据分类分块、由短及长原则，方便阅读；

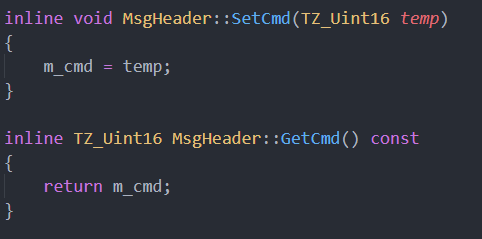




1. 类的public，private，protected修饰块顺序为public、protected 、private，各个修饰块的内部，采用virtual、static、普通函数的顺序排序
2. 类的成员变量尽量定义为private，并使用m\_开头命名（my\_），之后部分首字母小写；g\_（global）开头表示全局变量；局部变量采用驼峰命名；
3. 如外部需要访问类的成员变量，建立属性访问器；



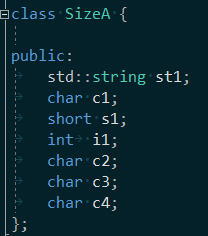
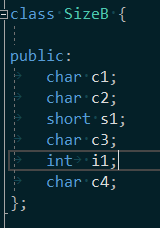
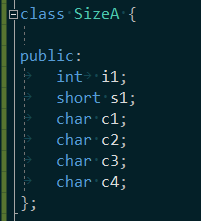
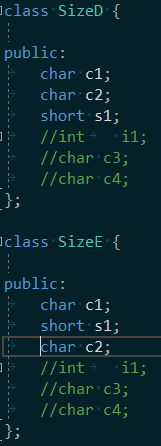
1. 属性访问器的Get方法，需要加上const修饰符。属性访问器实现采用inline函数；

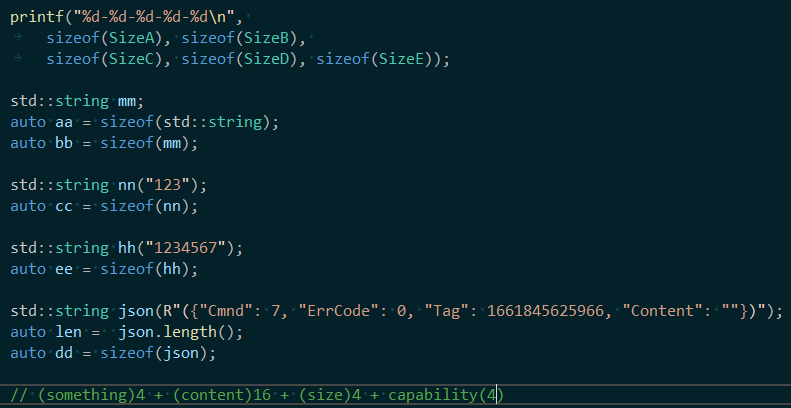


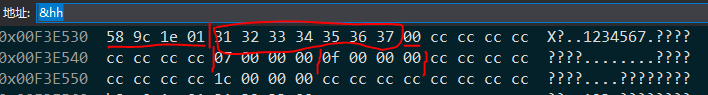
1. 类的private及protected成员函数首字母小写，public成员函数首字母大写；
2. 类的成员变量将相同类型的变量放置在一起，申明以整形（长度从小到大）、浮点型、指针、其他集合/锁等顺序排序；

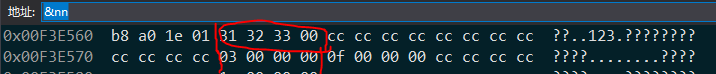
示例中相同字段不同的申明顺序，关注类的大小差异

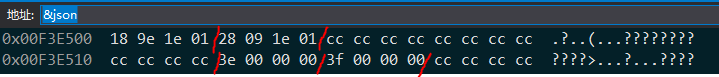
12-16-40-4-6

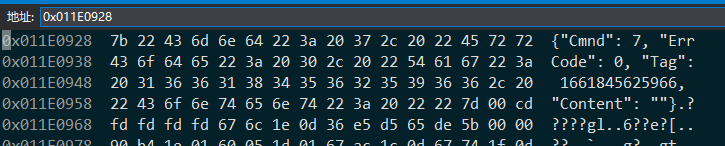
 

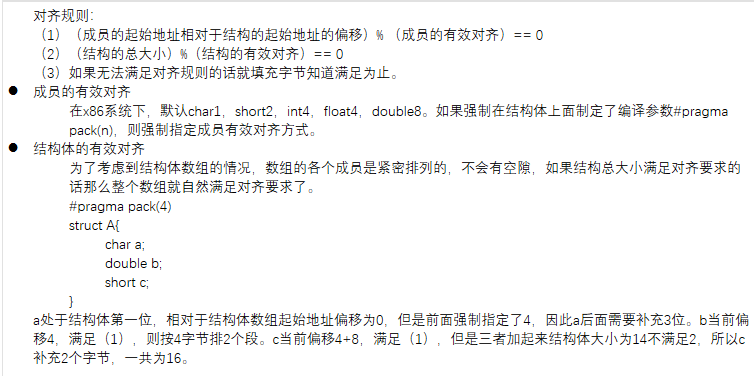




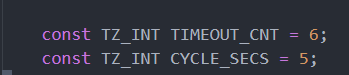




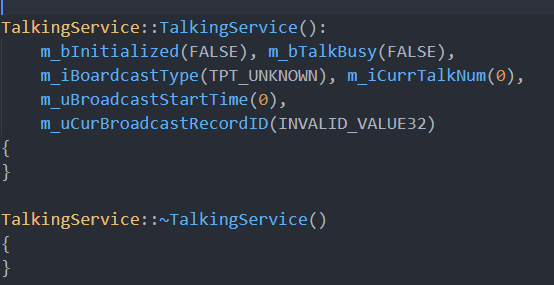




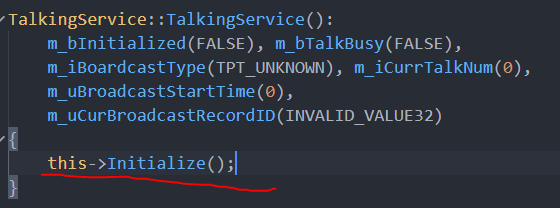
1. 类的const成员变量采用全大写命名，并放在所属块的最后；



1. cpp文件第一行需要include对应的.h文件，之后的include内容遵循1）和2）中规则；
2. cpp中函数以static、public、private顺序依次实现，各个部分的函数实现顺序顺序尽量和.h文件中保持一致；
3. 类的构造函数需要根据头文件中成员变量的申明顺序依次初始化，赋予初值，使类的实例在构造完成后处于初始化、可用状态；



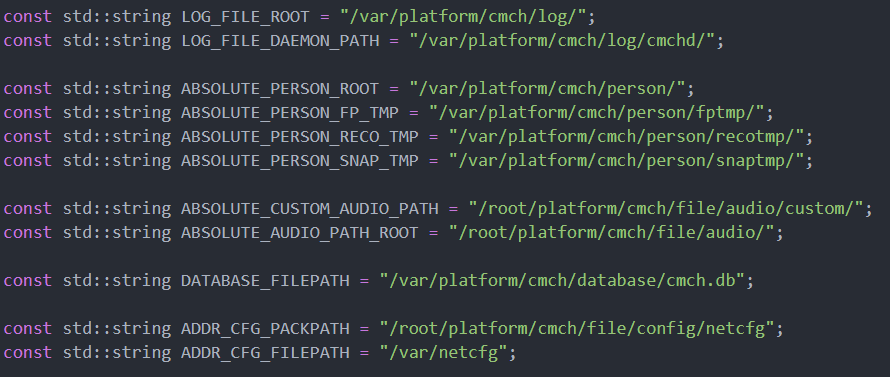
1. 非特殊需要，禁止在类的构造函数中调用其他函数；

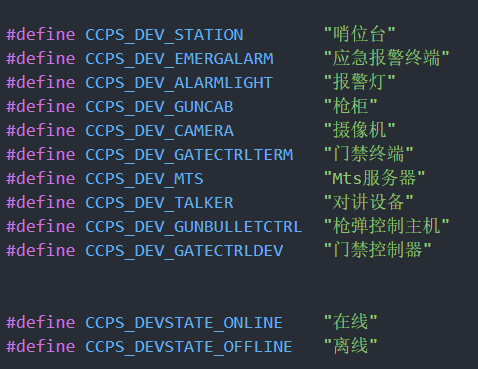


1. 基类的析构函数必须声明为virtual；

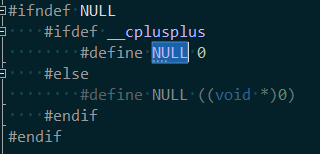
## 代码逻辑说明

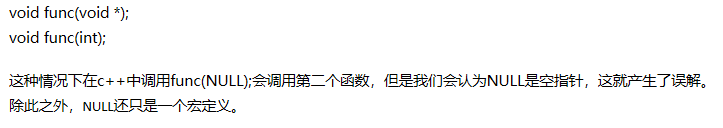
1. 日志打印时，Error、Warn使用!!!结尾，Info使用~~~或.结尾；
2. 能用const &传递的都需要加上const &；
3. 常量字符串用宏或者const std::string代替；



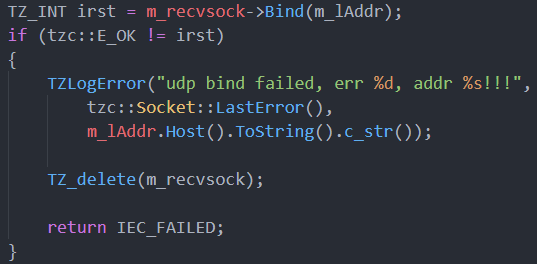
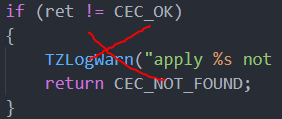


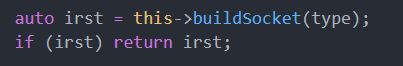
1. 使用c++11的nullptr关键字（std::nullptr\_t类型的constexpr变量）区分代替NULL，0。在c++中存在重载的情况；



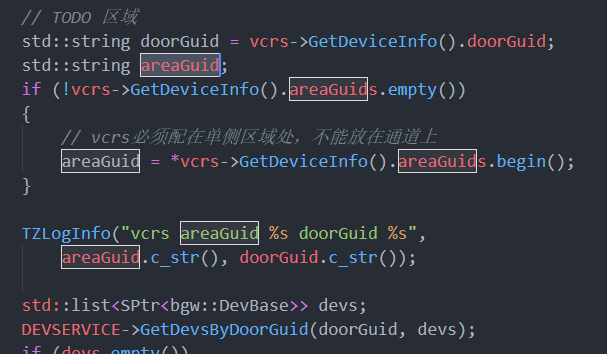


1. 用于存储返回值的变量，采用类型加读音简写方式命名，如irst、brst，分别表示整形result、bool类型result。
2. 判断==的逻辑时，常量需放在左边，变量在右边；为0判断可直接使用变量；

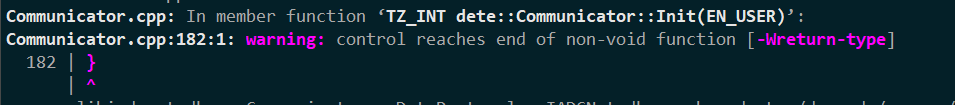


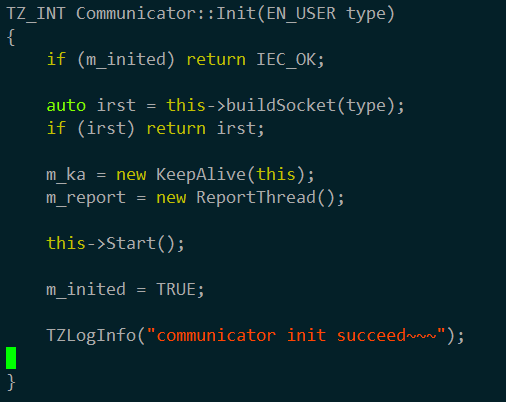


1. 尽量减少变量的生命周期，做到在使用时创建，无用时销毁，防止不必要的内存分配；

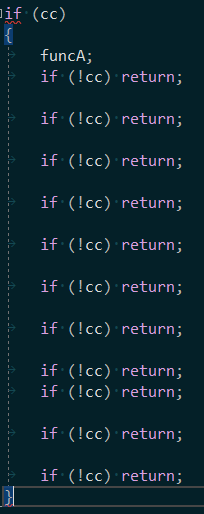
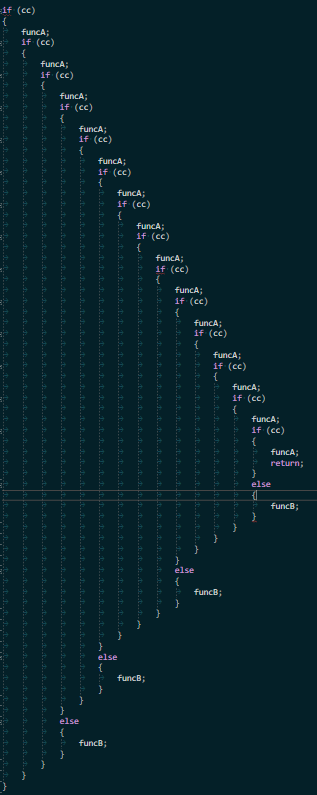


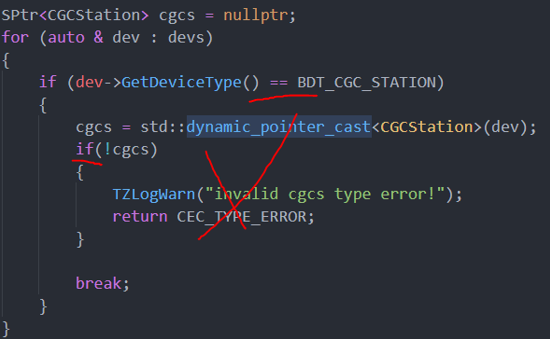
1. 在使用new的时候，要先写好TZ\_delete或TZ\_delete\_a，防止内存泄漏；
2. 在代码编译时，及时关注到打印的waring消息；

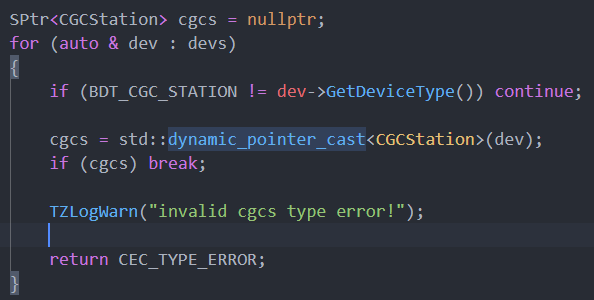


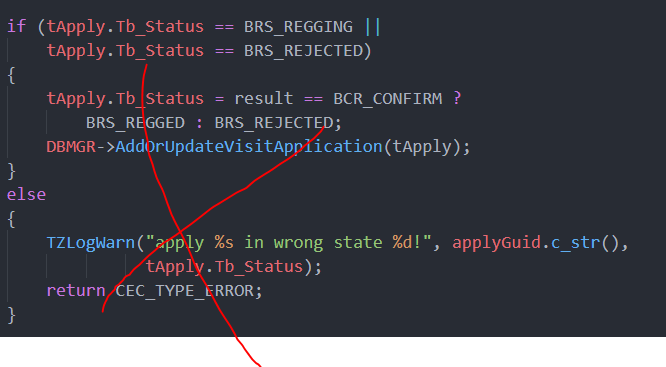


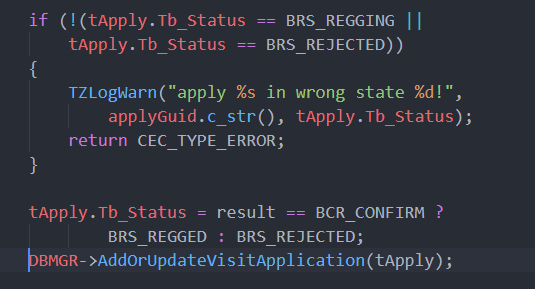
1. 为减少if及{}层数，先将异常情况、非法情况判断返回；



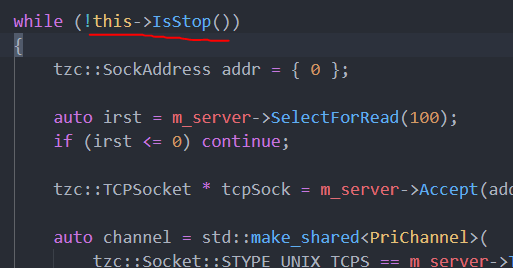








1. 在类的成员函数内部调用本类的成员函数，添加“this->”；



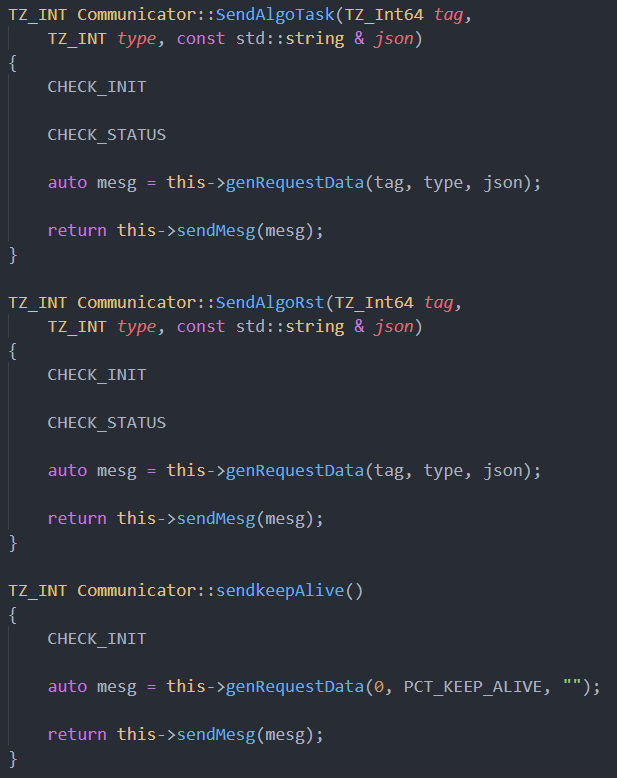
1. 代码块尽量短小，每个函数完成的职能尽可能的细化，函数封装要考虑到复用性以及功能性；

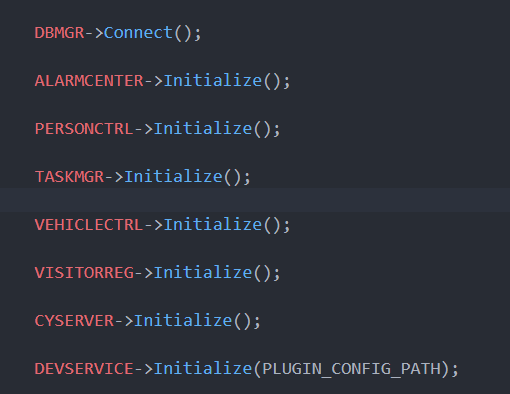
复用性保证函数在出现问题的时候，只用修改一处地方即可；

功能性体现函数职能，在函数调用层数多的情况下，代码阅读者能够专注到自己阅读的函数职能，而无需被无用信息所干扰，此情况可能出现该函数只被调用一次。

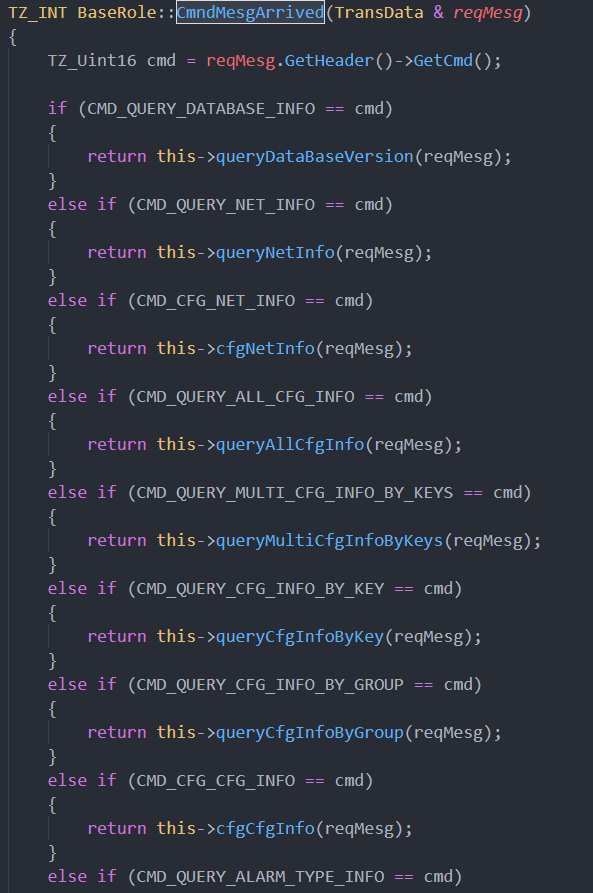
上述其实就是函数封装的好处。

什么情况下可以考虑封装？？











1. 看代码总结问题

为什么我们有那么多的管理类？？ 典型：SysHelper，FileMgr，ProtoCvt