C++/Qt 编码与评审规范(内部)V1.0

**引言**

本标准适用于C、C++/Qt 的客户端、测试与自动化项目。

**适用对象**

* 测试工程师
* 运维工程师
* 软件工程师

**规则目的**

“规范化的代码应当在第一眼就能看出意图、职责和边界，让任何接手者无需猜测即可开展评审与维护。”为了保证编写出的程序都符合相同的规范，保证一致性、统一性而建立的程序编码规范。

**版本记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **文档版本** | **文档编者** | **发布日期** | **修改说明** |
| V1.0 | Constantine | 20250926 | * 初稿拟定 |

**推荐资料**

华为 C 语言编程规范等业界优秀实践，结合 Qt/C++17 特性改写适用条款

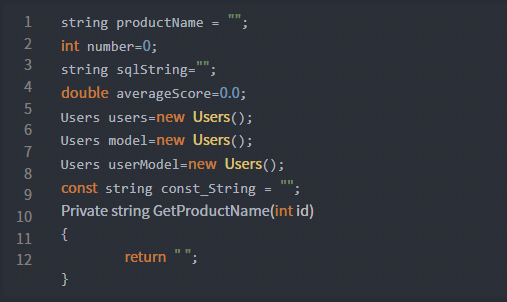
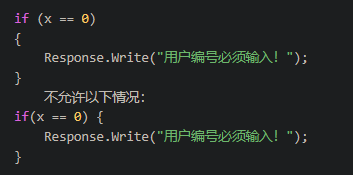
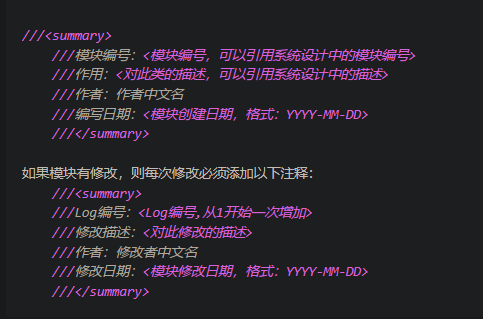
## 基本设计原则

* **可读性优先**：为后续维护与评审提供清晰意图，适度添加注释、设计描述和示例。
* **最小惊讶原则**：遵循业界和团队约定，避免奇特或隐式行为。
* **单一职责**：模块、类、函数应聚焦于单一职责，配合 SOLID 原则进行设计。
* **可测试性**：编写可被单元测试、集成测试覆盖的结构；解耦硬件I/O 依赖。
* **跨平台一致性**：Windows、Linux（如适用）需维护一致行为，避免平台条件编译带来行为差异。
* **简洁性：**通过编写简洁明了的代码来提升代码可靠性。废弃的代码(没有被调用的函数和全局变量)要及时清除，重复代码应该尽可能提炼成函数

## 项目结构与文件组织

* **目录层级**：遵循 src/, include/, tests/, resources/, docs/, tools/ 等标准布局。顶层目录保持职责单一，禁止在同一目录混杂接口、实现、测试。
* **模块划分**：以功能域拆分（如 Device/, Workflow/, UI/），公共组件common/、core/。
* **头文件/源文件配对**：ClassName.h/.cpp 成对出现；接口定义放头文件，内部实现置于.cpp。
  + 头文件仅暴露对外接口，内部工具函数、临时结构体禁止放在 .h 中。
  + 包含关系应“由不稳定指向稳定”：头文件尽量只依赖更通用/更稳定的模块，避免交叉依赖。
  + 必要时使用前向声明降低耦合；禁止在头文件中using namespace。
* **CMake/Qt 项目文件**：
  + CMakeLists.txt 按模块拆分，使用target\_link\_libraries、target\_compile\_definitions维护依赖。
  + Qt.pro/.pri文件需与CMake配置保持同步，避免重复或冲突。
* **资源与翻译文件**：图片、样式、QML放入resources/，通过qrc统一管理；字符串使用qsTr() 并维护.ts语言包。

## 命名与代码风格

* **命名约定**：
  + 类型（类、结构、枚举）使用PascalCase，命名应表达责任，如PressureController。
  + 函数、方法采用 camelCase，动宾短语表示动作，如 startMeasurement()。
  + 变量：局部变量使用 camelCase；类成员变量以 m\_ 前缀（只在成员上使用，避免与局部变量混淆）；静态成员以s\_前缀。跨线程访问的共享状态应以明显名称（如 m\_sharedState）突出用途。
  + 常量及枚举值使用 SCREAMING\_SNAKE\_CASE。
  + 类型/函数命名禁止使用无意义缩写（如 tmp, doStuff）；必要缩写需在注释中解释来源。
  + 布尔变量以 is/has/should/can 开头；集合使用复数名词（如 devices）。
  + 
* **格式化与排版**：
  + 统一使用UTF-8（无 BOM），行尾 LF。
  + 缩进4空格（团队约定），禁止 Tab 混用；请使用仓库根目录的 .clang-format配置（或 tools/style/clang-format.yaml）统一格式，提交前执行 clang-format/ninja format。
  + 每行不超过 120 列，表达式适度换行；链式调用分行对齐。
  + 不在同一文件内混用 CRLF；提交前执行 git diff --check 确保无尾随空格。
  + 块结构遵循“左对齐，右移”原则，if/else/while 块必须使用 {}，即便只有一行。
  + 
* **注释**：
  + 使用 // 单行、/\* ... \*/ 多行，Doxygen 注释 /// 或 /\*\* \*/ 描述公共接口。
  + 注释描述“为什么”而非“做什么”，避免陈述代码本身。
  + 文件头注释需包含版权信息、作者/维护人、最后更新时间、用途说明。
  + 公共 API 在头文件撰写 Doxygen 注释；重要算法、状态机在实现文件追加流程说明。
  + 注释须与代码保持一致，删除废弃代码时同步移除对应注释。
  + 
* **头文件防护**：采用 #pragma once（首选）或传统 include guard。
  + include guard 命名遵循 PROJECT\_MODULE\_FILENAME\_H 模式。
* **注释说明：**  
  A. 代码中遇到语句块时必须添加注释（if,for,foreach,……）,添加的注释必须能够说明此语句块的作用和实现手段（所用算法等等）。 对一个数值变量采用不是0,-1等的数值初始化，给出选择该值的理由。  
  B. 尽量多点注释，就算能一目了然的命名最好也顺便写一写注释，方便以后接收的人能更容易理解程序（方便不太懂英文的程序员）。  
  C. 如果因为某种原因使用了复杂艰涩的原理，为程序配备良好的文档和更多的注释。

## 现代 C++ 语言特性

* **标准版本**：默认 C++17 及以上，启用 <filesystem>, <optional>, <variant> 等特性。
* **智能指针**：优先使用 std::unique\_ptr，必要时使用 std::shared\_ptr+ std::weak\_ptr，避免裸指针拥有权。
* **RAII**：使用构造/析构管理资源，例如 QScopedPointer, std::lock\_guard，禁止手写 new/delete。
* **auto 使用**：仅在类型显而易见、或冗长模板类型时使用；避免降低可读性。
* **枚举类**：使用enum class代替传统枚举，并提供转换函数或 QMetaEnum 集成。
* **constexpr/const**：常量、魔法数字使用 constexpr 或 const；函数尽可能声明 noexcept。
* **[[nodiscard]] 与 std::span**：返回值影响流程的函数务必使用 [[nodiscard]]；处理连续缓冲区时优先使用 std::span/gsl::span，同时注明 Qt 5/6 兼容策略。
* **结构化绑定 / if with init**：在遍历 Qt 容器、pair 返回值时可使用，注意编译器版本兼容。
* **宏使用限制**：
  + 禁止滥用宏定义常量或内联函数，优先使用constexpr、inline；确需宏时全部大写并添加注释。
  + 宏替换内容需加括号防止优先级问题；宏函数内部避免副作用。

## 内存与资源管理

* **对象生命周期**：
  + Qt对象若有父子关系，使用父指针自动管理；无父对象时使用智能指针。
  + 避免在栈上创建大型QObject 子类（包含信号槽），优先使用 std::unique\_ptr<QObject>。
* **容器选择**：
  + 优先使用标准容器；若需 Qt 特性（如隐式共享、信号槽自动更新）可使用 QVector, QMap 等。
  + 明确迭代器与索引安全性，避免越界、迭代器失效。
* **资源封装**：对外设句柄、文件、套接字等封装为 RAII 类，确保异常或早退路径自动释放。
* **内存泄漏检测**：启用 ASan/Valgrind（Linux）或 Visual Studio 内存诊断（Windows）进行验证。

## 错误处理与诊断

* **异常策略**：
  + 核心业务逻辑优先返回 Expected<T, Error>/std::optional/outcome::result 等显式错误模型。
  + Qt 信号槽或回调中避免抛出异常；如需，必须捕获并记录。
  + 要精确记录错误的所有可能细节，包括发生的时间，和相关方法，类名等。
  + 如果需要，为每个执行的任务编写单独的 try-catch 模块。 这将帮你找出哪一段代码产生异常，并给用户发出特定的错误消息。如果应用程序需要，可以编写自己的异常类。自定义异常不应从基类SystemException派生，而要继承于. IApplicationException。
* **错误码体系**：统一定义错误枚举或结构体（含 code, desc, severity）。
* **日志与追踪**：
  + 遵循 spdlog-logging-standard.md，通过统一封装宏记录上下文。
  + ERROR/FATAL 级别必须包含 module, event, traceId, error.code 等关键信息。
* **断言与验证**：
  + 使用 Q\_ASSERT, Q\_ASSERT\_X 或 gsl::Expects 确保前置条件；Release 中保留必要的运行时代码路径检查。

## QT框架专项规范

* **QObject 与信号槽**：
  + QObject子类需使用 Q\_OBJECT 宏，确保元对象系统生效。
  + 新 API 使用functor/lambda + connect(sender, &Sender::signal, this, [=]{}) 的强类型连接；禁止旧式字符串连接。
  + 跨线程连接需显式指定Qt::QueuedConnection并确认目标线程拥有事件循环。
* **内存父子关系**：创建 QObject时立即传入父对象，或调用setParent()；禁止在析构中手写删除子对象。
* **线程亲和性**：
  + UI 操作必须在主线程 (QApplication::instance()->thread()) 执行。
  + 使用moveToThread()时，仅在对象构造完成且未开始处理事件前调用；确保目标线程启动事件循环后再触发信号，线程结束时需回到原线程释放。
* **QML / Qt Quick**：
  + C++ 暴露 QML 类型需通过 qmlRegisterType/qmlRegisterSingletonType，避免直接暴露裸指针。
  + 在 QML 中调用 C++ 接口，应使用 async/await 或信号反馈，防止阻塞 UI。
* **资源加载**：使用QResource或QFile访问资源；路径统一使用正斜杠，禁止硬编码绝对路径。

## 多线程与并发

* **线程创建**：优先使用QThreadPool, QtConcurrent, std::async；避免裸 std::thread/CreateThread。
* **同步原语**：使用std::mutex, std::shared\_mutex, QMutex；搭配 RAII (std::lock\_guard)。
* **数据共享**：
  + 明确读写锁策略，避免双向锁导致死锁；跨线程访问UI/QObject 属性需通过信号槽或QMetaObject::invokeMethod。
  + 共享状态封装为线程安全类，禁止全局可变单例。
* **原子与无锁**：使用std::atomic 管理计数器、标志位；复杂无锁结构需评审通过并配备测试。

## UI 与交互层规范

* **界面布局**：
  + 优先使用Qt Designer +.ui或QML文件；禁止在C++中手写大量布局代码。
  + 使用布局管理器（QGridLayout, QVBoxLayout）确保缩放自适应；资源字符串从翻译文件加载。
* **高分屏适配**：统一启用 QGuiApplication::setAttribute(Qt::AA\_EnableHighDpiScaling)、Qt::AA\_UseHighDpiPixmaps（Qt 6 默认开启）；字体通过 QFontDatabase::systemFont(QFontDatabase::GeneralFont) 获取；在 UI 设计中使用布局而非固定像素计算。
* **状态管理**：
  + UI 控件状态与业务模型解耦，使用 Model-View、MVVM 模式；通过信号槽同步。
  + 复杂流程需有状态机（QStateMachine 或自定义 FSM ）并记录状态转换。
* **可用性与国际化**：
  + 文字使用qsTr()并在翻译文件中维护；UI 需适配中英文长度。
  + 重要操作提供确认与撤销机制；错误提示需包含诊断信息。

## 构建依赖与配置管理

* **CMake 规范**：
  + 使用FetchContent/find\_package管理依赖，禁止手工拷贝二进制到仓库根目录。
  + 所有目标显式声明target\_compile\_features、target\_include\_directories、target\_compile\_definitions。
  + Release 构建启用 -DNDEBUG、-O2，Debug 构建启用 -DDEBUG、/Zi（MSVC）。
* **Qt 工具链**：
  + 使用qt\_add\_executable/qt\_add\_qml\_module（Qt 6），启用 AUTOMOC/AUTORCC/AUTOUIC。
  + 确保windeployqt/macdeployqt步骤纳入CI/CD。
* **配置文件**：
  + 所有可调参数放入 JSON/YAML/INI 配置，并提供版本字段。
  + 配置结构定义集中管理，提供加载、校验、热更新机制。

## 测试与质量保障

* **测试层级**：
  + 单元测试（QTest, Catch2, GoogleTest 等）覆盖核心算法和业务规则。
  + 集成测试模拟设备/网络交互，使用 Mock 或仿真器隔离外部依赖。
  + UI 自动化测试（squish, Qt Test Events, pytest-qt）用于关键场景回归。
* **覆盖率目标**：核心模块 > 70%，关键路径（安全、计费、设备控制） > 90%。
* **软件启动安全性**：应用程序启动时作些“自检”并确保所需文件和附件在指定的位置。如果需要的配置文件找不到，应用程序需能自己创建使用默认值的一份。如果在配置文件中发现错误值，应用程序要抛出错误，给出提示消息告诉用户正确值。
* **静态分析**：启用clang-tidy, cppcheck, include-what-you-use；Qt 项目启用 lupdate、lrelease 检测翻译缺失。
* **CI/CD**：在流水线中执行构建、测试、打包，输出制品及测试报告；失败自动阻断。

## 安全与隐私要求

* **数据脱敏**：敏感字段遵循公司脱敏策略，与日志标准一致。
* **凭据管理**：禁止硬编码账号、口令、密钥；通过QSettings +OS凭据库或专用服务加载。
* **输入校验**：对外部数据（TCP/串口/MES）进行严格校验，避免注入。
* **权限控制**：界面和命令应按照角色、工位权限控制；日志记录权限变更。
* **依赖漏洞扫描**：定期检查第三方库版本，关注Qt安全公告。

## 性能优化与资源消耗

* **性能预算**：在PRD或设计阶段明确 CPU、内存、时延指标。
* **剖析工具**：使用 Perf, VTune, Visual Studio Profiler, Qt Creator Analyzer 定期分析热点。
* **懒加载与缓存**：
  + UI 组件惰性初始化，避免一次性加载全部页面。
  + 设备通讯结果可缓存，设置过期策略。
* **异步 I/O**：串口/网口采用异步读写；UI 线程不可阻塞，必要时展示进度/加载动画。
* **资源释放**：确保长时间运行后内存、句柄无增长；使用QTimer清理过期资源。
* **代码清理**：
  + 删除未使用的函数、变量、头文件包含；在提交前运行 include-what-you-use/clang-tidy 修剪死代码。
  + 重复逻辑提炼为公共函数或模板，避免“复制粘贴”式维护。

## 其他注意事项：

* 避免硬编码，不在代码中使用具体的路径和驱动器名，使用相对路径，并使路径可编程。
* 所有可供用户输入的字段值，必须需忽略前后空白后(不包含密码)；在对字段值进行有效性验证。对提交进数据库的内容必须进行SQL注入过滤与XSS过滤。
* 开发模式规范：
  + 在编程实践中尽量使用通用的设计模式，以提高代码的可维护性。
  + 使用面向对象编程的原则如高内聚、低耦合等。**代码评审清单**

|  |  |  |  |
| --- | --- | --- | --- |
| 检查项 | 关键问题 | 典型陷阱 | 评审记录 |
| 设计符合单一职责 | 模块边界是否清晰，依赖是否必要 | 类/模块耦合过紧，隐藏全局状态 | ☑/☐ 说明 |
| 命名与风格一致 | 符合命名约定与 clang-format 规则 | 匈牙利命名、蛇形混用，行宽超限 | ☑/☐ 说明 |
| 现代C++使用 | 智能指针、RAII 是否正确使用 | 裸指针泄漏、shared\_ptr环引用 | ☑/☐ 说明 |
| Qt信号槽正确 | 强类型连接？跨线程连接显式声明？ | 旧式 SIGNAL/SLOT 字符串、槽未解绑 | ☑/☐ 说明 |
| 线程安全 | 跨线程访问是否安全、锁粒度合理 | UI 线程阻塞、死锁、数据竞争 | ☑/☐ 说明 |
| 错误处理完备 | 是否返回/传播错误码，日志上下文齐全 | 静默失败、异常泄露到 Qt 事件循环 | ☑/☐ 说明 |
| 安全与隐私 | 是否存在敏感信息泄露、权限漏洞 | 凭据硬编码、日志泄露密码 | ☑/☐ 说明 |
| 测试覆盖充分 | 单元/集成测试、自动化脚本是否更新 | 漏测关键路径、测试数据过旧 | ☑/☐ 说明 |
| 性能与资源 | 是否评估性能影响，资源释放 | 循环中动态分配、内存增长 | ☑/☐ 说明 |
| 文档与注释 | 接口、协议、配置是否有文档更新 | 忘记更新 README/版本说明 | ☑/☐ 说明 |
| 基础风格守则 | 命名、注释、宏、头文件等是否符合团队规范 | 缺少文件头注释、宏滥用、头文件职责混乱 | ☑/☐ 说明 |

**提示**：评审记录需提交至代码平台或质量系统，确保问题整改可追踪。

## 附录：华为 C 语言编程规范整理与示例

本附录整理自华为 C 语言编程规范精华，结合团队 C++/Qt 实际改写。所有示例均使用现代 C++ 语法，强调“清晰第一、简洁为美、风格一致”。

A.1 总体原则

* **清晰性优先**：代码先写给人看，再写给机器执行；必要时牺牲少量性能换取可读性。
* **简洁可维护**：删除死代码，提炼重复逻辑；保持函数短小，平均不超过 80 行。
* **风格统一**：继承所在模块既有风格，必要时通过 clang-format & .editorconfig自动化修正。



A.2 头文件与模块边界

* 头文件仅暴露接口，任何内部实现细节移动到.cpp。
* 遵循“稳定依赖原则”：业务层引用基础层，禁止反向依赖。
* 尽量使用前向声明，降低编译依赖；禁止在头文件引入不必要的大头文件链。



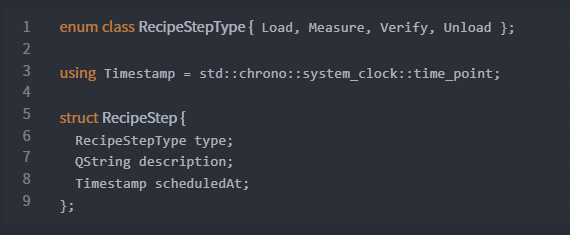
A.3 函数设计

* 单一职责，函数名体现动作；避免深度嵌套，超过 3 级即应拆分。
* 输入参数使用 const& 或 std::span，输出通过返回值或结构体；禁止通过裸指针隐式返回。
* 严格校验参数合法性，必要时返回错误码或使用 Expected 类型。



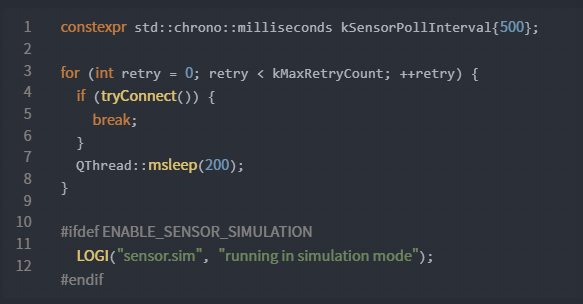
A.4 命名与类型定义

* 变量命名清晰表达含义，禁止tmp, data1等模糊词。
* 结构体、枚举命名以领域名开头：RecipeStep, UserRole；枚举值统一大写。
* typedef 更倾向 using；多层 using 统一收敛至一个头文件集中维护。



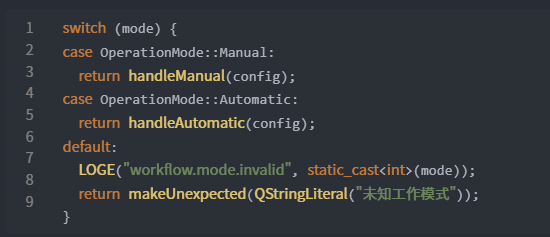
A.5 变量、常量与宏

* 变量定义就近，初始化即赋值；禁止先声明后赋值导致未定义行为。
* 常量使用constexpr或const; 尽量使用具名常量替代魔法数字。
* 宏仅用于编译开关或条件编译；宏体必须加括号并解释目的。



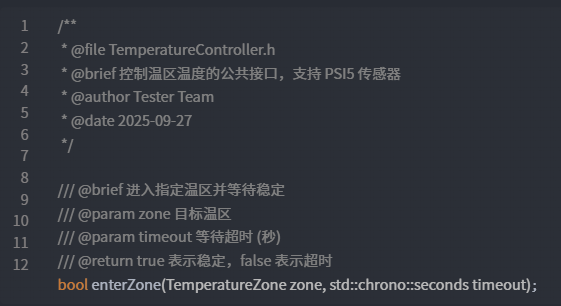
A.6 表达式与控制流

* 避免复杂表达式；对关键逻辑拆分临时变量。
* 条件表达式左值使用常量在前形式 if (0 == errorCode)，防止误写 =。
* switch 必须列出默认分支处理；使用 [[fallthrough]] 明确贯穿。



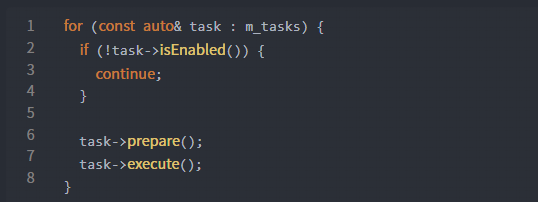
A.7 注释与文档化

* 文件头注释包含用途、作者、版权、历史版本；模块内维护 CHANGELOG。
* 对外接口使用Doxygen，参数、返回值、异常情况逐条说明。
* 对复杂状态机加入时序图或表格说明；注释需随代码更新。



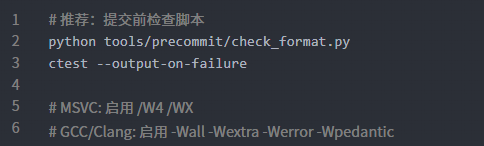
A.8 排版与格式

* 缩进统一4空格；花括号自立一行或与语句同行保持一致性。
* if/else、for 等语句无论执行体是否单行都必须加花括号。
* 保持逻辑块之间适当空行，分组相关语句。



A.9 编辑、编译与测试要求

* 所有提交必须通过 clang-format、clang-tidy、静态检查；禁止警告未清除。
* 编译开关按平台分类维护，如 config/compiler/msvc.cmake、config/compiler/gcc.cmake。
* 单元测试覆盖率写入 CI，核心模块不低于 70%。



A.10 常见反例速查

* **反例 1：头文件实现** —— 在 .h 编写完整函数导致重复编译。
* **反例 2：宏副作用** —— #define MAX(a,b) a > b ? a : b 遇到自增表达式出错。
* **反例 3：隐式类型转换** —— 混用窄宽整数字段，未显式static\_cast。
* **反例 4：注释过期** —— 修改代码忘记同步注释，评审时必须指出。

