# 日志记录与治理标准（v1.1）

**适用范围**：Winnie/Tigger 等 C++/Qt 项目 **日志框架**：spdlog 体系 **设计原则**：Prod 环境以稳定性与可观测性为先，Dev/QA 环境以定位效率为先 **推荐资料**：spdlog 官方 Wiki（线程安全、异步日志、Formatter 等最佳实践）

## 1. 目标与原则

* **必要性优先**：仅记录对定位、监控、审计或业务分析有价值的信息
* **结构化优先**：输出 JSON 或标准键值对，便于机器解析与聚合
* **分级精准**：不同级别面向不同受众和场景，避免滥用高级别
* **安全合规**：不记录敏感信息；默认启用脱敏；符合公司与法规要求
* **上下文完整**：单条日志即可关联到具体运行实例与业务对象

## 2. 日志级别与环境策略

### 2.1 级别定义

|  |  |  |
| --- | --- | --- |
| 级别 | 用途 | 适用场景 |
| **TRACE** | 极细颗粒诊断 | 仅限本地或临时排障 |
| **DEBUG** | 开发调试信息 | 不影响业务理解 |
| **INFO** | 关键业务里程碑 | 状态变更、外部交互摘要 |
| **WARN** | 可恢复异常 | 降级、生僻但非致命情况 |
| **ERROR** | 功能失败 | 对结果产生影响，需人工关注 |
| **FATAL** | 无法继续运行 | 进程需退出或立即告警 |

### 2.2 环境默认阈值

可动态调高或按模块覆盖：

* **Dev**：DEBUG
* **QA**：INFO
* **Prod**：WARN

### 2.3 强制规则

* Prod 禁用 DEBUG/TRACE（除非临时白名单且带过期时间）
* ERROR/FATAL 永不采样且必须包含错误详情

## 3. 字段词典（Schema）

所有日志建议采用 JSON 单行输出，字段命名统一小驼峰。

### 3.1 基础字段

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 必填 | 说明 |
| ts | String | ✓ | UTC ISO8601 时间戳，例如  2025-09-22T03:10:12.345Z  （统一输出 UTC，追加  Z  后缀） |
| level | String | ✓ | 日志级别（TRACE/DEBUG/INFO/WARN/ERROR/FATAL） |
| module | String | ✓ | 模块名（如  Device.Pressure  ,  Workflow.Runner  ） |
| event | String | ✓ | 事件名（动宾短语，层级用点号，如  Label.Print.Start  ） |
| messageKey | String |  | 短语键，便于国际化/聚合 |
| message | String |  | 人类可读简述 |
| app | String | ✓ | 应用名（TesterFramework 等） |
| version | String | ✓ | 应用版本（语义化版本或构建号） |
| configVersion | String |  | 配置/配方版本号 |

### 3.2 关联与上下文字段

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| traceId | String | 链路追踪 ID（无分布式也可本地生成） |
| spanId | String | 跨度 ID |
| runId | String | 本次执行批次/流程 ID |
| workflowId | String | 流程/配方标识 |
| stationId | String | 工位标识 |
| siteId | String | 站点标识 |
| socketId | String | 插槽标识 |
| dutSn | String | DUT 序列号 |

### 3.3 业务与测量字段（按需）

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| zone | String | 温区（LOW/NORMAL\_LOW/HIGH/NORMAL\_HIGH） |
| tempC | Number | 温度（摄氏） |
| pressureKPa | Number | 压力（kPa） |
| cmd | String | 外设指令/动作摘要 |
| attempt | Number | 重试计数 |
| durationMs | Number | 耗时，使用单调时钟计算 |
| success | Boolean | 操作是否成功 |

### 3.4 错误与异常字段（错误时必填）

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| error.code | String | 错误码（内部/系统/设备） |
| error.desc | String | 错误描述（可为英文+可选本地化） |
| errno | String | 系统错误号（如 ENOENT） |
| syscall | String | 相关系统调用或外设命令 |

### 3.5 运行时字段

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| threadId | String | 线程 ID |
| processId | String | 进程 ID |
| host | String | 主机名/设备编号 |
| file | String | 源文件（可选，注意性能与体积） |
| line | Number | 源行号（可选，注意性能与体积） |

### 3.6 JSON 示例与命名建议

事件命名建议采用 模块.动作.结果 结构，例如 Workflow.Run.Start、Workflow.Run.Done；messageKey 使用同结构小写变体（如 workflow.run.start）。

**成功示例：**

{"ts":"2025-09-22T03:10:12Z","level":"INFO","module":"Print","event":"Label.Print.Start","runId":"r-20250922-001","stationId":"B01","cmd":"1 2 3","dutSn":"SN123","traceId":"c6b..."}

**失败示例：**

{"ts":"2025-09-22T03:10:15Z","level":"ERROR","module":"Print","event":"Label.Print.Failed","runId":"r-20250922-001","stationId":"B01","cmd":"1 2 3","error":{"code":"ENOENT","desc":"文件或目录不存在"},"dutSn":"SN123","traceId":"c6b..."}

## 4. 采样与限速

### 4.1 采样策略

仅针对高频成功/状态类事件，错误类事件不采样。建议在 config/logging\_config.json 中集中配置（示例）：

{"sampling":{"default":{"dev":10,"qa":50,"prod":100,"key":["module","event","stationId"]},"overrides":{"Workflow.Run.Done":{"prod":10},"UI.Tab.Switch":{"qa":20,"prod":50}}}}

配置中的数字表示“每 N 条保留 1 条”；key 定义哈希维度。ERROR/FATAL 默认不参与采样。

### 4.2 限速策略

使用令牌桶/滑窗算法：

* **示例配置**：同 module+event 每实例 100 条/分钟
* **超出处理**：聚合为摘要日志

{"event":"...","droppedCount":123,"window":"60s"}

## 5. 敏感信息治理

### 5.1 黑名单字段

避免记录以下敏感字段：

* **认证相关**：password, passwd, token, secret, apiKey, privateKey
* **用户信息**：cardNo, ftpUser, ftpPass
* **位置信息**：精确位置信息

### 5.2 脱敏规则

* **标识类信息**：只显示后 4 位，其余以 \* 替代
* **路径/命令**：去除凭据部分（如 URL 凭据段）

### 5.3 自动化防护

* **封装层处理**：对黑名单键统一脱敏
* **CI 扫描**：正则扫描常见敏感模式（JWT、AKSK、卡号 Luhn 等）
* 参考官方“Error handling”建议：在错误回调中禁止再写日志，避免泄露敏感信息

### 5.4 合规与审计

* **用途明确**：最小可用保留期
* **访问控制**：访问控制与留痕

## 6. 写入、轮转与保留策略

### 6.1 写入策略

* **异步写入**：使用 spdlog 异步 logger，固定大小队列
* **背压处理**：失败启用背压/降级
* **线程安全**：遵循官方“Thread Safety”指引，跨线程共享 sink 时使用 \*\_mt 版本，禁止在多线程中复用 \*\_st

### 6.2 轮转策略

* **双重轮转**：按大小（例：50MB）+ 按日
* **保留策略**：保留 N 个或 N 天（Prod 建议 15~30 天）
* **压缩归档**：归档压缩为 .gz，落盘目录统一 logs/；初始化时若目录不存在需调用 std::filesystem::create\_directories

### 6.3 同步策略

* **Dev 环境**：更频繁 flush
* **Prod 环境**：批量 flush，异常自动 flush

### 6.4 时钟策略

* **时间记录**：统一记录 UTC
* **时长计算**：用单调时钟（避免系统时钟跳变）
* **日志元数据**：遵循官方推荐的 %Y-%m-%dT%H:%M:%S.%fZ 模式，确保跨平台解析一致性

## 7. 实施与落地（C++/Qt/spdlog）

### 7.1 结构化输出

* **方案 A**：nlohmann::json 组装再输出
* **方案 B**：自定义 spdlog formatter 输出 JSON
* 参考官方“Setting up JSON logging with spdlog”文档，优先使用 formatter 方式减少 JSON 序列化开销

### 7.2 封装宏（示意）

* LOGI(event, fields...) / LOGE(event, fields...) 自动拼入：traceId/runId/siteId/threadId/file:line
* 若在共享库或插件中使用，遵循官方“spdlog in DLLs”指导：每个模块独立创建 logger，并在卸载前调用 spdlog::drop()
* Logger 注册、获取或复用需参考“Logger registry”章节，避免重复创建导致的线程安全问题

### 7.3 动态配置

* **配置文件**：logging\_config.json：level, sampling, rateLimit, retention
* **热更新**：支持热更新触发（文件监控或 UI）
* 如需动态调级，可参考官方“Logger registry”章节，使用 spdlog::get() 获取已有 logger 并修改 level

### 7.4 采样/限速实现

封装层内置计数器与令牌桶；ERROR/FATAL 旁路

## 8. 评审清单（PR Gate）

|  |  |
| --- | --- |
| 检查项 | 要求 |
| **级别** | 是否合理使用  INFO/WARN/ERROR  ，Prod 是否可能产生高频 DEBUG |
| **字段** | 是否包含  runId/traceId/stationId/dutSn/zone  等必要上下文 |
| **敏感** | 是否可能输出敏感信息，脱敏是否到位 |
| **结构** | 是否为 JSON 单行，可被解析；是否通过 schema 校验 |
| **性能** | 是否考虑采样/限速；是否使用统一封装 |
| **运维** | 日志量是否可控；轮转/保留是否遵循标准；热更新/回滚是否验证 |

## 9. 验收与自测

### 9.1 单元测试

* **JSON 格式**：JSON 可解析、必填字段存在
* **脱敏功能**：脱敏函数对黑名单键的处理
* **采样限速**：采样与限速的统计正确性

### 9.2 自检脚本

扫描 logs/\*.log，统计各级别比例、Top 事件、异常比率、字段缺失率

## 10. 常用事件命名建议

|  |  |
| --- | --- |
| 模块 | 事件命名 |
| **Workflow** | Workflow.Load.Start/Done/Failed  ,  Workflow.Run.Start/Done/Failed |
| **Device** | Device.Conn.Open/Closed/Failed  ,  Device.Cmd.Send/Recv/Timeout |
| **Measurement** | Meas.Temp.Read  ,  Meas.Press.Read  ,  OutOfRange |
| **Recipe** | Recipe.Open/Save/Apply/Invalid |
| **MES** | MES.Upload.Start/Done/Failed/Retry |
| **UI** | UI.Tab.Switch  ,  Crash.Prevented |

## 变更记录

|  |  |  |
| --- | --- | --- |
| 版本 | 日期 | 变更内容 |
| **v1.1** | 2025-09-26 | 引入官方 Wiki 推荐实践：UTC 输出、采样配置示例、目录创建与评审清单 |
| **v1.0** | 2025-09-22 | 初版，覆盖等级/字段/采样/脱敏/轮转与 spdlog 落地建议 |

## 11. C++/Qt/spdlog 落地规范与代码模板

本章提供可直接落地的工程配置、初始化模板、封装宏、结构化（JSON）输出、采样/限速参考实现，以及 Windows/编码要点与 Review 清单补充。

### 11.1 工程与编译（CMake）

建议以包管理器（vcpkg/Conan）或已 vendor 的 spdlog 为准，统一版本；fmt 如非自带，请定义 SPDLOG\_FMT\_EXTERNAL 并统一外部 fmt 版本。

find\_package(spdlog CONFIG REQUIRED) add\_executable(app main.cpp) target\_link\_libraries(app PRIVATE spdlog::spdlog) # Release 裁剪编译期日志（仅编译到 INFO 及以上）target\_compile\_definitions(app PRIVATE SPDLOG\_ACTIVE\_LEVEL=SPDLOG\_LEVEL\_INFO) # 如使用外部 fmt（非必须）# target\_compile\_definitions(app PRIVATE SPDLOG\_FMT\_EXTERNAL)

### 11.2 初始化模板（控制台 + 滚动文件，异步）

#include<spdlog/spdlog.h>#include<spdlog/sinks/stdout\_color\_sinks.h>#include<spdlog/sinks/rotating\_file\_sink.h>#include<spdlog/sinks/daily\_file\_sink.h>#include<spdlog/sinks/dist\_sink.h>#include<spdlog/async.h>#include<filesystem>#include<fmt/chrono.h>namespace logging { inline spdlog::level::level\_enum parse\_level\_from\_env(){ try { if (constchar\* env = std::getenv("LOG\_LEVEL")) { std::string v(env); for (auto& c : v) c = (char)std::tolower(c); if (v == "trace") return spdlog::level::trace; if (v == "debug") return spdlog::level::debug; if (v == "info") return spdlog::level::info; if (v == "warn" || v == "warning") return spdlog::level::warn; if (v == "error" || v == "err") return spdlog::level::err; if (v == "critical" || v == "fatal") return spdlog::level::critical; if (v == "off") return spdlog::level::off; } } catch (...) {} return spdlog::level::info; // 默认 INFO } inlinevoidsetup(const std::string& app = "app", const std::string& version = "0.0.0", const std::string& log\_dir = "logs"){ // 线程池（队列 8192，1 个工作线程） spdlog::init\_thread\_pool(8192, 1); try { std::filesystem::create\_directories(log\_dir); } catch (const std::exception& e) { fprintf(stderr, "[spdlog] create log dir failed: %s\n", e.what()); // 官方建议：此处仅打印到 stderr，避免递归写入 spdlog } auto console = std::make\_shared<spdlog::sinks::stdout\_color\_sink\_mt>(); console->set\_level(spdlog::level::trace); // 滚动文件（大小 10MB，保留 5 个）auto rotating = std::make\_shared<spdlog::sinks::rotating\_file\_sink\_mt>( log\_dir + "/app.log", 10 \* 1024 \* 1024, 5, true); // 每日切割（00:00）auto daily = std::make\_shared<spdlog::sinks::daily\_file\_sink\_mt>( log\_dir + "/app\_daily.log", 0, 0, true, 5); auto dist = std::make\_shared<spdlog::sinks::dist\_sink\_mt>(); dist->add\_sink(console); dist->add\_sink(rotating); dist->add\_sink(daily); auto logger = std::make\_shared<spdlog::async\_logger>( "core", dist, spdlog::thread\_pool(), spdlog::async\_overflow\_policy::overrun\_oldest); spdlog::set\_default\_logger(logger); // 统一 pattern：时间 | 级别 | 线程 | logger | 消息 spdlog::set\_pattern("%Y-%m-%dT%H:%M:%S.%eZ | %^%l%$ | %t | %n | %v"); // 运行时日志级别（支持环境变量覆盖） spdlog::set\_level(parse\_level\_from\_env()); // 刷新策略 spdlog::flush\_on(spdlog::level::err); spdlog::flush\_every(std::chrono::seconds(2)); // 错误处理（避免递归写日志） spdlog::set\_error\_handler([](const std::string& msg){ // TODO: 上报/计数/降级；避免在此回调中再次写 spdlogfprintf(stderr, "[spdlog error] %s\n", msg.c\_str()); }); SPDLOG\_INFO("app={} version={} logging initialized", app, version); } } // namespace logging

\*\*使用说明：\*\*程序入口尽早调用 logging::setup("TesterFramework", APP\_VERSION)；退出前可调用 spdlog::shutdown() 确保异步落盘。

### 11.3 使用姿势与宏封装

**基础宏（级别映射）：**

#include<spdlog/spdlog.h>#define LOGT(...) SPDLOG\_TRACE(\_\_VA\_ARGS\_\_)#define LOGD(...) SPDLOG\_DEBUG(\_\_VA\_ARGS\_\_)#define LOGI(...) SPDLOG\_INFO(\_\_VA\_ARGS\_\_)#define LOGW(...) SPDLOG\_WARN(\_\_VA\_ARGS\_\_)#define LOGE(...) SPDLOG\_ERROR(\_\_VA\_ARGS\_\_)#define LOGF(...) SPDLOG\_CRITICAL(\_\_VA\_ARGS\_\_)

**结构化（JSON）输出示例（nlohmann::json）：**

#include<nlohmann/json.hpp>inlinevoidlog\_json(spdlog::level::level\_enum lvl, std::string module, std::string event, nlohmann::json fields){ using nlohmann::json; json j; j["level"] = spdlog::level::to\_string\_view(lvl); j["module"] = std::move(module); j["event"] = std::move(event); j["ts"] = fmt::format("{:%FT%TZ}", fmt::gmtime(std::chrono::system\_clock::now())); // 合并业务字段for (auto it = fields.begin(); it != fields.end(); ++it) { j[it.key()] = it.value(); } SPDLOG\_LOGGER\_CALL(spdlog::default\_logger\_raw(), lvl, "{}", j.dump()); } // 用法示例// log\_json(spdlog::level::info, "Print", "Label.Print.Start",// { {"runId","r-20250922-001"}, {"stationId","B01"}, {"dutSn","SN123"} });

**建议：**

* 头文件仅放宏与声明，初始化实现放 cpp；避免头文件里创建 logger
* 复杂对象需要格式化时，实现 fmt::formatter<T>，避免先行 to\_string() 带来不必要开销

### 11.4 采样与限速（参考实现）

**采样（每 N 条通过 1 条）：**

#include<unordered\_map>#include<mutex>#include<atomic>classSampler { public: boolaccept(const std::string& key, uint32\_t n){ if (n <= 1) returntrue; std::lock\_guard<std::mutex> lock(mu\_); auto& c = counters\_[key]; return (++c % n) == 0; } private: std::mutex mu\_; std::unordered\_map<std::string, uint32\_t> counters\_; };

**令牌桶限速（每秒 R 条，上限 B）：**

structTokenBucket { double tokens{0}, rate{10}, capacity{100}; std::chrono::steady\_clock::time\_point last{std::chrono::steady\_clock::now()}; boolallow(){ auto now = std::chrono::steady\_clock::now(); double dt = std::chrono::duration<double>(now - last).count(); last = now; tokens = std::min(capacity, tokens + rate \* dt); if (tokens >= 1.0) { tokens -= 1.0; returntrue; } returnfalse; } };

在业务日志入口统一应用采样/限速；对 ERROR/FATAL 旁路（不采样/不限速）。结合官方“Async logging”“Flush policy”建议，在高并发场景需评估采样与异步队列的配合策略，确保 flush\_on(err) 生效。

### 11.5 Windows/编码与文件策略要点

* **编码设置**：控制台建议使用 UTF-8 代码页；文件日志统一 UTF-8（无 BOM）
* **Sink 选择**：控制台 sink 可用颜色；文件 sink 禁止颜色/ANSI 控制符
* **线程安全**：多线程环境使用 \*\_mt sinks；不要混用 \*\_st
* **异步处理**：异步模式下务必在退出前调用 spdlog::shutdown()，避免日志丢失
* **目录权限**：日志目录建议 logs/，需要可写权限；异常时降级到控制台并上报告警

### 11.6 Review 清单补充（spdlog 专项）

|  |  |
| --- | --- |
| 检查项 | 要求 |
| **初始化** | 是否集中初始化、设置默认 logger、异步线程池/溢出策略是否合理 |
| **级别** | 是否使用编译期裁剪（  SPDLOG\_ACTIVE\_LEVEL  ）+ 运行时阈值；热路径无滥用  DEBUG/TRACE |
| **格式** | pattern 统一；文件无颜色；时间/线程/logger 信息齐全；必要时结构化 JSON |
| **输出** | 滚动/按日策略与保留一致；  flush\_on(err)  与周期刷新是否配置 |
| **并发** | 统一  \*\_mt  ；共享 sink 组织合理；错误回调无重入日志 |
| **安全** | 敏感字段已脱敏；无密钥/口令/证件号等输出 |
| **性能** | 避免在被裁剪级别内做重计算；复杂对象使用  fmt::formatter |