PR #7190 完整报告

PaddlePaddle/FastDeploy

[Feature] implement log channel separation and request log level system

合并时间：2026-04-16 15:13

原文链接：http://prhub.com.cn/PaddlePaddle/FastDeploy/pull/7190

执行摘要

- 一句话：实现日志通道划分和请求日志分级系统，优化日志管理和可配置性。
- 推荐动作：该 PR 值得精读，特别是日志通道划分和分级系统的设计决策。关注 RequestLogLevel 枚举的使用、log_request 和 log_request_error 的实现方式（如级别过滤和错误处理），以及如何通过环境变量实现动态配置。同时，注意 review 中讨论的性能和兼容性权衡，为类似基础设施重构提供参考。

功能与动机

针对 FastDeploy 的日志系统进行优化，预计分 4 个 pr 完成。当前 pr 目标是实现日志通道划分、request.log 级别划分和聚合，以简化日志文件（十几个散落文件 → 几个核心文件）、聚合请求全链路日志（统一写入 request.log），并支持按需调整详细程度（通过 FD_LOG_REQUESTS_LEVEL 环境变量）。

实现拆解

1. 新增核心日志模块：在 fastdeploy/logger/ 下添加 request_logger.py 和 config.py。request_logger.py 定义 RequestLogLevel 枚举（LIFECYCLE=0、STAGES=1、CONTENT=2、FULL=3）和核心函数 log_request、log_request_error，实现级别过滤（_should_log）和内容截断（_truncate）。config.py 提供 resolve_log_level 和 resolve_request_logging_defaults 函数，解析环境变量。
2. 更新日志设置：修改 fastdeploy/logger/setup_logging.py 中的 setup_logging 函数，支持日志通道划分（main、request、console），构建默认配置（_build_default_config），并处理自定义配置文件。新增 fastdeploy/logger/logger.py 中的 _get_channel_logger 来管理通道日志器。
3. 迁移业务模块日志调用：在 fastdeploy/entrypoints/、fastdeploy/engine/、fastdeploy/scheduler/ 等模块中，将原有 api_server_logger、llm_logger 等调用替换为 log_request 或 log_request_error，确保请求相关日志聚合到 request.log。例如，在 fastdeploy/utils.py 的 handle_request_validation_exception 函数中更新错误日志以包含 request_id。
4. 清理旧代码：从 fastdeploy/utils.py 中移除 ColoredFormatter 和 DailyRotatingFileHandler 类，统一日志实现到 logger 模块，避免重复代码。

5. 添加测试配套：新增 tests/logger/test_request_logger.py、tests/logger/test_logging_config.py，修改 tests/logger/test_setup_logging.py，覆盖日志级别、过滤逻辑和配置解析。
6. 更新文档：同步环境变量文档（如 docs/usage/environment_variables.md）和日志使用说明，反映新配置选项和默认值变更。

关键文件：

- fastdeploy/logger/request_logger.py（模块 日志模块；类别 infra；类型 core-logic；符号 RequestLogLevel, _should_log, _truncate, log_request）：核心实现文件，定义请求日志级别枚举和核心日志函数，实现日志过滤和内容截断逻辑。
- fastdeploy/logger/setup_logging.py（模块 日志模块；类别 infra；类型 configuration；符号 setup_logging, _build_default_config）：关键配置文件，更新日志初始化逻辑，支持通道划分和默认配置构建，影响整个系统的日志设置。
- fastdeploy/utils.py（模块 工具模块；类别 infra；类型 refactor；符号 ColoredFormatter, DailyRotatingFileHandler, getattr, _compute_fn）：清理旧日志代码，移除重复实现，并更新错误处理以使用新日志函数，确保向后兼容。
- tests/logger/test_request_logger.py（模块 测试模块；类别 test；类型 test-coverage；符号 TestRequestLogLevel, test_level_values, TestShouldLog, test_disabled_returns_false）：重要测试文件，覆盖请求日志级别和核心函数的单元测试，确保功能正确性和边界条件。

关键符号：RequestLogLevel, _should_log, _truncate, log_request, log_request_error, setup_logging, _build_default_config, resolve_log_level, resolve_request_logging_defaults, get_logger, _get_channel_logger

关键源码片段

fastdeploy/logger/request_logger.py

核心实现文件，定义请求日志级别枚举和核心日志函数，实现日志过滤和内容截断逻辑。

```
from enum import IntEnum
from fastdeploy import envs

class RequestLogLevel(IntEnum):
    """请求日志级别枚举，支持 L0-L3 四级详细度"""
    LIFECYCLE = 0 # L0: 关键生命周期事件，如请求创建、完成、中止
    STAGES = 1 # L1: 处理阶段细节，如信号量获取、首令牌时间记录
    CONTENT = 2 # L2: 内容和调度信息，如请求参数、调度状态、响应内容
    FULL = 3 # L3: 完整数据，包括原始请求和响应

def _should_log(level: int) -> bool:
    """检查是否应记录给定级别的日志，基于 FD_LOG_REQUESTS 和 FD_LOG_REQUESTS_LEVEL 环境变量"""
    if int(envs.FD_LOG_REQUESTS) == 0: # 如果请求日志被禁用，直接返回 False
        return False
    return int(level) <= int(envs.FD_LOG_REQUESTS_LEVEL) # 仅当级别不超过阈值时记录
```

```
def log_request(level: int, message: str, **fields):
    """记录请求日志，支持模板字符串和字段替换，自动过滤级别并截断 L2 内容"""
    if not _should_log(level): # 先检查是否应记录
        return
    from fastdeploy.logger import _request_logger # 延迟导入以避免循环依赖
    if not fields: # 无字段时直接记录消息
        _request_logger.info(message, stacklevel=2)
        return
    payload = fields
    if int(level) == int(RequestLogLevel.CONTENT): # L2 级别需要对内容截断
        payload = {key: _truncate(value) for key, value in fields.items()}
    _request_logger.info(message.format(**payload), stacklevel=2) # 使用模板格式化日志
```

fastdeploy/logger/setup_logging.py

关键配置文件，更新日志初始化逻辑，支持通道划分和默认配置构建，影响整个系统的日志设置。

```
def setup_logging(log_dir=None, config_file=None):
    """设置 FastDeploy 日志系统，支持通道划分和自定义配置"""
    if getattr(setup_logging, "_configured", False): # 避免重复配置
        return logging.getLogger("fastdeploy")
    if log_dir is None:
        log_dir = getattr(envs, "FD_LOG_DIR", "log") # 使用环境变量或默认日志目录
    Path(log_dir).mkdir(parents=True, exist_ok=True) # 确保日志目录存在
    setup_logging._log_dir = log_dir # 存储日志目录供后续使用

    if config_file is not None and Path(config_file).exists():
        with open(config_file, "r") as f:
            config = json.load(f) # 加载自定义 JSON 配置文件
        # 合并备份计数等默认参数
        for handler_name, handler_config in config.get("handlers", {}).items():
            if "backupCount" not in handler_config and \
               ("DailyRotating" in handler_config.get("class", "") or \
                "LazyFileHandler" in handler_config.get("class", "")):
                handler_config["backupCount"] = int(getattr(envs, "FD_LOG_BACKUP_COUNT", 7))
        logging.config.dictConfig(config) # 应用配置
    else:
        # 使用默认配置，构建 main、request、console 通道的处理器
        log_level = resolve_log_level() # 解析日志级别（基于 FD_LOG_LEVEL 或 FD_DEBUG）
        default_config = _build_default_config(log_dir, log_level, int(getattr(envs, "FD_LOG_
        BACKUP_COUNT", 7)))
        logging.config.dictConfig(default_config)
    setup_logging._configured = True # 标记已配置
    return logging.getLogger("fastdeploy")
```

fastdeploy/utils.py

清理旧日志代码，移除重复实现，并更新错误处理以使用新日志函数，确保向后兼容。

```
async def handle_request_validation_exception(request: Request, exc: RequestValidationError):
```

```
    """处理请求验证异常，记录错误日志并返回 JSON 响应"""
    first_error = exc.errors()[0] if exc.errors() else {}
    loc = first_error.get("loc", [])
    param = loc[-1] if loc else None
    message = first_error.get("msg", str(exc))

    request_id = None
    try: # 尝试从请求体中提取 request_id，便于错误追踪
        body = await request.body()
        if body:
            import json
            body_json = json.loads(body)
            request_id = body_json.get("request_id")
    except Exception:
        pass # 忽略提取失败，使用默认值

    log_request_error( # 使用新日志函数，替换原有的 api_server_logger.error 调用
        message="request[{request_id}] invalid_request_error: {url} {param} {msg}",
        request_id=request_id or "unknown", # 传递 request_id 字段
        url=str(request.url),
        param=param,
        msg=message,
    )
    err = ErrorResponse(
        error=ErrorInfo(
            message=message,
            type=ErrorType.INVALID_REQUEST_ERROR,
            code=ErrorCode.INVALID_REQUEST,
            param=param,
        )
    )
    return JSONResponse(content=err.model_dump(), status_code=HTTPStatus.BAD_REQUEST)
```

评论区精华

- 默认日志级别变更：FD_LOG_REQUESTS_LEVEL 默认值从 0 改为 2（CONTENT 级别），被识别为 Breaking Change，可能导致日志量增加。开发者解释 level=2 对齐当前 info 级别日志，但建议在文档中明确说明。
- 错误日志设计：讨论错误日志是否受 FD_LOG_REQUESTS 开关影响；开发者确认 log_request_error 不分级别，始终记录到 request.log 并映射到终端和 error.log，而 log_request 受级别控制。
- 代码质量建议：多次提到日志格式不一致（应使用模板字符串而非 f-string）、缺少异常处理（message.format 可能抛出 KeyError）、性能问题（L2 级别截断可能对复杂对象产生开销），以及缺少 exc_info 参数支持异常堆栈。部分建议在提交中被采纳，如添加 RequestLogLevel 枚举。
- 测试覆盖：指出 log_request_error 函数缺少单元测试，开发者后续提交中可能补充，但上下文未明确显示。

- 默认日志级别变更的 Breaking Change 风险 (design): 开发者确认 level=2 对齐当前 info 级别日志输出，但建议在文档中明确说明变更，或考虑保持默认值为 0。最终默认值设为 2，文档已更新。
- 错误日志是否受 FD_LOG_REQUESTS 开关影响 (design): 设计维持不变，log_request_error 始终记录，而 log_request 受级别控制，确保错误可追踪。
- 日志格式和异常处理改进建议 (correctness): 部分代码在后续提交中修复格式，但未添加全局异常处理；开发者关注到性能问题，但未完全解决。
- 测试覆盖不足和性能风险 (testing): 测试在 PR 中部分补充，但性能问题被标记为低优先级因默认级别较高；开发者未明确承诺优化。

风险与影响

- 风险：- 回归风险：修改了 52 个文件，涉及多个业务模块的日志调用替换，可能引入 bug，影响现有日志输出或错误处理。
- 性能风险：在 fastdeploy/logger/request_logger.py 的 _truncate 函数中，L2 级别对复杂对象（如大型 dict）先调用 str(value) 再截断，可能产生大量临时字符串，影响性能，尤其在日志频繁时。
- 兼容性风险：FD_LOG_REQUESTS_LEVEL 默认值从 0 改为 2，属于 Breaking Change，用户未显式设置时日志量会显著增加，可能影响磁盘 I/O 和存储。
- 维护风险：日志实现统一到 logger 模块，但部分旧代码残留（如 utils.py 中的日志相关移除不彻底）可能造成混淆。
- 影响：- 用户影响：用户可以通过 FD_LOG_REQUESTS_LEVEL 环境变量灵活控制请求日志详细程度（0-3 级），但默认变更可能需要用户调整配置以避免日志爆炸；错误日志增强 request_id 便于问题追踪。
- 系统影响：日志文件从散落多个精简为核心几个（fastdeploy.log、request.log、error.log、comm.log），结构更清晰，利于监控和调试；可能因默认级别提高而增加日志写入量。
- 团队影响：代码更统一，减少重复日志实现，便于后续维护和扩展；需要更新相关文档和培训团队成员新日志系统的使用。
- 风险标记：默认值变更 Breaking Change，性能开销，测试覆盖不足，跨模块修改回归风险

关联脉络

- PR #7412 [PD Disaggregation] Enable PD deployment without Router: 同为基础设施改进 PR，涉及配置和测试更新，共享类似跨模块影响模式。
- PR #7369 [BugFix] fix tool call parser: 涉及 DataProcessor 和 APIServer 模块的日志调用修改，与本 PR 的日志迁移有技术关联。