中山大学计算机学院

软件工程课程项目

# LifeMaster配置与运维文档

项 目 名 称: LifeMaster

组 员 姓 名: 刘昊、彭怡萱、马福泉

: 林炜东、刘贤彬、刘明宇

专 业: 软件工程

课 程 教 师: 郑贵锋

起 始 日 期: 2025年3月1日

结 束 日 期: 2025年7月6日

学 院: 计算机学院

# 目录

# 1 概述

LifeMaster是一个集成待办事项管理、记账管理和手账管理功能的个人生活管理系统。本文档详细描述了LifeMaster系统的配置管理、部署架构、运维流程以及常见问题的解决方案，为系统的稳定运行和持续维护提供指导。

# 2 系统部署架构概述

LifeMaster采用经典的Web应用三层结构：

- **前端层**：使用HTML + CSS（Tailwind）+ JavaScript构建用户界面
- **后端层**：使用Flask + Python 3.8+提供API服务
- **数据库层**：使用MySQL 5.7+进行数据存储
- **部署平台**：使用阿里云服务器实现云端部署

系统通过浏览器访问，支持多平台兼容，用户可通过互联网随时随地使用LifeMaster管理个人生活。

## 2.1 架构优势

- **技术栈成熟稳定**：采用业界主流技术，社区支持活跃
- **开发效率高**：前后端分离，支持并行开发
- **扩展性好**：三层架构便于后续功能扩展和性能优化
- **运维成本低**：基于云服务器，便于管理和维护

# 3 配置管理与版本控制

## 3.1 源码托管

表 1: 源码托管配置详情

| 内容 | 工具/方式 | 说明 |
| --- | --- | --- |
| 源码托管 | Git + GitHub | GitHub地址：https://github.com/cornhub919/LIFEmaster 开发过程中所有文件、文档、脚本均纳入版本控制，采用feature-branch分支模型确保主分支稳定，支持多成员并行开发，所有功能需通过Pull Request操作提交，利于代码审查和协作开发。 |

### 3.2 配置文件管理

表 2: 配置文件管理详情

| 内容 | 工具/方式 | 说明 |
| --- | --- | --- |
| 配置文件 | .env, config.py, requirements.txt | 通过.env文件集中存放敏感配置信息（如数据库连接、JWT密钥），requirements.txt管理依赖包，config.py中实现对不同运行模式（开发、测试、生产）的配置切换，便于迁移部署，提升系统可移植性与安全性。 |

### 3.3 虚拟环境

表 3: 虚拟环境配置详情

| 内容 | 工具/方式 | 说明 |
| --- | --- | --- |
| 虚拟环境 | venv/conda | 后端环境统一为Python 3.8+，使用包管理工具管理虚拟环境，实现不同环境之间的隔离。推荐使用venv或conda管理Python依赖环境，确保不同机器部署环境一致，降低运行异常风险。所有依赖通过pip freeze固定版本，保障项目的可复现性。 |

### 3.4 版本控制策略

#### 3.4.1 分支管理

- **main分支**：主分支，保存稳定可发布的代码
- **develop分支**：开发分支，集成最新的开发特性
- **feature分支**：功能分支，开发具体功能模块
- **hotfix分支**：热修复分支，紧急修复生产环境问题

#### 3.4.2 提交规范

- 提交信息使用规范格式：`type(scope): description`
- 常用类型：feat（新功能）、fix（修复）、docs（文档）、refactor（重构）
- 每次提交包含完整的功能点，避免部分提交
- 重要变更需要详细的提交说明

## 4 持续集成与测试流程

构建测试流程具备自动化潜力，遵循以下标准流程：

## 4.1 开发流程

1. 开发成员在本地完成模块开发
2. 提交代码至GitHub分支
3. 发起Pull Request请求
4. 代码审查和讨论
5. 测试相关负责人进行功能验证

## 4.2 测试策略

- **单元测试：**测试个别组件和函数的功能
- **集成测试：**验证模块间的接口和数据流
- **冒烟测试：**验证核心流程是否通畅
- **关键场景测试：**添加任务、创建手账、多用户并发测试等

## 4.3 部署流程

1. 手动集成部署至阿里云端服务器
2. 在测试环境进行完整功能验证
3. 测试通过后发布到生产环境
4. 记录部署日志和回滚方案

## 4.4 自动化扩展

后续可实现扩展，使用GitHub Actions实现：

- 自动化代码质量检查
- 自动化测试执行
- 自动化部署流程
- 自动化通知机制

# 5 部署计划与方式

## 5.1 Web服务器配置

表 4: Web服务器部署详情

| 内容 | 说明 |
| --- | --- |
| Web服务器 | 使用Nginx作为反向代理服务器，监听特定端口。后端采用Gunicorn作为WSGI服务运行Flask应用，实现高并发请求处理。Nginx负责将静态资源请求直接处理，将API请求反向代理至Gunicorn，保证了响应效率与安全隔离。 |

## 5.2 静态资源管理

表 5: 静态资源配置详情

| 内容 | 说明 |
| --- | --- |
| 静态资源 | 所有前端HTML、CSS、JS文件存放于Nginx指定目录，使用gzip压缩加速加载，支持浏览器缓存配置，减少服务器带宽压力。 |

## 5.3 数据库配置

表 6: 数据库配置详情

| 内容 | 说明 |
| --- | --- |
| 数据库 | 远程部署MySQL，开放特定端口，配置防火墙规则确保安全访问 |

## 5.4 部署频率与策略

表 7: 部署策略详情

| 内容 | 说明 |
| --- | --- |
| 部署频率 | 按开发阶段迭代部署，每完成一次核心功能（如番茄钟、财务分析、社交分享）开发后即进行部署测试，确保部署进度与开发进度同步。采用”拉代码→虚拟环境创建→数据库迁移→服务启动”的标准化流程，降低部署出错率。 |
| 部署方式 | 手动部署（上传代码、虚拟环境安装依赖、重启服务）后续可考虑把部署流程标准化后封装为Shell脚本，一键完成部署，并记录每次上线时间点与变更日志。 |

## 5.5 标准化部署流程

### 5.5.1 部署前检查

1. 确认代码已通过所有测试
2. 备份当前生产环境数据

3. 检查服务器资源状况
4. 准备回滚方案

#### 5.5.2 部署步骤

1. 从GitHub拉取最新代码
2. 激活虚拟环境并安装依赖
3. 执行数据库迁移脚本
4. 更新配置文件
5. 重启Web服务和后端服务
6. 验证部署结果

#### 5.5.3 部署后验证

1. 检查服务状态和日志
2. 执行冒烟测试
3. 监控系统性能指标
4. 记录部署日志

# 6 运行常见问题与解决方案

## 6.1 常见问题汇总

表 8: 常见问题与解决方案

| 问题类型 | 原因与处理方式 |
| --- | --- |
| 数据库连接失败 | 检查MySQL服务状态和.env文件配置，确认数据库服务器地址、端口、用户名、密码等配置项是否正确 |
| 模块导入失败 | 使用`pip install -r requirements.txt`安装依赖，确认虚拟环境已正确激活 |
| 数据库重置需求 | 执行`drop_all() + create_all()`处理脚本，注意备份重要数据 |
| API接口请求失败 | 检查后端是否监听对应端口、Token是否过期、防火墙设置等 |
| 静态资源加载失败 | 检查Nginx配置、文件路径、权限设置等 |
| 性能问题 | 监控数据库查询效率、检查代码逻辑、考虑增加缓存机制 |

## 6.2 故障排查步骤

#### 6.2.1 服务无法启动

1. 检查错误日志文件
2. 验证配置文件格式

3. 确认端口占用情况
4. 检查依赖包版本兼容性
5. 验证数据库连接状态

#### 6.2.2 性能问题诊断

1. 监控服务器资源使用情况
2. 分析数据库查询性能
3. 检查网络延迟
4. 评估并发用户数量
5. 识别性能瓶颈点

#### 6.2.3 数据丢失问题

1. 立即停止相关操作
2. 检查数据库备份
3. 分析日志文件
4. 评估数据恢复可能性
5. 制定数据恢复计划

# 7 后续运维计划

## 7.1 运维项目规划

表 9: 后续运维计划详情

| 项目 | 说明 |
| --- | --- |
| 日志管理 | Flask开发环境默认记录日志；后期可引入logging模块按模块记录错误和访问日志，实现日志分级、轮转和归档 |
| 错误监控 | 由于项目无自动化告警设计，须通过日志观察或人工测试发现异常，计划引入监控工具实现实时告警 |
| 数据备份 | 后期可尝试使用MySQL定期导出策略，手动备份MySQL数据库，未来可启用定时脚本实现自动备份 |
| 安全维护 | 禁止公网暴露数据库端口，密码加密存储，启用HTTPS传输，定期更新安全补丁 |
| 兼容性 | Web端兼容Chrome、Firefox、Safari、Edge等主流浏览器，CSS与JS已适配多平台，无需后续维护，系统兼容性经过测试，已经兼容Windows、Linux主流平台 |

## 7.2 监控与告警

### 7.2.1 系统监控指标

- **性能指标**：CPU使用率、内存使用率、磁盘I/O、网络流量
- **应用指标**：响应时间、错误率、并发用户数、API调用频率
- **数据库指标**：连接数、查询时间、锁等待、慢查询
- **业务指标**：用户活跃度、功能使用率、数据增长量

### 7.2.2 告警策略

- **紧急告警**：服务宕机、数据库连接失败、严重错误
- **警告告警**：资源使用率过高、响应时间超阈值
- **信息告警**：用户行为异常、潜在安全风险

## 7.3 数据管理

### 7.3.1 备份策略

- **全量备份**：每周进行一次完整数据库备份
- **增量备份**：每日进行增量数据备份
- **实时备份**：重要操作实时同步到备份系统
- **异地备份**：定期将备份数据同步到异地存储

### 7.3.2 数据恢复

- 制定数据恢复标准操作程序
- 定期测试备份数据完整性
- 建立恢复时间目标（RTO）和恢复点目标（RPO）
- 培训运维人员数据恢复技能

## 7.4 安全维护

### 7.4.1 安全检查清单

- 定期更新系统和依赖包版本
- 检查和修复安全漏洞
- 监控异常登录和操作行为
- 审查和更新访问权限
- 加强密码策略和双因子认证

#### 7.4.2 安全事件响应

- 建立安全事件分级响应机制
- 制定应急处理流程
- 建立事件记录和分析制度
- 定期进行安全演练

### 7.5 性能优化

#### 7.5.1 优化方向

- **前端优化：**代码压缩、图片优化、缓存策略
- **后端优化：**算法优化、数据库查询优化、缓存机制
- **数据库优化：**索引优化、查询优化、分库分表
- **架构优化：**负载均衡、微服务拆分、CDN加速

#### 7.5.2 性能评估

- 定期进行性能压测
- 监控关键性能指标
- 分析用户反馈和体验数据
- 制定性能改进计划

## 8 总结

LifeMaster配置与运维文档为系统的稳定运行和持续发展提供了全面的指导。通过规范的配置管理、标准化的部署流程、完善的监控体系和及时的故障处理，确保系统能够为用户提供稳定、安全、高效的服务。

随着系统的不断发展和用户需求的变化，运维工作也需要持续改进和优化。团队将根据实际运行情况，不断完善运维流程，提升系统的可靠性和用户体验。