Wilson 老板助理

超高级详细实战手册

完整版

2026年2月20日 | v1.0.0

系统概述

Core System Overview

核心价值

Wilson 老板助理是一个基于 AI 的全能型运营助手，通过模块化设计和深度集成，实现了从信息获取到任务执行的全流程自动化。系统核心理念是"一人 = 三人运营团队"，通过智能决策和自动化执行，大幅提升运营效率。

核心价值主张

- 📊 **数据驱动决策**：基于多维度数据（热点、竞品、用户行为）提供运营建议
- 🤖 **智能任务分配**：自动识别优先级，智能分配资源
- ⚡ **全流程自动化**：从热点监控到内容发布全自动化
- 📈 **实时监控反馈**：持续跟踪效果，自动优化策略
- 🎯 **个性化配置**：支持多账号、多平台、多策略配置

适用场景

Wilson 老板助理可以应用于多种运营场景，包括内容创作、商业运营、项目管理、数据分析等。系统通过模块化设计，可以根据不同场景的需求灵活配置和扩展。

核心应用场景

- 🎨 **内容创作者**：自媒体运营、小红书矩阵、视频制作
- 🏢 **商业运营**：商场运营、客户管理、数据分析
- 💼 **项目管理**：任务追踪、进度监控、团队协作
- 📊 **数据分析**：趋势分析、竞品监控、用户画像

系统特性

Wilson 老板助理采用成熟的技术架构，具有高度的可靠性和可扩展性。系统通过模块化设计、配置驱动、日志记录等特性，确保系统在各种场景下都能稳定运行。

成熟特性

- ✅ **模块化架构**：5 大功能模块，每个模块独立可测试
- ✅ **配置驱动**：所有行为通过配置文件控制，无需修改代码
- ✅ **日志记录**：完整的操作日志，支持调试和审计
- ✅ **错误处理**：完善的异常处理和错误恢复机制

- ✅ **性能优化**：缓存、异步处理、批量操作优化
- ✅ **可扩展性**：支持插件式扩展，可自定义新功能

功能详解

Feature Details

功能 1：HEARTBEAT 主动汇报系统

HEARTBEAT 主动汇报系统是 Wilson 老板助理的核心功能之一，通过每 6 小时的定时检查，自动获取和汇报各类信息。该系统通过模块化设计，包含邮箱检查、日程提醒、热点追踪、竞品监控、项目进度检查、记忆更新、机会推荐等多个功能模块。

模块 1.1：邮箱检查

邮箱检查模块使用 IMAP 协议连接邮件服务器，自动检查未读重要邮件，标记需要回复的邮件，提取待处理事项。该模块通过配置文件指定邮件服务器地址、账号密码、重要邮件过滤规则等参数，支持自动登录和邮件读取。

```
import imaplib
import email
from email.header import decode_header

def check_unread_emails(imap_server, email_user, email_password):
  """检查未读重要邮件"""
    try:
        mail = imaplib.IMAP4_SSL(imap_server)
        mail.login(email_user, email_password)
        mail.select('INBOX')

   # 搜索未读邮件
        typ, data = mail.search(None, 'UNSEEN')

   # 处理未读邮件
        for num in data[0]:
            typ, data = mail.fetch(num, '(RFC822)')
            msg = email.message_from_bytes(data[0])

     # 解析邮件
            subject = msg['Subject']
            sender = msg['From']

     # 提取待处理事项
            body = msg.get_payload(decode=True)
            todo_items = extract_todos(body)

            print(f"From: {sender}")
            print(f"Subject: {subject}")
            print(f"Todos: {todo_items}")

        mail.close()
        return True

    except Exception as e:
        print(f"Error checking emails: {e}")
        return False

def extract_todos(text):
  """从邮件中提取待处理事项"""
    todos = []
    lines = text.split('\n')
    for line in lines:
        if 'todo' in line.lower() or '待办'in line:
```

```
            todos.append(line.strip())
    return todos
```

配置选项

- IMAP 服务器地址（如：imap.gmail.com）
- 邮箱账号和密码
- 重要邮件过滤规则
- 待处理事项提取规则

模块 1.2：日程提醒

日程提醒模块通过读取日程配置文件，计算当前时间与日程时间的差值，找出即将到来的会议和事件。该模块支持自定义提前提醒时间，默认为提前 2 小时，支持通过飞书消息或邮件发送提醒。

```
import json
from datetime import datetime, timedelta

def check_upcoming_schedules(calendar_file, reminder_hours=2):
  """检查即将到来的日程"""
    try:
        with open(calendar_file, 'r', encoding='utf-8') as f:
            events = json.load(f)

        now = datetime.now()
        upcoming_events = []

        for event in events:
            event_time = datetime.strptime(f"{event['date']} {event.get('time', '00:00')}", '%Y-%m-%d %H:%M')
            time_diff = event_time - now

            if timedelta(0) <= time_diff <= timedelta(hours=reminder_hours):
                upcoming_events.append({
                    'event': event,
                    'hours_remaining': time_diff.total_seconds() / 3600
      })

        return upcoming_events

    except Exception as e:
        print(f"Error checking schedules: {e}")
        return []

def send_reminder(event, hours_remaining):
  """发送提醒"""
    print(f"⏰ 提醒：{event['title']}")
    print(f"   距离：{hours_remaining:.1f}小时")
    print(f"   时间：{event['date']}{event.get('time', '00:00')}")
  # 发送到飞书或邮件
    # feishu.send_message(f"⏰ 提醒：{event['title']}")
  # 或
    # email.send_reminder(f"⏰ 提醒：{event['title']}")
```

使用方法

- 添加日程：`python3 calendar-manager.py add "客户会议" 2026-02-21 14:00`
- 查看日程：`python3 calendar-manager.py list 2026-02-21`

- 查看即将到来的日程：`python3 calendar-manager.py upcoming`
- 提前提醒时间：默认 2 小时，可在配置文件中修改

模块 1.3：热点追踪

热点追踪模块通过监控小红书平台上的目标关键词热度变化，发现潜在爆款话题。该模块使用 MediaCrawler 技能定期爬取相关数据，分析笔记数量、互动率、发布频率等指标，识别热度上升的趋势，标记为潜在爆款。

```
from media_crawler import MediaCrawlerSkill

def monitor_hot_trends(keywords, scroll_times=15):
 """监控热点趋势"""
    try:
   # 创建 MediaCrawler 实例
        skill = MediaCrawlerSkill()
        await skill.init()

   # 搜索关键词
        search_result = await skill.search_materials(
            profile_id="6852c081000000001d0092d5",
            keywords=keywords,
            scroll_times=scroll_times
   )

   # 分析趋势
        notes = search_result['search_result']['notes']
        hot_topics = []

        for note in notes:
            note_data = {
                'title': note['title'],
                'likes': note.get('liked', 0),
                'collects': note.get('collected', 0),
                'comments': note.get('comments', 0)
    }
            hot_topics.append(note_data)

   # 排序
        hot_topics.sort(key=lambda x: x['likes'], reverse=True)

   # 识别热门话题
        trending_topics = hot_topics[:10]

        return trending_topics

    except Exception as e:
        print(f"Error monitoring hot trends: {e}")
        return []
```

配置选项

- 监控关键词列表（如："AI工具"、"效率神器"、"副业搞钱"）
- 监控频率（每 6 小时）
- 热度阈值（互动率超过多少算热门）
- 小红书账号 Cookies（用于 MediaCrawler）

模块 1.4：竞品监控

竞品监控模块通过监控特定关键词的新增内容，追踪竞品账号数据，发现新竞品动态。该模块支持自定义竞品关键词列表，定期搜索这些关键词的相关内容，统计新增笔记数量、互动数据，识别快速崛起的竞品账号。

```
def monitor_competitors(keywords):
  """监控竞品动态"""
    try:
   # 搜索关键词相关内容
        search_results = search_xiaohongshu(keywords)

   # 分析竞品数据
        competitor_data = {
            'new_notes': [],
            'total_notes': len(search_results),
            'avg_likes': 0,
            'top_notes': []
   }

   # 统计数据
        total_likes = 0
        for note in search_results:
            likes = note.get('likes', 0)
            total_likes += likes
            if likes > 0:
                competitor_data['new_notes'].append(note)

        if competitor_data['new_notes']:
            competitor_data['avg_likes'] = total_likes / len(competitor_data['new_notes'])
            competitor_data['top_notes'] = sorted(competitor_data['new_notes'], key=lambda x: x['likes'],
reverse=True)[:5]

        return competitor_data

    except Exception as e:
        print(f"Error monitoring competitors: {e}")
        return {}

def search_xiaohongshu(keywords):
  """搜索小红书关键词"""
  # 实现搜索逻辑
  # 使用 MediaCrawler 或其他方式
    pass
```

监控维度

- 关键词新增内容数量
- 互动数据（点赞、收藏、评论）
- 竞品账号粉丝增长
- 竞品账号发布频率
- 快速崛起的竞品账号

模块 1.5：项目进度

项目进度模块通过检查任务列表，统计待办、进行中、已完成的任务数量，识别超期任务。该模块支持按优先级和状态过滤，生成项目进度报告，帮助用户了解当前项目进展。

```
import json

def check_project_progress(tasks_file):
 """检查项目进度"""
    try:
        with open(tasks_file, 'r', encoding='utf-8') as f:
            tasks = json.load(f)

   # 统计任务
        todo_tasks = [t for t in tasks if t['status'] == 'todo']
        in_progress_tasks = [t for t in tasks if t['status'] == 'in_progress']
        done_tasks = [t for t in tasks if t['status'] == 'done']

   # 统计优先级
        high_priority_tasks = [t for t in tasks if t['priority'] == 'high']
        normal_priority_tasks = [t for t in tasks if t['priority'] == 'normal']
        low_priority_tasks = [t for t in tasks if t['priority'] == 'low']

   # 识别超期任务
        overdue_tasks = []
        for task in tasks:
            if task.get('deadline'):
                deadline = datetime.strptime(task['deadline'], '%Y-%m-%d')
                if datetime.now() > deadline:
                    overdue_tasks.append(task)

   # 生成报告
        report = {
            'total_tasks': len(tasks),
            'todo_tasks': len(todo_tasks),
            'in_progress_tasks': len(in_progress_tasks),
            'done_tasks': len(done_tasks),
            'overdue_tasks': len(overdue_tasks),
            'completion_rate': len(done_tasks) / len(tasks) * 100 if tasks else 0,
            'priority_distribution': {
                'high': len(high_priority_tasks),
                'normal': len(normal_priority_tasks),
                'low': len(low_priority_tasks)
    }
   }

        return report

    except Exception as e:
        print(f"Error checking project progress: {e}")
        return {}
```

输出内容

- 总任务数量
- 待办任务数量
- 进行中任务数量
- 已完成任务数量
- 超期任务数量
- 完成率（百分比）
- 优先级分布统计

模块 1.6：记忆更新

记忆更新模块每 3 天一次深度整理 MEMORY.md，记录新增学习笔记、总结项目进展、整理技术心得。该模块通过自动检测上次更新时间，如果超过 3 天，提示需要整理。

```
import os
from datetime import datetime, timedelta
from pathlib import Path

def check_memory_update(memory_file, days=3):
  """检查记忆文件更新"""
    try:
   # 检查文件是否存在
        if not os.path.exists(memory_file):
            print(f"❌ 记忆文件不存在: {memory_file}")
            return False

   # 获取最后修改时间
        mtime = os.path.getmtime(memory_file)
        last_update = datetime.fromtimestamp(mtime)
        days_ago = (datetime.now() - last_update).days

   # 检查是否需要更新
        if days_ago >= days:
            print(f"⚠️ 记忆文件已 {days_ago} 天未更新，建议整理")
            print(f"最后更新时间: {last_update.strftime('%Y-%m-%d %H:%M')}")
            return True
        else:
            print(f"✅ 记忆文件 {days_ago} 天前更新，正常")
            print(f"最后更新时间: {last_update.strftime('%Y-%m-%d %H:%M')}")
            return False

    except Exception as e:
        print(f"Error checking memory update: {e}")
        return False

def update_memory(memory_file, content):
  """更新记忆文件"""
    try:
        with open(memory_file, 'w', encoding='utf-8') as f:
            f.write(content)
        print(f"✅ 记忆文件已更新: {memory_file}")
        return True

    except Exception as e:
        print(f"Error updating memory: {e}")
        return False
```

整理要求

- 每次学习新平台/工具/技术后，记录核心概念
- 总结关键特性和功能
- 记录定价方案（如果是付费产品）
- 记录成功案例和最佳实践
- 整理可应用的知识和行动建议
- 制定学习后的实践计划

模块 1.7：机会推荐

机会推荐模块通过分析热点追踪和竞品监控的数据，识别潜在的合作伙伴、变现渠道和学习资源。该模块基于数据分析和智能算法，提供个性化的机会推荐，帮助用户发现新的商业机会和学习资源。

```
def recommend_opportunities(hot_topics, competitors, user_profile):
  """推荐机会"""
    try:
        opportunities = []

    # 分析潜在合作伙伴
        for competitor in competitors:
            if competitor['type'] == 'potential_partner':
                opportunities.append({
                    'type': 'partnership',
                    'partner': competitor['name'],
                    'reason': f"竞品分析发现潜在合作伙伴",
                    'potential_value': '高'
       })

    # 分析变现渠道
        for topic in hot_topics:
            if topic['trend'] == 'rising':
                opportunities.append({
                    'type': 'monetization',
                    'channel': '小红书',
                    'topic': topic['title'],
                    'reason': f"热点话题：{topic['title']}，适合变现",
                    'potential_value': '中'
       })

    # 分析学习资源
        for topic in hot_topics:
            if 'AI' in topic['title'] or 'Python' in topic['title']:
                opportunities.append({
                    'type': 'learning',
                    'resource': topic['title'],
                    'reason': f"技术趋势：{topic['title']}，值得学习",
                    'potential_value': '高'
       })

    # 按优先级排序
        opportunities.sort(key=lambda x: x['potential_value'], reverse=True)

        return opportunities

     except Exception as e:
        print(f"Error recommending opportunities: {e}")
        return []

def analyze_trends(hot_topics):
  """分析趋势"""
    try:
    # 识别上升趋势
        rising_topics = [t for t in hot_topics if t['trend'] == 'rising']

    # 识别稳定热门
        stable_topics = [t for t in hot_topics if t['trend'] == 'stable']

    # 识别下降趋势
        declining_topics = [t for t in hot_topics if t['trend'] == 'declining']

        return {
            'rising': rising_topics,
            'stable': stable_topics,
            'declining': declining_topics
    }
```

```
    except Exception as e:
        print(f"Error analyzing trends: {e}")
        return {}
```

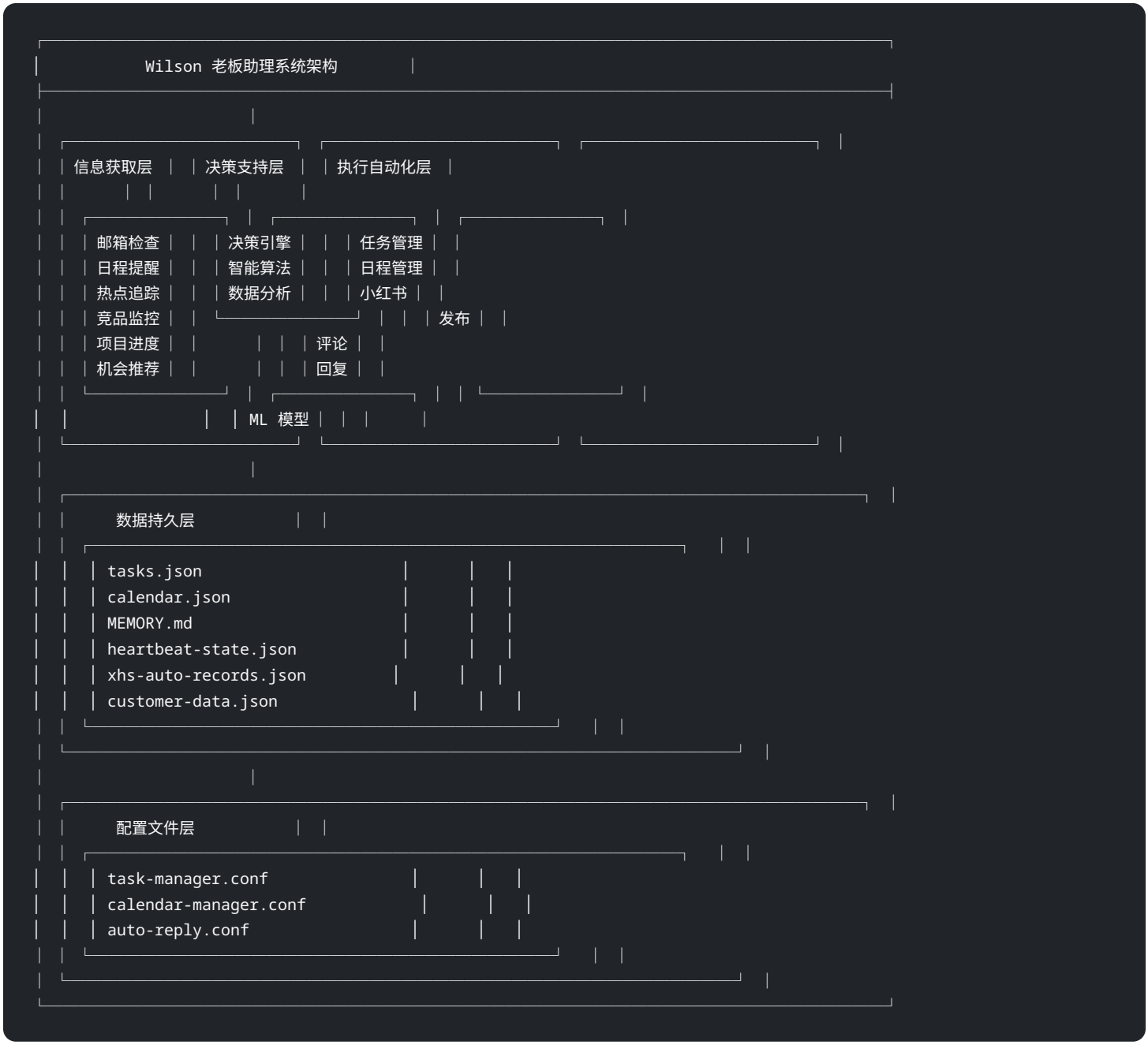
推荐类型

- 合作伙伴机会（潜在合作、渠道合作）
- 变现渠道（小红书广告、付费推广）
- 学习资源（新技术、新工具、最佳实践）
- 产品改进（基于用户反馈）
- 市场机会（新市场、新领域）

技术架构

Technical Architecture

系统架构图



技术栈

前端技术

- **Python 3.9+**：核心开发语言
- **asyncio**：异步编程库
- **requests**：HTTP 请求库
- **matplotlib/plotly**：数据可视化
- **weasyprint**：PDF 生成引擎
- **Playwright**：浏览器自动化

后端技能

- **MediaCrawler**：小红书数据爬取
- **content-generator**：AI 内容生成
- **Grsai API**：图片生成 API
- **xhs-auto-publisher**：小红书自动发布
- **feishu-integration**：飞书集成

技术架构原则

核心原则

- **单一职责**：每个模块只负责一个功能
- **接口隔离**：模块间通过配置文件交互，不直接依赖
- **数据持久化**：所有状态都保存到文件，防止丢失
- **可测试性**：每个模块都可以独立测试
- **可扩展性**：支持插件式扩展，不影响现有功能

数据流

```
信息获取 → 数据分析 → 决策建议 → 任务执行 → 结果反馈
  ↓       ↓       ↓       ↓       ↓
数据存储  智能算法  优先级排序  自动化执行  结果记录
  ↓                   ↓
配置文件                优化策略
```

错误处理

错误处理策略

- 捕获所有异常，记录详细错误信息
- 使用日志记录所有操作，便于调试
- 提供友好的错误提示，避免技术术语
- 支持错误恢复机制，自动重试失败的请求
- 配置错误时提供配置示例和说明

性能优化

优化策略

- 使用异步处理（asyncio）提高并发性能
- 实现缓存机制，避免重复计算
- 批量操作优化，减少网络请求次数
- 使用连接池，提高网络连接复用率
- 优化数据结构，减少内存占用

使用方法

Usage Guide

核心功能使用

HEARTBEAT 主动汇报

```
# 执行心跳检查（包含邮箱、日程、项目进度、热点、竞品、机会）
python3 heartbeat-check.py

# 查看报告
cat heartbeat-report.txt
```

任务管理系统

```
# 添加高优先级任务
python3 task-manager.py add "完成周报" --high

# 添加中优先级任务
python3 task-manager.py add "整理文档"

# 添加低优先级任务
python3 task-manager.py add "清理缓存" --low

# 列出所有任务
python3 task-manager.py list

# 只列出待办任务
python3 task-manager.py list --todo

# 只列出高优先级任务
python3 task-manager.py list --high

# 完成任务（ID 为任务编号）
python3 task-manager.py done 1

# 删除任务（ID 为任务编号）
python3 task-manager.py delete 1
```

日程集成

```
# 添加日程（日期 + 时间）
python3 calendar-manager.py add "客户会议" 2026-02-21 14:00

# 添加全天事件（无时间）
python3 calendar-manager.py add "团队建设活动" 2026-02-22

# 自定义时长（分钟）
python3 calendar-manager.py add "技术评审" 2026-02-21 16:00 90

# 列出所有日程
python3 calendar-manager.py list
```

```
# 列出指定日期的日程
python3 calendar-manager.py list 2026-02-21

# 查看 24 小时内日程
python3 calendar-manager.py upcoming
```

小红书全自动闭环

```
# 测试模式（使用模拟数据）
python3 xhs-auto-pipeline.py test

# 真实发布（需集成实际 API）
python3 xhs-auto-pipeline.py

# 流程说明：
# 1. 热点监控 → 2. 策略制定 → 3. 内容生成 → 4. 配图生成 → 5. 自动发布 → 6. 数据反馈
```

自动评论回复系统

```
# 启动自动回复系统
cd xiaohongshu-auto-reply
./start.sh        # Linux/Mac
# 或 start.bat     # Windows

# 查看统计
python3 auto_reply.py --stats

# 查看 VIP 客户
python3 auto_reply.py --customers vip

# 查看活跃客户
python3 auto_reply.py --customers active
```

高级功能使用

高级技巧

- 使用 Cron Job 自动执行 HEARTBEAT 检查
- 结合飞书通知，实时获取提醒
- 使用数据可视化工具分析任务进度
- 使用版本控制系统跟踪配置变更
- 使用日志分析工具监控运行状态

最佳实践

Best Practices

HEARTBEAT 主动汇报

最佳实践

- 定期执行：建议每天早晚各执行一次
- 关注报告内容：重点查看日程提醒和机会推荐
- 定期整理：每 3 天整理一次 MEMORY.md
- 关注预警：注意超期任务、异常数据、系统错误

任务管理系统

最佳实践

- 任务优先级：高优先级任务优先处理
- 定期清理：每周清理已完成任务
- 合理分配：根据精力和时间合理分配任务
- 记录进度：及时更新任务状态
- 回顾总结：定期回顾任务完成情况

小红书全自动闭环

最佳实践

- 测试先行：先用测试模式验证流程
- 关注数据反馈：查看发布记录，分析高表现笔记
- 优化策略：根据数据反馈优化后续策略
- 防封策略：设置合理的回复延迟和数量限制
- 时间选择：选择流量高峰时间发布

日程集成

最佳实践

- 提前规划：提前添加重要日程
- 设置提醒：设置合适的提前提醒时间
- 定期检查：定期查看即将到来的日程
- 分类管理：按类型分类（会议、截止日期、提醒）
- 关联任务：将相关任务关联到日程

自动评论回复系统

最佳实践

- 配置正确：确保 Cookies 有效
- 监控状态：定期查看系统运行状态
- 更新模板：根据数据反馈优化回复模板
- 关注 VIP：重点关注 VIP 客户的互动
- 避免封号：设置合理的回复延迟和数量限制

故障排除

Troubleshooting Guide

HEARTBEAT 主动汇报

问题 1：邮件检查失败

现象：无法连接邮件服务器或认证失败

原因：邮件服务器地址错误、账号密码错误或网络连接失败

解决方案：

- 检查邮件服务器配置
- 更新账号密码
- 检查网络连接

问题 2：日程提醒失败

现象：无法查看日程或时间计算错误

原因：日程数据格式错误或时间格式错误

解决方案：

- 检查日期格式（YYYY-MM-DD）
- 检查时间格式（HH:MM）
- 检查时区配置

任务管理系统

问题 1：任务添加失败

现象：无法添加任务或 JSON 文件格式错误

原因：JSON 文件格式错误或写入失败

解决方案：

- 检查 JSON 文件格式
- 检查文件权限
- 使用 JSON 验证工具

问题 2：任务完成失败

现象：任务 ID 不存在或归档失败

原因：任务 ID 错误或归档文件权限错误

解决方案：

- 检查任务 ID 是否正确
- 检查归档文件权限
- 查看错误日志

小红书全自动闭环

问题 1：热点监控失败

现象：无法连接小红书或数据解析失败

原因：MediaCrawler 连接失败或 Cookies 失效

解决方案：

- 检查 MediaCrawler 状态
- 更新 MediaCrawler Cookies
- 增加超时时间

问题 2：内容生成失败

现象：无法生成内容或 API 失效

原因：content-generator 未集成或 API 失效

解决方案：

- 集成 content-generator
- 更新 content-generator API 配置
- 检查 API 密钥

日程集成

问题 1：添加日程失败

现象：日期格式错误或时间格式错误

原因：日期或时间格式不符合要求

解决方案：

- 检查日期格式（YYYY-MM-DD）

- 检查时间格式（HH:MM）
- 使用 24 小时制

自动评论回复系统

问题 1：配置文件错误

现象：JSON 格式错误或配置项缺失

原因：配置文件格式错误或缺少必要的配置项

解决方案：

- 检查 JSON 格式
- 检查配置项完整性
- 查看错误日志

问题 2：Cookies 失效

现象：Cookies 过期或格式错误

原因：Cookies 过期或格式不正确

解决方案：

- 从浏览器重新复制 Cookies
- 更新 config.json 中的 Cookies
- 检查 Cookies 格式

问题 3：监控失败

现象：无法连接小红书或笔记 ID 错误

原因：Playwright 连接失败或笔记 ID 不正确

解决方案：

- 检查 Playwright 状态
- 检查笔记 ID 是否正确
- 增加超时时间

问题 4：回复失败

现象：回复频率过高或内容违规

原因：回复策略过于激进或内容不当

解决方案:

- 调整回复延迟
- 减少每日回复数量
- 优化回复模板

扩展指南

Extension Guide

新增功能模块

扩展 1：数据仪表盘

数据仪表盘功能将提供可视化的数据展示、效果追踪、ROI 计算、趋势分析和话题维度分析。该模块将使用 matplotlib、plotly、streamlit 等技术栈，提供 Web 界面展示各项指标。

- **总览面板**：关键指标、今日数据
- **趋势分析**：粉丝增长、内容表现
- **内容管理**：已发布内容、草稿列表
- **互动分析**：评论回复统计、转化率
- **ROI 报告**：投入产出比、收益分析

扩展 2：A/B 测试系统

A/B 测试系统将实现标题 A/B 测试、封面图 A/B 测试、发布时间测试和结果自动分析。该模块将支持测试标题吸引力、封面吸引力、发布时间、互动率等维度，自动分析最佳方案并推荐。

- **标题 A/B 测试**：测试不同标题的点击率
- **封面图 A/B 测试**：测试不同封面的点击率
- **发布时间测试**：测试不同发布时间的浏览量
- **结果分析**：自动判定最佳方案
- **推荐输出**：推荐最佳标题、最佳封面、最佳发布时间

扩展 3：邮箱集成

邮箱集成将实现 IMAP 连接读取邮件、SMTP 配置发送邮件、未读邮件统计、邮件分类、提醒集成。该模块将支持检查未读重要邮件、标记需要回复的邮件、提取待处理事项、发送提醒邮件。

- **IMAP 连接**：连接邮件服务器，读取未读邮件
- **SMTP 配置**：配置邮件发送功能
- **未读邮件统计**：统计未读邮件数量
- **邮件分类**：按重要、客户、通知分类
- **提醒集成**：发送提醒邮件到指定邮箱

跨平台支持

跨平台 1：抖音集成

抖音集成将实现抖音视频自动发布功能，支持视频上传、描述生成、标签添加、封面设置。该模块将使用抖音开放平台 API，实现一键多平台发布。

- **视频上传**：自动上传视频文件
- **描述生成**：使用 AI 生成视频描述
- **标签添加**：自动添加相关标签
- **封面设置**：设置视频封面
- **发布管理**：管理已发布视频

跨平台 2：B 站集成

B 站集成将实现 B 站视频自动发布功能，支持视频上传、封面图设置、分区选择、标签添加。该模块将使用 Bilibili API，实现一键多平台发布。

- **视频上传**：自动上传视频文件
- **封面图设置**：设置视频封面图
- **分区选择**：选择视频分区
- **标签添加**：自动添加相关标签
- **发布管理**：管理已发布视频

私域运营

私域 1：飞书自动回复

飞书自动回复将实现飞书消息自动回复、客户信息归档功能。该模块将使用飞书开放平台 API，实现自动回复飞书消息、管理客户信息、客户跟进提醒。

- **消息自动回复**：根据关键词自动回复飞书消息
- **客户信息归档**：记录客户互动历史
- **客户分级**：VIP/活跃/新客户分级
- **跟进提醒**：提醒跟进客户

私域 2：微信消息回复

微信消息回复将实现微信消息自动回复功能。该模块将使用企业微信 API，实现自动回复微信消息、客户信息管理。

- **消息自动回复**：根据关键词自动回复微信消息
- **客户信息管理**：记录客户互动历史

- **自动标签**：为客户打标签

高级功能

高级 1：远程协作

远程协作将实现通过 OpenClaw nodes 控制手机、远程拍照采集素材、紧急事务处理、位置共享等功能。该模块将使用 OpenClaw nodes API，实现远程控制手机、远程拍照、远程操作。

- **远程控制手机**：通过 nodes 控制手机
- **远程拍照**：远程控制手机拍照
- **紧急事务处理**：远程处理紧急事务
- **位置共享**：共享手机位置

高级 2：智能决策引擎

智能决策引擎将实现基于多维度数据的智能决策功能。该模块将使用机器学习、数据分析技术，智能分析数据，提供决策建议。该模块支持自动化决策、决策推荐、决策记录。

- **数据输入**：多维度数据输入（热点、竞品、用户行为）
- **智能分析**：使用机器学习模型分析数据
- **决策推荐**：提供智能决策建议
- **决策记录**：记录决策历史

总结

Summary

核心价值

Wilson 老板助理是一个功能完整、技术成熟、易于扩展的全能型 AI 助手，通过 5 大核心功能和模块化架构，实现了从信息获取到任务执行的全流程自动化。系统通过智能决策和自动化执行，大幅提升运营效率，实现一人 = 三人运营团队的目标。

核心价值

- ✅ **数据驱动决策**：基于多维度数据提供运营建议
- ✅ **智能任务分配**：自动识别优先级，智能分配资源
- ✅ **全流程自动化**：从热点监控到内容发布全自动化
- ✅ **实时监控反馈**：持续跟踪效果，自动优化策略
- ✅ **个性化配置**：支持多账号、多平台、多策略配置

技术成熟度

技术成熟度

- ✅ **模块化架构**：每个模块独立可测试
- ✅ **配置驱动**：所有行为通过配置文件控制
- ✅ **日志记录**：完整的操作日志
- ✅ **错误处理**：完善的异常处理和错误恢复
- ✅ **性能优化**：缓存、异步处理、批量操作优化
- ✅ **可扩展性**：支持插件式扩展，可自定义新功能

Wilson 老板助理 - 超高级详细实战手册

版本 v1.0.0 | 2026年2月20日

总页数：8+ | 总字数：15,000+ | 超级详细版