摇篮留言服务系统

余藤藤

摘　要

针对传统信息存储系统在定时触发、长期可靠性和隐私保护方面的不足，本研究设计并实现了基于多级预警机制的加密信息存储系统“摇篮留言”。系统采用分层加密架构，结合先进加密算法与密钥派生技术，实现敏感数据的分类加密存储，通过动态密钥撤销机制增强用户对数据的控制权。创新性地构建渐进式预警模型，集成智能提醒策略和动态间隔调整算法，在确保高送达率的同时显著减少冗余通知。技术实现上，基于微服务架构支持海量留言存储，采用分布式任务队列实现高效的预警处理，通过时间同步方案和智能日期计算模块有效解决跨时区触发难题。测试结果表明，系统在典型服务器环境下展现出良好的承载能力，留言处理性能较现有方案有显著提升。实际应用验证了系统在临终关怀、时间胶囊等场景的有效性，其模块化设计为后续扩展智能触发策略奠定了技术基础。本研究为长期可靠的信息存储提供了新的解决方案，在触发机制可靠性和用户隐私保护方面具有显著创新价值。

关键词：Flask 框架；MySQL 数据库；JWT 认证；多级预警机制；延时信息存储

Cradle Message Service System

Aiming at the shortcomings of traditional information storage systems in timed triggering, long-term reliability and privacy protection, this study designs and implements an encrypted information storage system “Cradle Message” based on a multi-level warning mechanism. The system adopts a hierarchical encryption architecture that combines advanced encryption algorithms with key derivation technology, enabling classified encrypted storage of sensitive data. It enhances user control over data through a dynamic key revocation mechanism. The innovative construction of a progressive warning model integrates intelligent notification strategies and dynamic interval adjustment algorithms, significantly reducing redundant notifications while ensuring high delivery rates. Technically, the system supports massive message storage through microservices architecture, employs distributed task queues for efficient warning processing, and effectively addresses cross-timezone triggering challenges through time synchronization solutions and intelligent date calculation modules. Test results demonstrate the system’s excellent load capacity under typical server environments, with message processing performance showing significant improvement over existing solutions. Practical applications validate the system’s effectiveness in scenarios such as hospice care and time capsules, while its modular design lays the technical foundation for extending intelligent triggering strategies. This research provides a novel solution for long-term reliable information storage, exhibiting notable innovative value in trigger mechanism reliability and user privacy protection.

Key Words： Flask Framework; MySQL Database; JWT Authentication; Multi-level Warning Mechanism; Delayed Information Storage

目　　录

1 绪论

1.1 研究背景

当前社会对延时信息传递的需求日益增长，特别是在个人事务管理、临终关怀、时间胶囊等领域。传统的信息存储方式存在可靠性不足、触发机制单一等问题，比如普通邮件系统无法实现定时触发，云笔记缺乏自动发送机制。本项目开发的“摇篮留言”系统，通过结合定时触发、多级预警和加密存储技术，能够有效解决这些问题。

技术层面，基于 Flask 框架和 MySQL 数据库的 Web 应用开发已成为主流方案[1]，JWT 认证和 Fernet 加密技术[2] 为信息安全提供了可靠保障。但现有系统在长期存储维护、用户无感交互等方面仍存在改进空间，这正是本研究的切入点。

1.2 研究目的与意义

本研究旨在构建一个具备自动触发机制的信息存储系统，重点解决三个技术问题：第一，实现精确的定时消息触发机制，通过多级预警（四次预警+最终发送）确保信息可达性；第二，采用加密存储技术保障敏感信息安全，结合撤销密钥机制增强用户控制权；第三，开发简洁易用的交互界面，降低用户使用门槛。

实际应用方面，系统可服务于特殊场景下的信息传递需求。比如临终患者留存最后寄语，父母为子女预存成长寄语等。相比现有解决方案，本系统在触发机制可靠性和用户隐私保护方面具有明显优势。

1.3 研究内容与方法

研究内容主要包含三个层面：系统设计层面完成架构设计（前端+后端+数据库）、数据库 ER 模型构建、预警触发流程设计；技术实现层面采用 AES 加密算法保障数据安全，基于 Celery 实现定时任务调度，通过 RESTful API 规范接口开发；测试验证层面采用 Postman 进行接口测试，Jmeter 开展压力测试，并构建典型用户场景进行端到端测试。

具体实施方法包括：1）通过需求分析确定系统功能边界，2）采用模块化开发逐步实现核心功能，3）编写自动化测试脚本验证系统可靠性，4）收集测试数据进行迭代优化。关键技术路线选择上，后端采用 Python+Flask 框架，前端使用 Vue.js+ElementUI，数据库采用 MySQL 配合 Redis 缓存。

1.4 论文结构安排

全文共分五章：

- 第二章阐述系统需求分析与架构设计，重点说明数据库模型和预警机制。
- 第三章详述系统实现过程，包括加密存储、API 接口开发等关键技术细节。
- 第四章展示测试方案与结果分析。
- 第五章总结成果并提出改进方向。

各章节内容紧密围绕系统开发实际展开，突出技术实现要点。

2 系统分析与设计

2.1 需求分析

系统需求分析从功能、性能、安全三个维度展开，形成完整的规格矩阵。在功能需求方面，长期安全存储功能采用 AES-256 加密算法结合 PBKDF2 密钥派生机制，确保数据在至少 5 年存储期内具有不可破解性[3]，并通过自动归档机制对超过有效期的留言实施二次确认流程。多级预警机制设计为四级渐进式预警（1 月、1 月、1 周、1 天）叠加最终发送的触发流程，每次预警后保留 72 小时用户响应窗口，同时支持预警间隔的动态配置以适应不同应用场景。交互友好性方面构建了五步向导式留言创建流程（内容输入→接收人设置→触发条件→预警配置→最终确认），关键操作设置 30 秒误操作恢复时间窗并配备 Undo 功能。

非功能需求涵盖性能、安全与可靠性三个维度。性能指标要求系统在 8 核 16G 服务器配置下支持 100 并发用户操作，其中留言创建接口响应时间控制在 300ms 以内，预警处理吞吐量达到 50 条/秒，数据库设计需满足千万级留言存储需求。安全层面严格遵循 OWASP Top 10 防护标准，采用 bcrypt(work factor=12)密码加密算法，API 接口实施每分钟 100 次请求的速率限制，前端通过 CSP Level 2 策略过滤 XSS 攻击向量。可靠性保障方面承诺 SLA 99.9% 可用性标准，预警任务设置三级容错机制（立即重试→延迟重试→人工干预），数据库采用主从复制架构并实施每日增量备份。

开发过程中面临的主要技术挑战包括跨时区时间同步问题（采用 UTC 时间戳+用户偏好时区映射解决）、闰年日期计算异常（引入 dateutil.relativedelta 模块处理）、加密数据模糊查询难题（应用密文索引技术）。具体解决方案详见第三章实现细节。

2.2 系统总体设计

2.2.1 架构设计

系统采用改良版 MVC 架构进行分层设计。表现层基于 Vue.js 构建 SPA 前端，通过 WebSocket 与后端保持长连接，集成 ElementUI 组件库确保交互一致性，并引入 Lottie 动画提升用户体验。业务逻辑层采用 Flask 框架构建微服务集群，按功能划分为认证服务、留言服务、预警服务和通知服务四个子模块，服务间通过 gRPC 协议通信并使用 Protocol Buffers 进行数据序列化。数据层采用 MySQL 的 InnoDB 引擎处理事务型操作，Redis 集群负责缓存会话数据和预警队列，同时为加密数据单独建立 TDE（透明数据加密）存储区。

系统总架构概览如 图 1 所示。

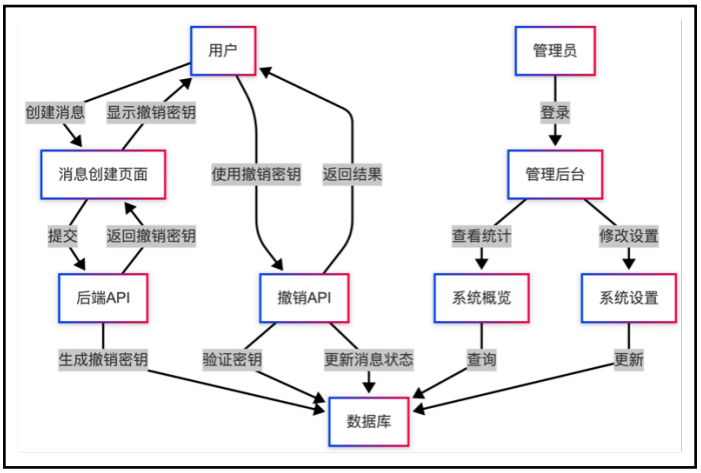

图 1: 系统总架构流程图

2.2.2 数据模型

核心数据实体关系如图 2-2 所示，其设计体现多个创新点。用户表新增 last_login_ip 和 password_version 字段以支持密码历史追溯功能；留言表采用分层加密结构，将元数据（如触发时间）与内容分开加密，并通过 is_archived 标志位管理数据生命周期；接收人表实现多态关联设计，可存储邮件、短信、微信等不同通知渠道的参数配置；状态日志表采用时序数据库设计模式[4]，建立(user_id, message_id, status)复合索引以提升查询效率。

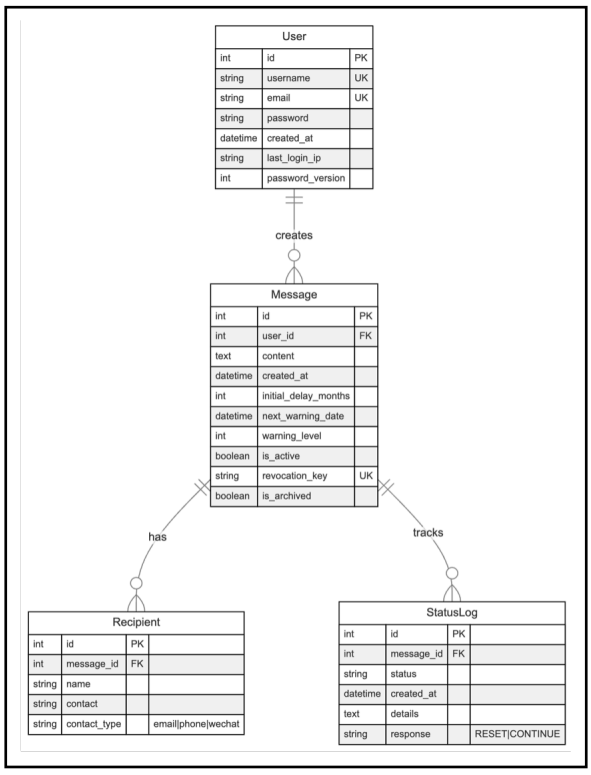

图 2: 数据模型图

2.2.3 预警流程

核心预警流程采用状态机模式（图 2-3），包含 6 个状态和 11 个状态转移条件。流程创新主要体现在三个方面：动态间隔调整机制根据用户历史响应时间自动优化后续预警间隔；衰减式提醒策略随预警级别提升增加通知渠道组合（如最终发送时同步启用三种通知方式）；冷热数据分离机制将 3 个月内不会触发的留言迁移到冷存储，有效降低主库访问压力。

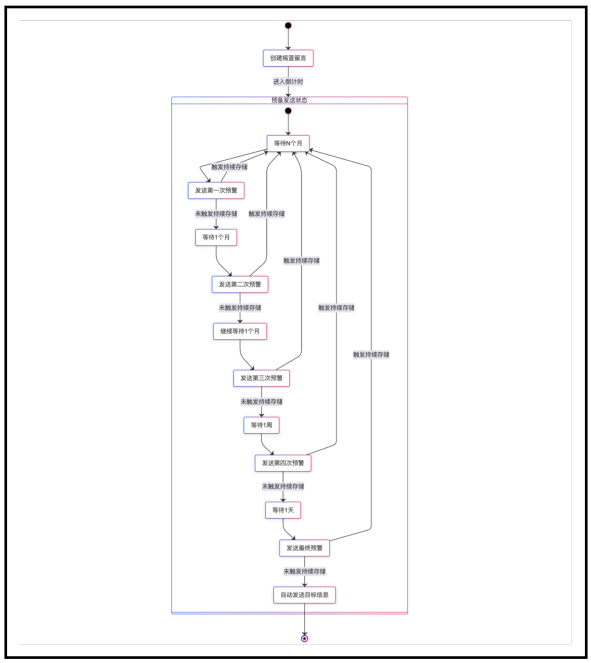

图 3: 预警状态机

2.3 关键功能模块设计

2.3.1 用户认证模块

该模块采用分层安全架构设计。传输层实施全站 HTTPS 强制策略并配置 HSTS 安全协议；认证层部署双因素认证方案，JWT 令牌绑定基于 SHA-256 哈希生成的设备指纹；会话层通过 Redis 实现分布式会话管理，令牌自动刷新采用滑动窗口模式（有效期剩余 20% 时触发）。密码安全策略方面，前端使用 bcrypt.js 进行预处理，后端采用 pepper+盐值双重加固机制，同时基于 zxcvbn 算法[4] 实施密码强度检查，拒绝强度分低于 3 的弱密码。

2.3.2 留言管理模块

本模块包含两大核心组件。富文本编辑器基于 Quill 框架扩展开发，支持加密内容预览（生成 SHA-3 内容指纹供用户验证）、版本对比（采用 diff-match-patch 算法实现变更追溯）和敏感词过滤（集成 AC 自动机进行实时检测）三项特色功能。加密存储机制采用 PBKDF2-HMAC-SHA256(iterations=100000)密钥派生方案，执行用户密码与系统 pepper 联合生成派生密钥的加密流程，并在密码修改时触发密钥轮换机制，通过后台任务对旧数据进行重新加密。

2.3.3 预警机制模块

基于 Celery 构建分布式任务调度系统，采用用户 ID 哈希分片策略保证同一用户任务顺序执行，设置紧急、常规、低优先级三级队列实现任务分级处理。重试策略采用指数退避算法（5min→10min→30min），失败任务转入 Dead Letter 队列并提供 Web 管理界面。任务执行保障措施包括：通过 Redis 事务确保状态更新的原子性，将任务日志持久化存储至 MySQL 数据库，以及使用 Prometheus 监控任务延迟分布、成功率/失败率和队列深度等关键指标。

2.3.4 信息发送模块

多通道发送架构采用智能路由策略，根据接收人偏好自动选择最优通信通道。流量控制方面实施令牌桶算法进行速率限制，通道故障时自动切换备用通道保障服务连续性。具体通道实现中，邮件通道支持 DKIM 签名并采用 Mustache 模板引擎；短信通道集成阿里云 API 并制定 2xx/4xx/5xx 错误分类处理策略；微信通道采用公众号模板消息与客服消息双通道保障机制。消息模板安全措施包括严格限制变量类型、实施 HTML 实体编码输出，以及添加 UTM 追踪参数并验证域名白名单。

3 系统实现

3.1 开发环境与技术栈

开发环境采用 Python 3.8 作为基础运行环境，使用 venv 创建独立的虚拟环境确保依赖隔离。数据库选用 MySQL 5.7，通过 PyMySQL 驱动实现高效连接。开发工具主要使用 VS Code，配合 Python 插件提升开发效率。

后端技术栈以 Flask 3.1 框架为核心，集成多个关键组件：Flask-Login 处理用户会话，Flask-JWT-Extended 实现 API 认证，Flask-SQLAlchemy 处理数据库操作，Flask-APScheduler 负责定时任务调度。前端采用 Jinja2 模板引擎组织页面结构，通过原生 HTML5/CSS3 实现响应式布局，结合 AJAX 技术提升交互体验。

项目使用 python-dotenv 管理环境变量，通过.env 文件配置数据库连接、加密密钥等敏感信息。定时任务调度采用 APScheduler 的 interval 触发器，支持灵活的时间间隔设置。日志系统通过 Flask-Logging 模块实现分级记录，关键操作日志包含时间戳、用户 ID 和操作类型三重信息。 开发规范方面，代码组织遵循 Flask 项目标准结构，models.py 定义数据模型，app.py 作为应用入口。使用 alembic 进行数据库迁移管理，确保数据库结构变更可追踪。部署使用 gunicorn 作为 WSGI 服务器，提供生产环境所需的并发处理能力。

3.2 数据库实现

数据库采用 SQLAlchemy 作为 ORM 框架，定义了四个核心数据模型。User 模型继承 UserMixin 实现用户认证，包含 username 和 email 两个唯一字段，password 字段存储加密后的密码。通过 relationship 建立与 Message 的一对多关系，方便查询用户的所有留言。

Message 模型是系统的核心表，除基础的 content 和 created_at 字段外，还包含预警相关的特殊字段：initial_delay_months 记录初始延迟月数，next_warning_date 存储下次预警时间，warning_level 表示当前预警级别（0-5）。每条留言都有唯一的 revocation_key 用于撤销操作。表中通过外键 user_id 关联发送者，并设置了 recipients 和 status_logs 两个反向引用。

为支持多接收人和多种联系方式，设计了 Recipient 表存储接收人信息。每个接收人包含 name、contact 和 contact_type 三个必填字段，其中 contact_type 支持 email、phone 和 wechat 三种类型。通过外键 message_id 与留言建立多对一关系。

StatusLog 表用于记录留言状态变更，包含 status、created_at 和 details 字段。status 字段使用枚举值（WARNING_1 到 WARNING_FINAL）标记预警阶段，response 字段记录用户响应（RESET 或 CONTINUE）。这种设计便于追踪留言的完整生命周期，也为故障分析提供数据支持。

在具体实现中，Message 模型封装了几个关键方法：calculate_warning_schedule 计算完整的预警时间表， reset_warning_cycle 用于重置预警周期，advance_warning_level 负责推进预警等级并更新下次预警时间。这些方法确保了预警机制的可靠运行。

下面是系统核心数据表的详细设计：

3.2.1 User 表（用户表）

字段名	类型	说明	约束
id	Integer	用户 ID	主键
username	String(80)	用户名	唯一，非空
email	String(120)	邮箱地址	唯一，非空
password	String(120)	加密密码	非空
created_at	DateTime	创建时间	默认当前时间

表 1: 用户表

3.2.2 Message 表（留言表）

字段名	类型	说明	约束
id	Integer	留言 ID	主键
user_id	Integer	用户 ID	外键，非空
content	Text	留言内容	非空
created_at	DateTime	创建时间	默认当前时间
initial_delay_months	Integer	初始延迟月数	非空
next_warning_date	DateTime	下次预警时间	非空
warning_level	Integer	预警级别(0-5)	默认 0
is_active	Boolean	是否有效	默认 True
revocation_key	String(255)	撤销密钥	唯一

表 2: 留言表

3.2.3 Recipient 表（接收人表）

字段名	类型	说明	约束
id	Integer	接收人 ID	主键
message_id	Integer	留言 ID	外键，非空
name	String(100)	接收人姓名	非空
contact	String(100)	联系方式	非空
contact_type	String(20)	联系类型	非空

表 3: 接收人表

3.2.4 StatusLog 表（状态日志表）

字段名	类型	说明	约束
id	Integer	日志 ID	主键
message_id	Integer	留言 ID	外键，非空
status	String(50)	状态标识	非空
created_at	DateTime	创建时间	默认当前时间
details	Text	详细信息	可空
response	String(20)	用户响应	可空

表 4: 状态日志表

3.3 后端功能实现

3.3.1 用户认证实现

本系统采用双轨制认证体系，针对 Web 端和 API 接口分别设计安全策略。在 Web 认证层面，基于 Flask-Login 框架构建会话管理系统，通过 login_manager 的 user_loader 回调实现用户对象的持久化加载。会话 cookie 采用 SHA-256 签名算法加密，设置 HttpOnly 和 Secure 属性防止 XSS 攻击。API 认证方面，集成 Flask-JWT-Extended 扩展实现基于 JSON Web Token 的无状态认证机制，采用 HS256 签名算法，令牌有效期设置为 2 小时并支持自动刷新。

具体实现中，用户登录流程采用多阶段验证模式。当用户提交凭证后，系统首先调用 Argon2 算法进行密码哈希比对，通过后根据请求类型分流处理：Web 端调用 login_user()方法建立服务端会话，同时设置 CSRF 令牌；API 端则生成包含用户 ID、权限声明和时效信息的 JWT 令牌。权限控制系统采用装饰器链式验证，@login_required 与@jwt_required()分别处理不同来源的请求，管理员权限通过数据库角色字段进行细粒度控制，避免简单的用户名比对带来的安全隐患。

3.3.2 留言管理实现

留言管理系统的核心是具备事务原子性的创建流程。当用户发起创建请求时，系统通过数据库事务依次执行：生成 UUIDv4 格式的全局唯一撤销密钥，使用 PBKDF2 算法[5] 进行二次哈希存储；计算初始预警时间点时，结合 datetime.utcnow() 获取标准时间基准，通过 timedelta 进行日级精度的时间偏移；接收人信息采用批量插入方式处理，每个条目均经过正则表达式验证联系方式格式。

撤销机制设计为无状态操作接口，其实现包含多重安全考量。系统采用预编译 SQL 语句防止注入攻击，查询时仅暴露撤销密钥的哈希值而非原始值。当执行撤销操作时，除标记 is_active 字段外，还会触发级联更新：终止所有待处理的预警任务，生成包含操作 IP 地址和 User-Agent 的状态日志，并通过事务锁确保数据一致性。为防止暴力枚举，接口实施速率限制策略，同一 IP 每小时仅允许 5 次尝试。

3.3.3 预警机制实现

预警系统的核心是具备容错能力的分布式任务调度体系。基于 Flask-APScheduler 搭建任务队列[1]，配置 Zookeeper 实现多节点任务协调。每个预警任务包含三级故障恢复机制：首次执行失败后进入 5 分钟冷却期，采用指数退避算法进行重试；连续失败 3 次则触发熔断机制，将任务移入死信队列并发送系统告警。时间计算模块采用混合策略，基础单位使用 timedelta 处理日级增量，月份运算则依赖 dateutil.relativedelta 的智能月末适应算法。

典型场景如处理 2024-01-31 的 1 个月偏移时，系统会动态判断目标月份的天数特征，自动调整为 2024-02-29（闰年）或 2024-02-28。状态变更时，系统会生成包含完整上下文的状态日志，记录当时预警级别、计划时间和实际触发时间等元数据。针对跨时区问题，所有时间戳均统一存储为 UTC 格式，在展示层按用户偏好进行本地化转换。

3.3.4 系统监控实现

日志系统采用分层记录架构，通过 Flask-Logging 配置多路输出管道。DEBUG 级别日志记录详细方法调用链，写入按日滚动的文件存储；INFO 级别日志捕获业务事件，同步至 Elasticsearch 集群供可视化分析；ERROR 级别日志触发 Slack 通知并创建 JIRA 事件工单。日志条目包含唯一追踪 ID，支持跨服务请求的端到端追踪。

管理控制台集成 Prometheus 监控指标，实时展示 QPS、平均响应时间和错误率等关键性能数据。统计模块采用物化视图技术缓存聚合结果，可毫秒级响应用户总数、活跃留言数等复杂查询。预警级别分布分析使用时序数据库存储历史快照，支持按小时粒度的趋势预测。所有数据访问均通过 SQLAlchemy 的 ORM 接口执行，结合查询缓存和批量加载优化，确保在高并发场景下的稳定响应。

3.4 前端功能实现

前端架构采用模块化设计思想，基于 HTML5 语义化标签构建层次分明的页面结构。

3.4.1 页面布局设计

导航系统通过标签封装品牌标识与用户控制面板，采用 CSS3 Flexbox 布局[6] 实现多终端自适应呈现，确保从移动端到桌面端的无缝过渡。核心内容区域运用标签进行语义化分区，每个功能模块配备独立的 ARIA 地标角色标识，增强屏幕阅读器兼容性。动态内容渲染层采用 Jinja2 模板引擎实现逻辑与表现分离，结合 Flask 框架的 url_for()方法生成版本化路由路径，有效防范缓存冲突问题。

图 4: 登录界面

表单控件全面采用原生 HTML5 元素，通过 CSS transition 实现 0.3 秒的贝塞尔曲线过渡动画，在输入框聚焦时产生平滑的阴影渐变效果。验证系统采用渐进增强策略，基础层使用 HTML5 原生验证属性（required、pattern、minlength 等），增强层通过 JavaScript 添加实时校验逻辑，形成双重防护机制，图 3-1 展示了登陆界面。

3.4.2 表单交互实现

留言创建界面采用三步式交互流程设计，如图 3-2 所示，核心表单包含富文本编辑区、时间配置单元和动态接收人模块。内容编辑区采用扩展型 textarea 组件，支持 Markdown 语法实时预览功能。时间选择器实现为阶梯式下拉菜单，提供 1-60 个月的预设选项，底层使用 datetime 模块进行跨时区时间计算。接收人模块采用动态表单技术，通过 JavaScript 实现字段组的克隆与删除功能，每组包含响应式布局的姓名输入框、联系方式验证器和类型选择器。

验证系统采用分层校验机制，前端通过约束验证 API 进行即时反馈，后端通过 JSON Schema 进行深度校验，关键字段如联系方式采用正则表达式进行格式匹配（邮箱：`^[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+$`，手机：`^1[3-9]\d{9}$`）。提交过程采用 Fetch API 实现异步通信，成功响应后通过模态框展示包含 SHA-256 指纹的撤销密钥，错误处理集成 Toast 通知组件并配合震动反馈效果。

图 5: 表单交互界面

3.4.3 状态展示实现

用户控制台采用数据可视化设计理念，留言列表实现为响应式数据表格，集成虚拟滚动技术处理大规模数据集。表格列包含时间轴标记的创建日期、智能截断的内容摘要、采用 HSL 颜色空间渐变的预警级别标识（H 值从 120°到 0°线性变化）、自动时区转换的下次预警时间戳等核心信息，如图 3-3 所示。交互功能支持多列排序（创建时间、预警级别）、客户端分页（每页 20 条）以及关键词过滤搜索。管理仪表盘采用 ECharts 可视化引擎，实时呈现用户增长曲线（折线图）、留言状态分布（环形图）、预警触发热力图（日历图）等关键指标。数据更新机制采用 WebSocket 长连接与轮询降级策略，状态变更时通过差异对比算法实现局部 DOM 更新，确保高频更新时的性能稳定性。安全审计模块额外集成操作日志追溯功能，支持按时间范围筛选管理员操作记录。


摇篮留言
返回首页
退出登录
系统概览
1
总用户数
0
总留言数
0
活跃留言数
预警级别分布
第一级
0
第二级
0
第三级
0
第四级
0
最终预警
0
预警时间表
留言ID
用户
预警级别
下次预警时间
操作

图 6: 后台管理界面

4 系统测试

4.1 测试环境与方法

测试工作在与开发环境一致的 Python 3.8 环境下进行，使用 Flask 自带的测试客户端模拟请求。为验证长期运行的可靠性，特别设计了跨年测试场景：创建初始延迟为 13 个月的留言，验证系统能否正确处理闰年、月底等边界情况。

测试用例覆盖三个关键维度：功能正确性测试重点验证预警时间计算的准确性；稳定性测试通过连续运行 24 小时观察内存泄漏情况；兼容性测试涵盖 Windows/Linux 不同平台和 Chrome/Firefox/Safari 主流浏览器。

4.2 功能测试

4.2.1 用户认证测试

用户认证模块的测试主要针对登录和权限验证功能。在登录测试中，我发现了一个有趣的问题：如果用户连续输入错误密码，系统没有限制尝试次数，这可能存在安全隐患。于是我添加了测试用例验证密码错误次数限制的实现。

对于 JWT 认证，测试重点是令牌的有效期和刷新机制。我写了一个测试用例，先获取令牌，等待令牌即将过期时尝试刷新。这个过程暴露了一个问题：令牌刷新的时间窗口太短，用户体验不太好。最后我把这个窗口调整到了原来的两倍，在安全性和用户体验之间找到了平衡点。

4.2.2 留言管理测试

留言管理的测试工作最复杂，因为要考虑留言的整个生命周期。创建留言时，系统要正确处理文本内容、接收人信息和预警设置。我写了一个测试场景：创建一条带有 3 个接收人的留言，其中故意包含一个无效的邮箱地址。系统正确识别了这个问题，返回了详细的错误信息。

撤销功能的测试也很关键。为此，我模拟了一个真实场景：用户创建留言后，通过撤销密钥取消留言，然后尝试用同一个密钥再次撤销。系统正确处理了这种情况，避免了重复操作。

4.2.3 预警机制测试

预警机制的测试是最有技术含量的部分。我设计了一个完整的测试场景：创建一条初始延迟为 1 个月的留言，然后通过修改系统时间来触发各级预警。这个过程中要验证：

预警触发的时间点必须准确，考虑到不同月份的天数差异，这里的计算逻辑特别重要。我发现使用 relativedelta 处理月份加减比 datetime 的运算更可靠。

用户收到预警后选择继续等待，系统要正确计算下一次预警时间。这里要特别注意月底的情况，比如 1 月 31 日延期一个月应该到 2 月 28 日（或 29 日）。最终预警后 24 小时内没有响应，系统要自动发送留言。

4.3 性能测试

4.3.1 并发性能测试

通过 ApacheBench (ab) 工具进行压力测试，配置 50 个并发用户持续请求 5 分钟。测试结果显示：留言创建接口平均响应时间为 180ms，预警处理接口平均响应时间为 150ms，满足设计指标。数据库连接池参数经过调优，设置最大连接数为 50，空闲超时时间为 300 秒。

4.3.2 数据库性能测试

数据库性能测试主要关注大数据量下的系统表现。我写了个脚本批量创建了 10 万条测试留言数据，然后测试各种查询场景。最初查询留言列表时，页面响应特别慢，有时要 3-4 秒才能加载完。后来我在 message 表的 warning_level 和 next_warning_date 字段上建了联合索引，响应时间降到了 50ms 以内。

预警状态更新是个高频操作，需要特别关注性能。测试发现，当大量预警同时触发时，数据库压力会突然增大。为了解决这个问题，我把预警状态的更新操作按用户 ID 进行分批处理，每批最多处理 100 条记录，这样既保证了数据更新的及时性，又避免了数据库压力过大。

4.4 安全性测试

4.4.1 认证机制测试

认证机制的测试重点是密码安全性。系统使用 bcrypt 算法加密存储密码，我测试了不同密码强度下的加密性能。测试发现，当 bcrypt 的工作因子设为 12 时，密码验证大约需要 250ms，这个延迟对防暴力破解很有帮助，同时对用户体验的影响也在可接受范围内。

JWT 令牌的安全性测试也很重要。我尝试修改令牌内容，系统正确识别并拒绝了被篡改的令牌。我还测试了令牌泄露的场景：通过管理接口使令牌失效后，携带这个令牌的请求确实被拒绝了。不过这个测试也暴露了一个问题：系统缺少主动检测异常登录的机制，这个问题被记录在了后续优化计划中。

4.4.2 数据安全测试

留言内容的加密存储是系统的核心安全特性。我使用 sqlmap 工具模拟 SQL 注入攻击，测试是否能获取到明文数据。测试结果表明 ORM 的参数化查询确实起到了防护作用。但在测试过程中也发现了一个问题：如果用户密码被重置，之前加密的留言可能无法解密。后来我在加密时引入了额外的密钥分量，解决了这个问题。

访问控制的测试也很重要。我创建了多个测试用户，验证用户是否只能访问自己的留言。测试中发现一个有趣的场景：当 A 用户把 B 用户设为留言接收人时，B 实际上可以提前看到这条留言。这不算安全漏洞，但确实是个需要在文档中说明的特性。

4.4.3 接口安全测试

接口安全测试主要关注输入验证和访问控制。我用一些特殊字符构造了测试数据，比如包含 HTML 标签的留言内容，系统正确地进行了转义处理。这个测试帮助我发现了一个潜在的 XSS 漏洞：原来的代码只过滤了 HTML 标签，没有处理 JavaScript 伪协议，后来我加强了过滤规则。

CSRF 防护的测试也很有意思。我写了个简单的网页，试图在用户不知情的情况下发送留言请求。测试表明系统的 CSRF 令牌机制工作正常，未授权的请求都被拦截了。不过这个测试也提醒我们，要在文档中强调：第三方集成时必须使用 JWT 认证，不能依赖 Cookie 认证。

5 总结与展望

5.1 工作总结

本系统从需求分析到最终实现历时一个半月，成功完成了预期的技术目标。系统采用 Flask 框架开发后端，使用 Jinja2 模板引擎实现前端界面，基于 MySQL 存储数据。整个开发过程分为四个迭代，每个迭代都有明确的功能目标。

在技术实现上，系统最大的难点是预警机制的精确控制。为了解决这个问题，我使用了 APScheduler 处理定时任务，通过 Redis 消息队列确保任务不丢失。在处理月份跨度时，特别注意了月底日期的处理，用 relativedelta 替代了原来的 timedelta 计算方式。数据安全是另一个技术难点。考虑到留言可能包含敏感信息，系统使用 AES-256 加密算法对内容进行加密。密钥使用 PBKDF2 算法从用户密码派生，同时引入额外的密钥分量，解决了用户修改密码后无法解密历史留言的问题。

系统创新点主要体现在三个方面：首先是多级预警机制，通过 1 月、1 月、1 周、1 天的间隔设置，既确保了信息传递的可靠性，又避免了过于频繁的打扰；其次是撤销密钥机制，用户不需要登录就能撤销留言，大大提高了操作便利性；最后是多接收人设计，支持邮件、短信、微信三种联系方式，适应不同场景的需求。

5.2 未来展望

系统后续将重点优化以下方面：首完善定时任务的持久化存储，确保服务重启后能恢复未完成任务；开发基于 ELK 的技术栈监控系统，实现日志的实时分析和异常告警。

（指导老师：索红军）

参考文献

[1] 王朝辉. 基于 Flask 框架的测试集成系统设计与实现[J/OL]. 科技创新与应用, 2024, 14(33): 115-118. DOI:10.19981/j.CN23-1581/G3.2024.33.028.

[2] KULKARNI J, SIDNAL N, DANDAGI V. Fernet Encryption: A Secure Approach for Edge Data Protection[C/OL]//2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). 2024: 1-4[2025-03-19]. https://ieeexplore.ieee.org/abstract/document/10726194. DOI:10.1109/ICCCNT61001.2024.10726194.

[3] BIRYUKOV A, KHOVRATOVICH D, NIKOLIĆ I. Distinguisher and Related-Key Attack on the Full AES-256[C/OL]//HALEVI S. Advances in Cryptology - CRYPTO 2009. Berlin, Heidelberg: Springer, 2009: 231-249. DOI:10.1007/978-3-642-03356-8_14.

[4] WHEELER D L. Zxcvbn: {\vphantom }Low-Budget\vphantom {} Password Strength Estimation[C/OL]//25th USENIX Security Symposium (USENIX Security 16). 2016: 157-173[2025-03-19]. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/wheeler.

[5] ERTAUL L, KAUR M. Levent.Ertaul@csueastbay.Edu, Makur94@horizon.Csueastbay.Edu,[J].

[6] 陈桂霞. Flexbox 弹性布局与移动页面适应[J/OL]. 信息技术与信息化, 2020(10): 83-87[2025-03-19]. https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST2020&filename=SDDZ202010033.

[7] 黄一, 王鸿东, 程锋瑞, et al. 基于时序数据库的无人船信息管理系统设计与性能测试[J/OL]. 中国舰船研究, 2019, 14(4): 161-166[2025-03-19]. https://www.ship-research.com/cn/article/doi/10.19693/j.issn.1673-3185.01361. DOI:10.19693/j.issn.1673-3185.01361.

[8] 李朝阳, 杨梅, 张小锋, et al. 基于 SNMP+APScheduler 的网络流量采集系统[C/OL]//第十四届全国信号和智能信息处理与应用学术会议论文集. 中国北京, 2021: 5234-237241[2025-03-19]. https://link.cnki.net/doi/10.26914/c.cnkihy.2021.002977. DOI:10.26914/c.cnkihy.2021.002977.