**软件开发文档2.0.0**

**网络通信软件开发文档（增强版）**

**文档版本**

* **版本号**：2.0.0
* **更新日期**：2025年5月15日
* **更新内容**：
* 新增模块接口详细说明
* 补充通信协议消息示例
* 扩展安全机制配置细节
* 添加系统性能优化建议
* 增加未来扩展方向

**目录**

1. **系统概述**
2. **系统架构**
3. **模块详细说明**
4. **数据库设计**
5. **通信协议**
6. **安全机制**
7. **安装与配置指南**
8. **使用说明**
9. **测试方法**
10. **性能优化建议**
11. **常见问题**
12. **未来扩展方向**

**1. 系统概述**

**核心功能**

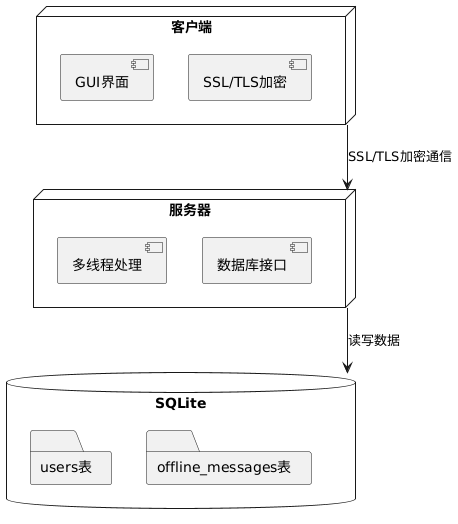
* **用户管理**：支持注册、登录、密码哈希存储（Bcrypt），管理员权限控制。
* **实时通信**：
* 文本消息：点对点聊天，支持离线消息存储与转发。
* 文件传输：支持任意文件类型，自动保存至本地files目录。
* **管理员功能**：
* 查看/删除用户
* 发送全局公告
* **安全机制**：
* SSL/TLS 1.2加密通信
* 自签名证书验证（支持SAN扩展）
* 密码哈希存储（Bcrypt，12轮盐值）

**技术栈**

* **客户端**：Tkinter GUI、SSL/TLS、多线程消息监听
* **服务器**：Python Socket、多线程处理、SQLite3数据库
* **依赖库**：pyOpenSSL、bcrypt、sqlite3

**2. 系统架构**

**架构图**



**数据流说明**

1. **用户认证**：

* 客户端通过SSL握手连接服务器，发送login或register请求。
* 服务器验证密码哈希，返回认证结果及离线消息。

1. **消息传输**：

* 客户端发送消息时，若目标用户在线则直接转发，否则存储至offline\_messages表。

1. **管理员操作**：

* 管理员通过admin\_command消息执行用户管理或公告发送。

**3. 模块详细说明**

**3.1 SSL证书模块（SSL.py）**

* **功能**：生成自签名证书，支持多域名和邮箱绑定。
* **输入参数**：

|  |
| --- |
| Python domain\_list = ["tset.cn"] # 证书支持的域名列表（首个为Common Name） email = "2421484330@qq.com" # 证书关联邮箱 |

* **输出文件**：
* tsetcn.crt：X.509证书文件
* tsetcn.key：RSA私钥文件（2048位）
* tsetcn.pem：合并的PEM文件（私钥+证书）
* **依赖库**：OpenSSL
* **异常处理**：
* 域名格式错误时抛出ValueError。
* 文件写入失败时记录日志并终止程序。

**3.2 数据库模块（database.py）**

**表结构**

* **users表**：

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 约束 |
| id | INTEGER | PRIMARY KEY |
| username | TEXT | UNIQUE, NOT NULL |
| password\_hash | TEXT | NOT NULL |
| is\_admin | BOOLEAN | DEFAULT FALSE |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

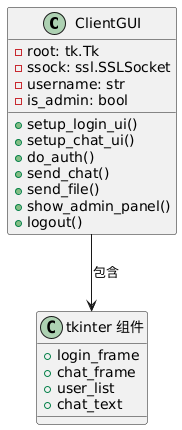
* **offline\_messages表**：

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 约束 |
| id | INTEGER | PRIMARY KEY |
| sender | TEXT | NOT NULL |
| receiver | TEXT | NOT NULL |
| message\_type | TEXT | NOT NULL ('chat'或'file') |
| content | BLOB | NOT NULL |
| filename | TEXT | 可选（仅文件类型） |
| timestamp | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**关键方法**

* **add\_user(username: str, password\_hash: bytes) -> bool**
* **功能**：添加用户，返回是否成功。
* **异常**：sqlite3.IntegrityError（用户名重复）。
* **save\_offline\_message(sender: str, receiver: str, message\_type: str, content: bytes, filename: str = None) -> None**
* **功能**：保存离线消息，支持文本和文件类型。
* **get\_offline\_messages(receiver: str) -> List[Tuple]**
* **返回值**：格式为(sender, message\_type, content, filename)的列表。

**3.3 客户端模块（gui.py）**



**类与方法**

* **ClientGUI**\*\*类\*\*：
* **方法**：
* \_\_init\_\_()：初始化GUI，创建登录和聊天界面。
* do\_auth()：处理登录/注册请求，建立SSL连接。
* send\_chat()：发送文本消息，需选择目标用户。
* send\_file()：通过文件对话框选择并发送文件。
* show\_admin\_panel()：管理员专属功能入口。

**界面布局**

* **登录界面**：
* 单选按钮切换登录/注册模式。
* 输入框：用户名、密码（掩码显示）。
* 确定按钮触发do\_auth()。
* **主聊天界面**：
* **左侧**：Treeview组件展示在线用户及状态（管理员/普通用户）。
* **右侧**：
* 消息显示区域（Text组件，只读模式）。
* 消息输入框（Entry组件）。
* 功能按钮：发送消息、发送文件、刷新用户列表、退出登录。
* **状态栏**：显示连接状态或错误提示。

**3.4 协议模块（protocol.py）**

**消息格式**

* **头部（Header）**：

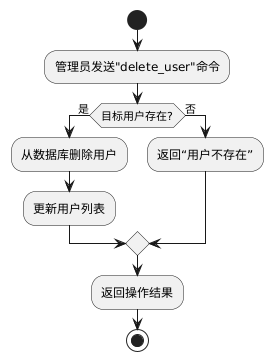
|  |
| --- |
| JSON {  "type": "chat", // 消息类型（必填）  "length": 1024, // 消息体长度（字节）  "to": "target\_user", // 目标用户（chat/file类型必填）  "filename": "data.txt", // 文件名（file类型必填）  "action": "list\_users" // 管理员命令类型（admin\_command必填） } |

* **消息体（Body）**：
* 文本消息：UTF-8编码字符串。
* 文件消息：二进制数据流。

**关键方法**

* **send\_message(sock, msg\_type, content, extra\_headers=None)**
* **参数说明**：
* sock：已连接的socket对象。
* msg\_type：消息类型（如chat、file）。
* content：消息内容（str或bytes）。
* extra\_headers：附加头部字段（字典格式）。
* **recv\_message(sock) -> (header, content)**
* **返回值**：
* header：解析后的字典（可能包含type、length等字段）。
* content：原始字节流。
* **异常**：
* 消息头解析失败时抛出json.JSONDecodeError。
* 数据接收不完整时返回(None, None)。

**3.5 服务器模块（server.py）**



**核心逻辑**

* **多线程处理**：
* 每个客户端连接由独立线程处理（handle\_client方法）。
* 使用client\_map\_lock确保client\_map的线程安全。
* **消息分发逻辑**：

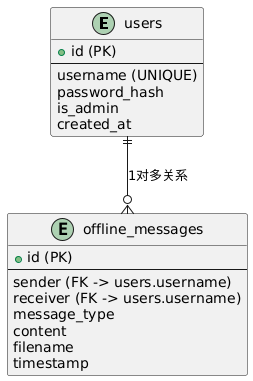
|  |
| --- |
| Python if 目标用户在线:  直接转发消息 else:  保存至offline\_messages表  通知发送方“用户离线” |

**管理员命令**

* **list\_users**：
* 返回所有用户列表（管理员可见全部，普通用户仅可见非管理员）。
* 数据格式：[[username, is\_admin], ...]（JSON序列化）。
* **delete\_user**：
* 删除指定用户，并返回更新后的用户列表。
* **announcement**：
* 广播消息至所有在线用户，头部标记为[系统公告]。

**4. 数据库设计**

**ER图**

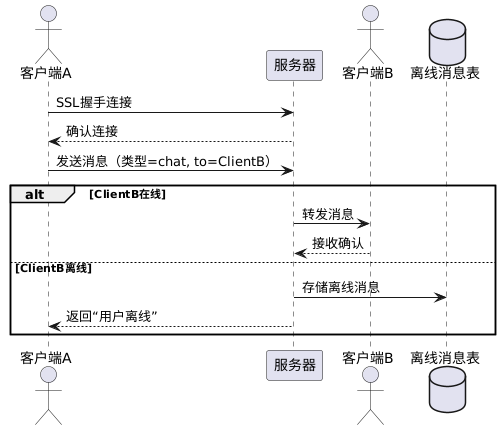


* **外键关系**：offline\_messages.sender和receiver关联users.username。

**索引优化**

* **users.username**：唯一索引加速用户查询。
* **offline\_messages.receiver**：B树索引加速离线消息检索。

**5. 通信协议**



**消息示例**

1. **用户登录请求**：

|  |
| --- |
| JSON // Header {  "type": "login",  "length": 5,  "password": "hashed\_password" } // Body: "admin" |

1. **文本消息发送**：

|  |
| --- |
| JSON // Header {  "type": "chat",  "length": 11,  "to": "user2" } // Body: "Hello World" |

1. **文件传输**：

|  |
| --- |
| JSON // Header {  "type": "file",  "length": 2048,  "to": "user2",  "filename": "report.pdf" } // Body: <二进制数据> |

**通信流程图**

|  |
| --- |
| Plain Text sequenceDiagram  participant ClientA  participant Server  participant ClientB   ClientA->>Server: SSL握手建立连接  Server-->>ClientA: 确认连接  ClientA->>Server: 发送消息（类型=chat, to=ClientB）  alt ClientB在线  Server->>ClientB: 转发消息  ClientB-->>Server: 接收确认  else ClientB离线  Server->>Database: 存储离线消息  Server-->>ClientA: 返回“用户离线”  end |

**6. 安全机制**

**SSL/TLS配置**

* **协议版本**：TLS 1.2
* **证书验证**：
* 客户端：验证服务器证书的CN和SAN（tset.cn）。
* 服务器：加载PEM文件（证书+私钥）。
* **加密套件**：默认使用ECDHE-RSA-AES256-GCM-SHA384。

**密码存储**

* **哈希算法**：Bcrypt（12轮盐值，自动生成salt）。
* **验证逻辑**：

|  |
| --- |
| Python stored\_hash = db.get\_user(username)[0] if bcrypt.checkpw(password.encode(), stored\_hash):  # 登录成功 |

**7. 安装与配置指南**

**环境要求**

* **Python版本**：3.7+
* **依赖安装**：

|  |
| --- |
| Bash pip install pyopenssl bcrypt |

**证书配置**

1. 运行SSL.py生成证书：

|  |
| --- |
| Bash python SSL.py # 生成tsetcn.crt、tsetcn.key、tsetcn.pem |

1. 将证书文件放置于SSL/目录下。

**服务器启动**

|  |
| --- |
| Bash python server.py # 默认监听127.0.0.1:8090 |

**客户端配置**

* **证书路径**：修改gui.py中load\_verify\_locations("SSL/tsetcn.crt")。
* **服务器地址**：修改client\_socket.connect(('127.0.0.1', 8090))。

**8. 使用说明**

**用户操作流程**

1. **登录/注册**：![登录界面](假设截图链接)

* 选择“登录”或“注册”模式，输入用户名和密码。

1. **发送消息**：

* 在用户列表中选择目标用户。
* 输入消息内容，点击“发送”。

1. **发送文件**：

* 点击“发送文件”按钮，选择本地文件。

1. **管理员功能**：

* 登录管理员账户后，点击“管理面板”按钮。
* 支持查看用户列表、删除用户、发送公告。

**9. 测试方法**

**单元测试**

* **数据库测试**：

|  |
| --- |
| Python def test\_add\_user():  db = Database()  assert db.add\_user("test", bcrypt.hashpw("123".encode(), bcrypt.gensalt())) is True |

* **协议测试**：

|  |
| --- |
| Python def test\_send\_message():  sock = create\_dummy\_socket()  send\_message(sock, "chat", "Hello")  assert sock.sent\_data == expected\_header\_and\_body |

**集成测试**

* **场景**：模拟两个客户端同时登录并通信。
* **步骤**：

1. 客户端A发送消息给离线客户端B。
2. 客户端B登录后应自动接收离线消息。

**10. 性能优化建议**

* **数据库优化**：
* 启用SQLite WAL模式提升并发性能。
* 使用连接池管理数据库连接。
* **消息队列**：引入Redis或RabbitMQ处理高并发消息。
* **异步IO**：使用asyncio重构服务器以支持异步处理。

**11. 常见问题**

**Q1：客户端连接失败**

* **可能原因**：
* 服务器未启动或端口被占用。
* SSL证书路径错误。
* **解决方案**：
* 检查服务器日志，确认端口监听状态。
* 验证客户端证书路径配置。

**Q2：文件传输失败**

* **可能原因**：
* 目标用户离线且服务器files目录权限不足。
* 文件大小超过系统限制。
* **解决方案**：
* 确保files目录存在且可写。
* 分片传输大文件（未来扩展功能）。

**12. 未来扩展方向**

1. **功能扩展**：

* 群组聊天（多播消息）。
* 消息撤回与编辑。
* 端到端加密（AES-GCM）。

1. **性能优化**：

* 支持WebSocket协议降低延迟。
* 引入缓存机制加速用户列表加载。

1. **多平台支持**：

* 开发Web端（Flask/Django）。
* 移动端适配（Kivy框架）。

**备注**：本文档基于代码版本1.0.0编写，后续更新需同步修改版本号及对应功能说明。