留言板说明文档

目录

简介

python 实现的一个简单留言板 web 应用。包含用户注册、登录、登出、留言、回复、修改密码和邮件找回密码的功能。

前端采用 BOOTSTRAP3 框架设计。使用 javascript 实现动态内容的生成，使用 xml 实现前后端交互。

后端采用 mysql 存储数据。没有采用第三方框架，而是用自己基于 Socket 设计的 TCP-HTTP-CGI 三层结构实现服务。IO 采用 selectors 库实现多路复用。

用户验证采用 cookie 验证 url 参数的方式进行。

部署说明

系统为 ubuntu 16。程序已在我的服务器 106.53.106.196:8000 上部署，可以直接访问。

1. 数据库

首先需要安装 mysql server。

接下来有两种方案，任择其一即可：

a. 指定服务器连接数据库使用的用户名，密码和 database。

打开 /framework/mysql_info.py 。按照注释修改 database_info 中的对应属性值。主要是用户名，密码和 database。指定的用户应当对指定 database 拥有全部权限。

b. 创建服务器需要的用户名，密码和 database。

命令为：

```
create database MessageBoard;
grant all privileges on MessageBoard.* to 'xujing'@'localhost' identified by 'xujing_pass' with grant option;
flush privileges;
```

2. Python

使用版本为 Python 3.6.9。需要额外引入 pymysql 库用于实现 Python 和 mysql 的交互。

配置命令：

```
pip3 install pymysql
```

3. 程序入口

在主目录执行 python3 main.py 即可。

使用说明

默认端口为 8000。可以在浏览器直接访问我的服务器 106.53.106.196:8000 开始使用。已在 IE8，Chrome（电脑端和移动端），FireFox 三个浏览器测试，运行正常。

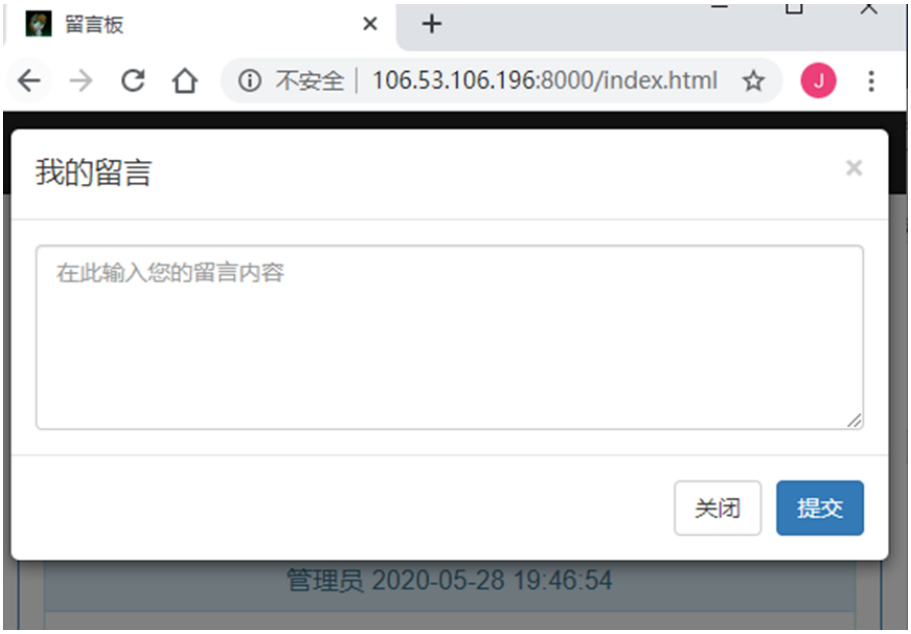
一共三个页面。大部分功能通过弹出模态框来实现。用户操作的结果通过改写 HTML 界面展示，并一般在 3 秒后返回主页面。以下介绍页面交互：

1. 主界面

访问地址 106.53.106.196:8000 后首先进入的就是这一页面。页面交互按钮功能为：

1 进入登录/注册页面。（登录后是进入用户档案页）

2 进入留言窗口。（下为点击后的结果示例）

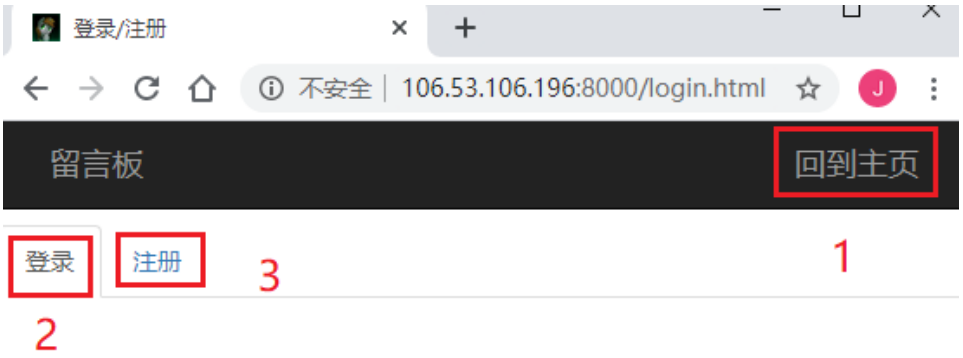


3 加载指定留言的回复。（下为点击后的结果示例）

4 对指定留言进行回复。（窗口样式同 2）

5 加载更多留言（初始仅加载最近 10 条，每次加载 10 条）。

2. 登录/注册



页面交互按钮功能为：
1 进入主页面
2 切换至登录界面（图中界面）
3 切换至注册界面：

4 弹出重置密码窗口。填写并提交信息后，服务器将尝试用 163 邮箱向用户注册邮箱发送重置后的密码。
5 根据所填信息尝试登录。（在注册界面是注册）

3. 用户资料

页面交互按钮功能为：
1 回到主页面
2 退出登录状态并回到主页面
3 进入修改密码界面（如图所示）

4. 注意事项

4.1 在关闭浏览器时自动退出登录状态，而非关闭标签页时。用户也可以

选择在用户资料页点击登出来主动退出。

4.2 设计上仅能回复留言，无法回复对留言的评论。

4.3 不能多设备同时登录同一账号，不允许同一浏览器同时登录多个账号。

设计说明

1. 需求分析

首先，确定留言板所需的基本功能，并对功能进行基本分析，制成下表：

功能	需要验证登录状态	可能结果（不考虑网络问题和 Bug）
获取留言/评论	否	无留言/评论
		有留言/评论
新增留言/评论	是	未登录，留言失败
		留言成功
登录	否	用户不存在
		密码错误
		登录成功
注册	否	用户名或邮箱已存在
		注册信息不合法
		注册成功
找回密码	否	非用户本人
		用户本人，但无法向注册邮箱发邮件
		重置成功
获取用户信息	是	获取成功
		验证失败
修改密码	是	改密成功
		验证失败
登出	是	登出成功
		未登录

依据这一表格进行后续设计。

2. 数据库设计

不难看出，数据主要分为两类：

1 用户信息。

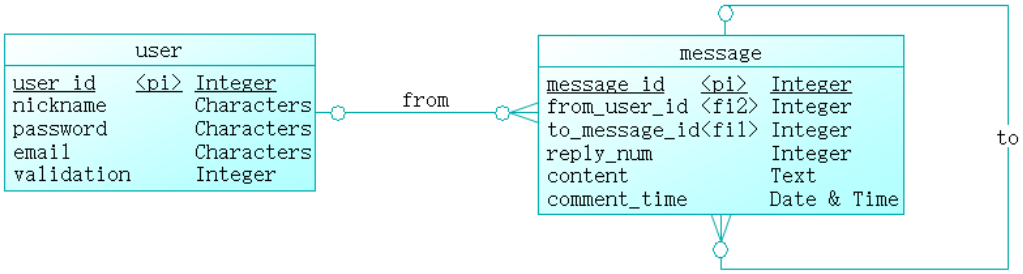
2 留言/评论信息。

因此比较自然的设计是分成两个表存储。一个表 user 负责用户信息，另一个表 message 负责留言/评论信息。

根据功能分析，user 必须存储用户名（nickname），密码（password）和找回密码所需邮箱（email）；message 必须存储的信息为作者（from_user_id）、目标留言（to_message_id）和留言内容（content）。

由于后续技术实现的需要，对 user 额外添加主键和验证码（验证登录

validation），对 message 添加主键，留言时间（用于排序，comment_time）和回复数（提高查询性能,reply_num）。最终数据库设计为：



其中除 user 的 validation 属性外均不可为 NULL。nickname 和 email 为 unique 属性。详细数据库建表语句如下：

```
CREATE TABLE IF NOT EXISTS `user`(
    `user_id` INT UNSIGNED AUTO_INCREMENT,
    `nickname` VARCHAR(30) UNIQUE KEY NOT NULL,
    `password` VARCHAR(30) NOT NULL,
    `email` VARCHAR(80) UNIQUE KEY NOT NULL,
    `validation` INT,
    PRIMARY KEY ( `user_id` )
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `message`(
    `message_id` INT UNSIGNED AUTO_INCREMENT,
    `from_user_id` INT UNSIGNED NOT NULL,
    `to_message_id` INT UNSIGNED NOT NULL,
    `reply_num` INT NOT NULL,
    `content` TEXT NOT NULL,
    `comment_time` DATETIME NOT NULL,
    PRIMARY KEY ( `message_id` )
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3. 前端设计

综合数据库设计和是否需要验证登录状态，可以把功能分为三类：

3.1 需要访问 message 表。只有两项功能：获取留言/回复和新增留言/回复。

3.2 不需要访问 message 表，也不需要验证登录状态。包含三项功能：登录、注册和找回密码。

3.3 不需要访问 message 表，但需要验证登录状态。包含三项功能：获取用户信息，修改密码和登出。

依据以上分类，将功能分划为三个页面（index 主页面、login 登录注册页面、profile 用户资料页面）分别处理。通过 javascript 动态生成页

面内容，通过 XMLHttpRequest 和 HTML 中的 form 表单两种方式与后端交换数据。

4. 后端框架设计

主要分为三层：

4.1 TCP 层。这一层主要负责 TCP 连接的建立，连接的处理交给下一层。

4.2 HTTP 层。这一层主要负责解析 HTTP 报头，生成 HTTP 响应报头，根据报头处理网络 IO。但是如何处理解析后的报头和报文内容，以及响应报文内容则交给下一层。

4.3 CGI 层。这一层实现具体业务逻辑。以字典形式接收 HTTP 层解析完毕的 HTTP 报文和报头，然后根据具体业务逻辑对数据库进行操作，最后以字典形式返回响应报文内容和报头。由 HTTP 层组装并发送。

5. 用户登录状态

由于 HTTP 是无状态协议，这里用 cookie 实现状态的保持和验证。设计上在每次登录时生成一个随机验证码存储在用户数据库，并将验证码和用户 id 一并存储在前端。用户的每次需要验证登录状态的操作都会先比对前后端存储的验证码和用户 id，确认是否为用户本人。除注册外，前端不可能获取用户密码，也就最大限度的防止了盗号现象。

6. 文件结构

main.py 程序入口。

/static 存储前端代码和资源。包括三个页面的 HTML 和 JS 代码以及一个网站头图。

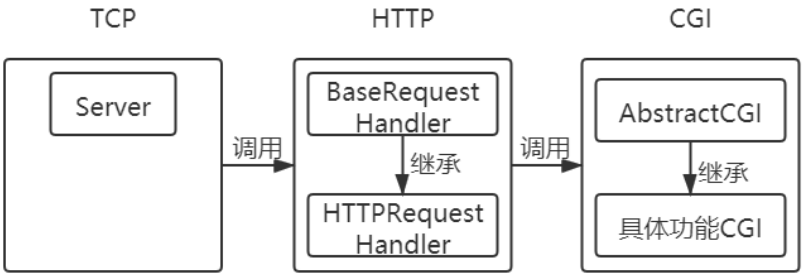
/framework 存储后端代码，其中：

email_info.py:	邮箱配置文件
mysql_info.py:	数据库配置文件
httphandler.py:	HTTP 层实现
server.py:	TCP 层实现
/cgi	CGI 层实现及代码，每个文件对应一个功能。

框架代码说明

1. 总体框架

具体分工见设计说明的“后端框架设计”一节。各 class 继承和调用关系如下图。接下来三节详述各层具体实现。



2. TCP 层

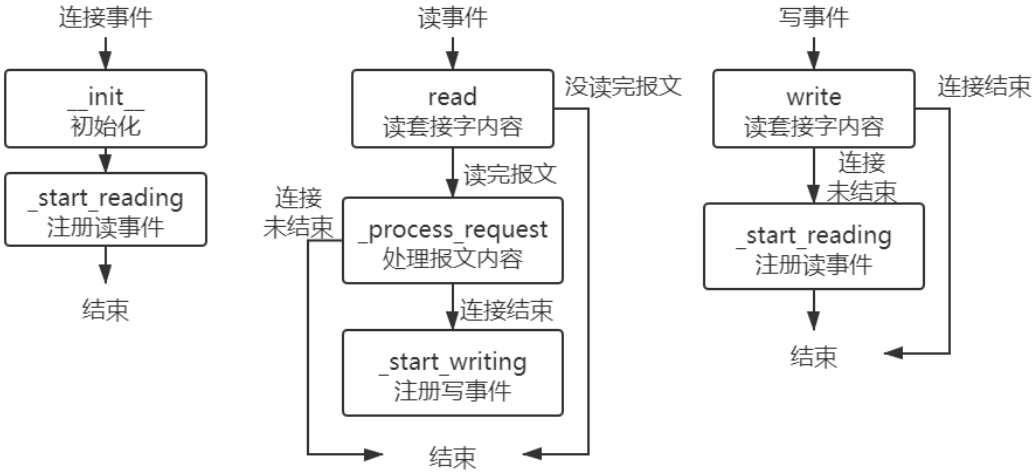
这一层在 server.py 中实现。核心是 Server 类。用户先调用 __init__() 方法初始化（包括数据库初始化），然后调用 serve_forever 方法。该方法核心代码如下：

```
75      def serve_forever(self, max_wait=0.5, backlog=100):
76          """ serving forever until shut down signal.
77          Arguments are:
78          * max_wait: when receive shutdown signal, close socket in max_wait seconds
79          * backlog:  the number of unaccepted connections that the system will
80                      allow before refusing new connections.
81          """
82          self._socket.listen(backlog)
83          self._selector.register(self._socket, selectors.EVENT_READ, self._handle_request)
84          while True:
85              events = self._selector.select(max_wait)
86              if self._shutdown_server_request:
87                  self._shutdown_server_request = False
88                  break
89              for key, mask in events:
90                  callback = key.data
91                  callback()
92
93          self._server_close()
```

核心逻辑是在 selector 上注册套接字的连接事件，然后在每次事件发生时调用_handle_request（这一函数在每次调用时新建一个 HTTP 层实例处理连接后事务）。每轮循环均会检测关闭 flag，检测到后退出循环。

3. HTTP 层

这一层在 httphandler.py 中实现。主要流程可以按触发事件做如下划分：



通过一个位掩码变量来控制读写，保证单一连接不可能同时读并且写。只在读写完成时取消注册读/写事件。

_process_request 函数通过调用_handle_get 方法处理 GET 请求，调用_handle_post 方法每次创建新的 CGI 层实例处理 POST 请求。返回结果再调用_assemble_response 方法根据 CGI 层返回结果生成 HTTP 报文。

4. CGI 层

这一层在/cgi 文件夹中实现，原则上是每个功能一个文件。HTTP 层通过解析请求 URL 信息来确定调用哪个文件的内容。所有文件均继承同一个 AbstractCGI 类。这一抽象类主要提供通用的工具函数（如 XML 转义_xml_special_symbol_normalize，登录状态验证_vaild_user，数据库连接和关闭）以及供 HTTP 层调用的入口函数 handle 函数的定义。

自我总结

1. 亮点

1.1 自行设计 Web 后端框架处理连接建立、IO、数据库、HTTP 报文解析和生成等事务。网络相关只使用了 socket，urllib，smtplib 和 email。对后端业务逻辑有了较全面的认识。

1.2 使用 selectors 多路复用实现 IO。

1.3 HTML，XML，SQL 三层防注入，对特殊字符支持性好。

2. 改进方向

2.1 用户感受/功能设计

2.1.1 HTTP 升级为 HTTPS（目前为了方便部署采用 HTTP）。没有 HTTPS 很容易被伪基站盗号。

2.1.2 注册时需要验证邮箱（目前为了方便测试时注册没有验证）

2.1.3 在关闭标签页时退出登录状态，而非关闭浏览器时。

2.1.4 对管理员账号添加更多特权，包括回复重点显示（使用红色外框），用户管理（禁言，封号等）和消息管理（撤回等）。

2.1.5 添加用户头像，允许用户上传头像文件，并在前端显示。

2.2 后端设计

2.2.1 TCP 层和 HTTP 层的隔离不够。由于浏览器兼容性问题，原本设计的每次请求后断开连接没能实现（火狐浏览器不能接受 HTTP1.1 以下版本）。在 keep-alive 模式下 HTTP 层需要控制 IO 来实现对不同请求之间的分隔。使得原本设计中 TCP 管 IO，连接；HTTP 层管解析报文的分工失败。HTTP 层需要操作 TCP 层设计中的私有成员 selector。考虑将两层合并，或者将 IO 转移给 TCP 层，在 HTTP 层分割请求，将 TCP 层作为 buffer 使用。

2.2.2 HTTP 层的数据库初始化函数破坏了 HTTP 和 CGI 的隔离。考虑增加配置文件，使其能够从 CGI 库中读取指定文件初始化数据库。

2.2.3 前后端进一步分离。操作结果界面的 HTML 目前大多数写在 CGI 层。考虑要么转移到 static 文件夹里，由 CGI 层读取并返回；要么由前端页面 JS 代码生成，后端只负责返回操作状态而非结果的 HTML 页面。

2.2.4 完善 HTTP 支持。增加对 HTTP 的 HEAD 请求的支持。在 Response 中增加更多报头项。对 Request 报头项做出更多个性化反应。将 GET 请求的实现从 HTTP 层转移至 CGI 层。

2.2.5 在后端数据库存储登录状态。这将进一步加强安全性，并使多设备同时登录成为可能。同时，后端实现登出操作也可以防止前端通过遍历验证码的方式绕开登录过程而被后端认可为已登录状态。

2.2.6 部分配置文件改为以 xml/JSON 存储。目前以 py 存储的坏处是，要想更改公用邮箱地址就必须中断服务，修改文件后重启程序。换为 xml 等非代码格式并在每次请求时重新加载，可以在付出一定运行效率的基础上使可能的配置变化更为平滑，无需中断服务。

2.2.7 优化目前 debug 信息。细化各种异常的处理，添加打印 traceback。

2.3 前端代码

2.3.1 JS 代码和 HTML 代码分离不够。考虑各页面 JS 模块独立出来作为单独文件。

2.3.2 优化代码规范。目前存在一些单行代码过长，结构混乱的问题。

2.3.3 主页面获取 comment/reply 的两个函数合并，加强代码复用性。

2.3.4 在实现 2.2.3 的基础上美化返回页面，使之与其他页面风格统一。