信息安全科技创新
项目结题报告

项目名称：应用代理防火墙

项目组成员：

姓名	班级	学号
徐宇飞	F2103603	521021910393
陈冠宇	F2103603	521021910401
林中阳	F2103603	521021910395
柴继晨	F2103603	521021910340
闫梓涵	F2103603	521021910457

院系：电子信息与电气工程学院

学期：2022-2023 学年 第 3 学期

报告完成日期：2023.7.19

项目摘要

随着互联网的普及和应用程序的广泛使用，网络安全威胁不断增加。Web 应用程序通常直接面向公共网络，是攻击者的主要目标之一。传统的网络防火墙主要关注网络层和传输层安全，对于应用层的攻击和恶意行为缺乏有效的防护能力。应用代理防火墙旨在通过**便捷交互**、**访问控制**、**数据过滤**、**隐私保护**和**日志记录**的功能，提高 Web 安全性，强化对 Web 应用程序的防护。

本项目分为**用户管理**、**信息处理**和**图形界面**三个主要实现模块，各个模块又分为不同的功能模块。

用户管理模块实现了用户的增加、删除、查找和修改等基本功能。通过有效的用户管理，可以降低恶意用户对系统的攻击风险，并提高系统的安全性。

信息处理模块则是项目的核心模块，其包括规则控制、消息解析、文件控制、内容过滤和日志、缓存等子功能。**规则控制功能**基于预定义规则，对输入和输出的数据进行过滤和控制，从而限制恶意行为和漏洞利用的风险。**消息解析功能**针对 HTTP 消息的常见部分进行解析，以便对其中的恶意信息进行检测和拦截。**文件控制功能**则针对文件的常见属性，如名称、大小、类型等进行安全验证和检查，防止恶意文件的传输和使用。**内容过滤功能**可以根据设定的规则，对传输的内容进行过滤和筛选，防止敏感信息的泄露和不当使用。**日志缓存功能**用于记录系统的操作日志，方便追溯和审计系统的使用情况。

图形界面模块为用户提供了友好和直观的交互界面，使用户能够方便地配置和管理防火墙的各项功能。通过图形界面，用户可以轻松地修改规则、及时了解系统的安全情况。

以上三个模块相互协作，共同强化了 Web 应用程序的安全性。本项目同时设计了**客户端**和**服务端**，**一个服务端可以同时为多个用户提供服务**。其中，客户端的用户可以使用图形界面便捷地进行控制使用，服务端作为防火墙主体，主要以命令行的方式完成信息处理和用户管理的功能。

本项目完成了初期规划的基本功能，但未完成 FTP 消息和 HTTPS 消息的有效控制，同时文件控制逻辑较为简单、控制范围较小，消息解析对细节的解析不足等部分功能优化不足。针对基本功能，本项目的测试情况良好，在充分实现基本功能的基础上，体现了作为应用代理防火墙**在应用层较好的控制能力和安全性**，以及**便捷友好的服务特性**。

目　录

第一章 需求分析 4
1. 项目需求分析 4
2. 项目所要完成的功能目标 4
第二章 总体设计 6
1. 总体结构图 6
2. 模块 6
第三章 详细设计 9
1. 请求接收模块 9
2. 缓存模块 9
3. 规则控制模块 10
4. 用户管理模块 12
6. 日志模块与传输模块 15
7. 客户端图形界面 16
8. 文件控制模块 21
9. HTTP 消息解析模块 22
第四章 系统实现 25
1. 实现环境 25
2. 开发工具 25
3. 源文件情况 25
4. 程序组成与调用流程 33
第五章 系统测试 34
1. 测试流程 34
2. 编译测试 34
3. 运行测试 36
4. 测试总结 53
第六章 项目小结 54

第一章 需求分析

1. 项目需求分析

应用代理防火墙是一种网络安全解决方案，其主要目标是保护网络和系统免受各种网络威胁和攻击。本项目旨在实现 **Http 应用代理防火墙**，并提供**更加强大的安全控制功能**。具体而言，我们需要注意以下几个方面：

（1）增强控制规则要素的管理：控制规则控制满足何种条件的网站可以通过代理服务器实现与客户端的通信，因此，控制规则的管理是很重要的，可以存在如下问题，例如规则的配置不够灵活、不能针对不同的用户进行定制等。因此，我们需要增加一些控制规则要素的管理功能，以便更好地满足实际需求。

（2）增加文件传输安全控制：传输文件是客户端与网站通信过程中的常用操作，但在传输过程中可能存在一些安全隐患，例如文件名、文件大小、文件类型等信息可能会被恶意攻击者利用。因此，我们需要增加文件传输安全控制功能，例如根据用户的文件传输管理规则限制传输文件的类型、大小等，以提高系统的安全性。

（3）满足用户使用需求：例如增强用户个人信息的安全管理、增加动态控制和控制规则、文件传输规则配置界面等，以提供更加友好和便捷的操作方式。

（4）实现功能优化扩展：例如，通过缓存功能提高通信效率，对具体文件修改时防止资源冲突，提供错误信息处理等功能，方便程序的进一步优化与使用。

2. 项目所要完成的功能目标

根据项目需求分析，我们针对 http 通信预计实现如下功能，在完成后对 ftp 通信进行同样的尝试：

（1） 客户端：

- 客户端拥有**图形界面**，用户可以通过图形界面方便地注册、登录和销号。
- 用户的口令在注册时通过 **sha256 加密处理**，以 16 进制的形式统一储存于服务器 account.txt 文件中。
- 在创建和删除账号时，能够**创建和删除相应的用户文件**。

（2） 控制规则的添加和管理：

- 用户可以通过客户端**自主添加控制规则**，包括 IP 地址、端口、是否允许通信。
- 用户可以通过图形界面方便地**查看**和**管理**自己的控制规则。
- 每个用户有**专属的文件夹**与文件储存控制规则与文件传输规则，**account.txt** 在修改与读取时**上锁**，防止防火墙和管理员同时访问导致资源冲突。

（3） 代理服务器的建立和链接：

- 代理服务器成功地与客户端建立链接，并获取远程套接字。
- 代理服务器成功地与目标网站建立链接，并获取远程套接字。
- 采用多线程方式，对于每个请求创建新的子线程，实现**多用户同时使用防火墙**。
- 代理服务器能够分析请求报文，读取用户控制规则，通过比对请求报文和控制规则判断是否符合规则，若符合控制规则，则将请求报文发送给目标网站。

- 代理服务器能够接受目标网站的返回报文，并获取 **keywords.txt** 获取**过滤关键词**。判断是否在报文中找到关键词，以决定是否将返回报文传回客户端并允许客户端与网站通信。

（4） 缓存管理和淘汰算法：

- 代理服务器能够将请求报文和返回报文储存在缓存中，并利用 **LFU 算法**实现缓存淘汰。
- 每次客户端尝试通信时，先判断缓存中是否已有相应的通信报文，若有，则直接传输缓存中的报文，在多次反复通信时免于每次都进行连接操作。

（5） 文件传输规则添加

- 用户通过**图形界面**实现文件传输控制规则的添加、删除、修改。
- 用户通过**图形界面**可以方便的查看、管理文件传输控制规则。

（6） 文件传输控制

- 代理服务器能够获取文件的基本信息，并根据文件传输规则判断是否允许客户端与网站之间传输文件。

（7） 服务器账户与过滤关键词管理：

- 在服务器端，我们实现了通过命令行增加、删除、修改账户与过滤关键词的功能。
- 在创建和删除账号时，能够创建和删除相应的用户文件。
- 在对公共资源进行修改和读取时，上**进程锁**，防止资源冲突

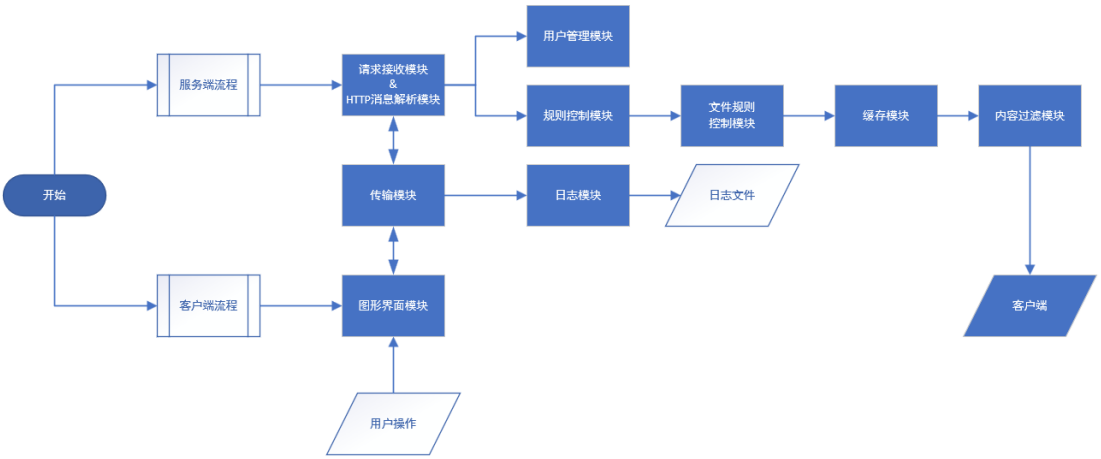
（8） 错误信息处理：

- 程序运行一旦发生错误，便会**在服务器端输出相应错误信息**，方便程序的优化与维护。

总体而言，我们成功地实现了应用 http 代理防火墙的主要功能，包括通过控制规则管理客户端与网站通信、客户端图形界面、用户账户管理与控制规则管理，但未对 ftp 通信进行尝试。同时，我们还根据实际需求进行了一定的功能扩展，如缓存管理、内容过滤、文件传输管理、错误信息处理等。这些功能的实现有助于提高系统的安全性和可靠性，同时也为用户提供了更加便捷和友好的操作体验。

第二章 总体设计

1. 总体结构图



2. 模块

（1） 缓存模块：

①功能：

将经常访问的数据缓存起来，减少对后端服务器的请求，提高数据的访问速度和性能，并且优化数据传输过程，减少延迟和带宽消耗。

②依赖的信息：

客户端请求报文、服务器返回报文、使用频率。

③输出：

保存在内存中的返回报文，每一条请求信息对应一组返回报文；请求内容是否被缓存的标志。

（2） 内容过滤模块：

①功能：

根据报文中的信息判断是否符合传输要求，判断是否存在敏感信息。

②依赖的信息：

服务器发送的报文数据内容、缓存在内存中的报文数据内容、关键词库。

③输出：

通过匹配敏感信息，返回是否允许继续传输的标志。

（3） 用户管理模块：

①功能：

一方面，对于用户来说，该模块实现用户的注册、登录、退出、注销操作，对密码采用 sha256 加密。

另一方面，对于防火墙管理员来说，该模块管理用户，包括用户的增删查改功能，将所有用户的信息写入指定文件中。

②依赖的信息：

客户端的套接字、客户端发来的报文信息、管理员的控制指令。

③输出：

内存中相应用户的信息、对保存用户信息的文件进行相同的操作，为新建立的用户创建用户文件夹。

（4） 规则控制模块

①功能：

采用白名单的形式，控制传输过程中的各种要素，包括 URL、客户端 IP、服务器 IP 等，同时与传输模块结合，根据客户端发送报文中的关键词进行相应操作，例如增删查改规则，并将报文数据内容与设定的规则要素进行匹配。

②依赖的信息：

客户端修改控制文件请求报文、客户端访问网页请求报文、用户的控制规则文件。

③输出：

与日志管理模块结合，记录对规则进行的操作，同时包括是否成功进行操作的响应报文。如果符合控制规则，则将服务器返回的报文传递给客户端，否则拒绝传输并关闭套接字。

（5） 传输模块：

①功能：

在客户端和代理之间传输信息，包括客户端处用户的操作例如创建、注销用户，增删查改控制规则等。

②依赖的信息：

用户的具体操作（与图形界面的结合）。

③输出：

不同的操作对应的报文（包含识别不同操作的关键词）。

（6） 日志管理模块：

①功能：

存储用户修改规则的操作。

②依赖的信息：

当前用户的用户名，用户相应操作对应报文中的关键词。

③输出：

将用户的操作记录等写入文件中。

（7） 图形界面模块：

①功能：

为用户的使用提供便捷、易于查看的图形界面。

②依赖的信息：

用户的操作、各个模块输出的信息。

③输出：

用户可视化的界面，以图形化的方式展示防火墙的各项功能和信息。

（8） 文件规则控制模块：

① 功能：

采用黑名单的方式，根据四种文件规则（禁止出现的文件名、禁止出现的文件类型、最大文件大小、文件传输方向）来决定文件的传输是否通过。

②依赖的信息：

客户端和服务器发送的报文数据、从文件规则中读取到的特定的规则内容。

③输出：

是否允许传输，或能否通过规则匹配。

（9） HTTP 消息解析模块：

①**功能：**

解析 HTTP 报文的信息，将 HTTP 报文的信息保存在一个类中，并提供便于其他模块获取信息的接口。

②依赖的信息：

客户端发送的报文数据、服务器发送的报文数据。

③输出：

报文数据中的各部分常见内容，如请求方法、请求 URL、请求头部各字段、响应头部等。

（10） 请求接收模块：

①**功能：**

从客户端接受请求，为每个请求创建子线程进行处理。

②依赖的信息：

客户端发送的请求报文。

③输出：

处理该请求的子线程。

第三章 详细设计

1. 请求接收模块

（1） 功能：

创建监听套接字，监听 8888 端口，从客户端接受请求，为每个请求创建子线程进行处理。

（2） 依赖的信息：

客户端发送的请求报文。

（3） 数据处理流程：

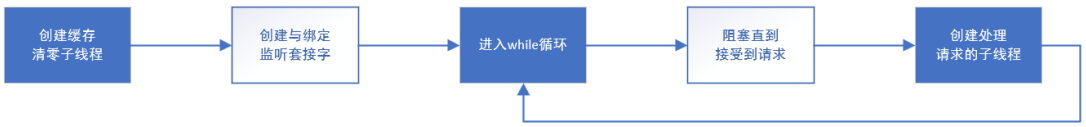
（4） 主要数据结构的设计：

① **struct sockaddr_in serverAddress：**监听的地址与端口。其中，地址设置为所有 ipv4 地址，端口设置为 8888。

② **int listenSocket = socket(AF_INET, SOCK_STREAM, 0)：**监听套接字，创建之后将其绑定到 **serverAddress** 上，从端口接收消息。

（5） 函数设计：

int main()：给出服务端启动的提示信息，创建与绑定监听套接字，启动缓存模块中清零的子线程，同时为每个收到的请求创建处理的子线程。



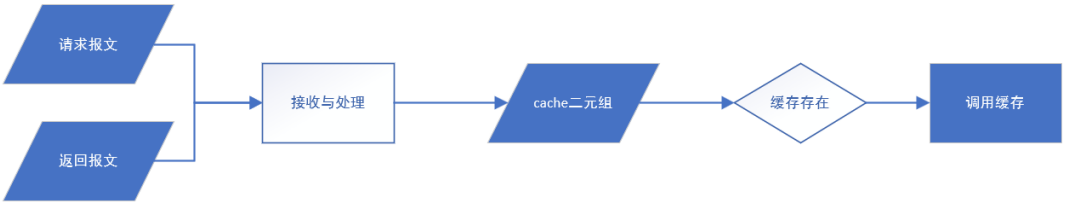
2. 缓存模块

（1） 功能：将经常访问的数据缓存起来，减少对后端服务器的请求，提高数据的访问速度和性能，并且优化数据传输过程，减少延迟和带宽消耗。

（2） 输入数据：客户端请求报文、服务器返回报文、使用频率。

（3） 输出数据：服务器内容是否被缓存的标志，若被缓存，则同时返回缓存的内容。

（4） 数据处理流程：



（5） **主要数据结构的设计：**

① **buff 结构体：**

```
struct buff
{
    vector<string> buffer;  // 缓存
    bool flag = false;  // 启用标志
    int frequence = 0;  // 频数
};
```

vector 容器 **buffer** 存储同一个请求的所有返回报文。

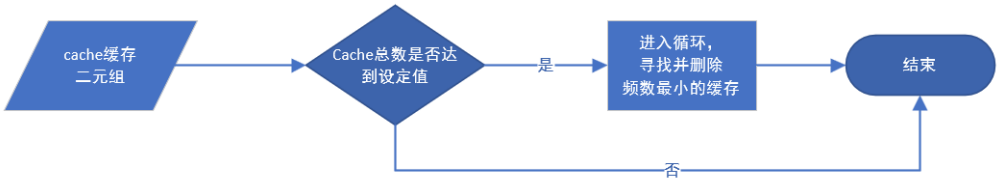
flag 标志服务器返回报文是否完整，若 flag 为 false，则表示返回报文不完整，此时其他子线程不能使用该缓存；若 flag 为 true，则表示该报文返回完整，可以被其他子线程使用。

frequency 表示对该服务器请求的次数，而不是缓存被使用的次数，这样更贴合实际。

② **unordered_map<string, buff> cache：**这个符号将缓存与请求报文联系起来，每个报文对应一组请求，以实现对缓存的调用。

（6） **函数设计：**

① **void LFU()：**最不经常使用(最少次)淘汰算法。遍历 **cache** 二元组，淘汰一段时间内使用次数最少的缓存，避免长时间使用后，内存被大量占用。



② **void *zero(void *arg)：**清零函数。在服务端运行防火墙后创建的子线程，用来定时对 buff 数据结构中的频数清零，以辅助实现淘汰算法。

（7）设计难点与解决办法：

难点：对于有很多分支的逻辑结构，加锁之后很难判断在什么时候进行解锁，或者说有没有解锁成功，这就导致经常会有子线程卡主导致整个防火墙瘫痪。

解决办法：添加判断语句“**if (pthread_mutex_trylock(&mutex)) pthread_mutex_unlock (&mutex);**”。解释：如果能运行到这一步，说明本线程已经对资源加锁，此时，尝试加锁，若加锁不成功则表示本线程未成功解锁，需对资源进行解锁操作。

3. 规则控制模块

（1） **功能：**

采用白名单的形式，控制传输过程中的各种要素，包括 URL、客户端 IP、服务器 IP 等，同时与传输模块结合，根据客户端发送报文中的关键词进行相应操作，例如增删查改规则，并将报文数据内容与设定的规则要素进行匹配。

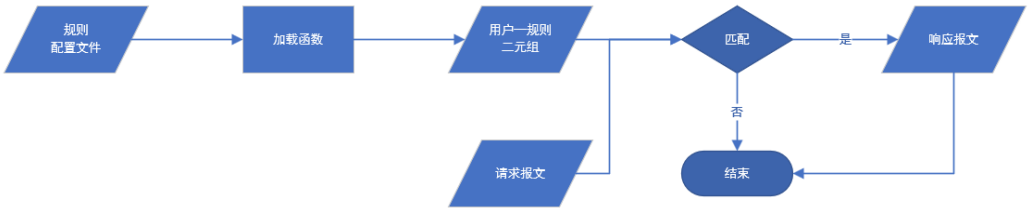
（2） **依赖的信息：**

客户端修改控制文件请求报文、客户端访问网页请求报文、用户的控制规则文件。

（3） **输出：**

与日志管理模块结合，记录对规则进行的操作，同时包括是否成功进行操作的响应报文。如果符合控制规则，则将服务器返回的报文传递给客户端，否则拒绝传输并关闭套接字。

（4） **数据处理流程：**



（5） **主要数据结构的设计：**

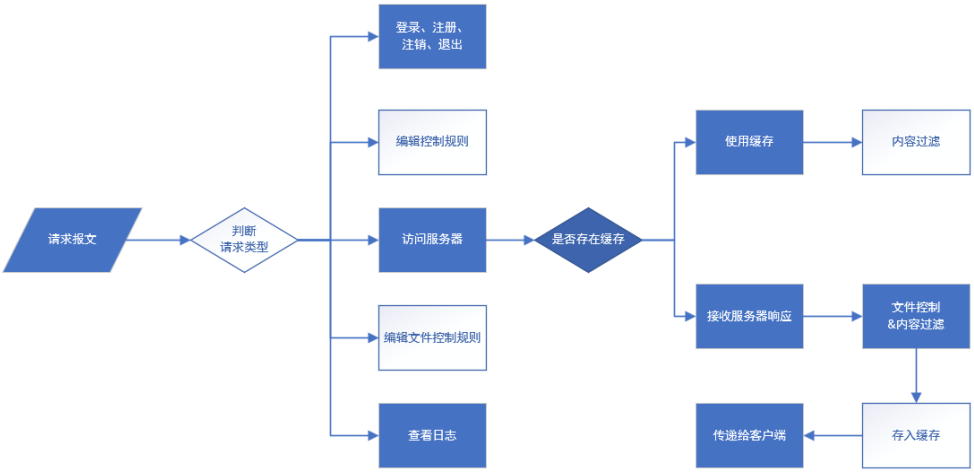
① **unordered_map<string, string> cip：**绑定用户名与客户端的 ip 地址，实现针对每个客户使用客户自己的配置规则，同时避免了每次访问网页都需要输入密码。

② **unordered_map<string, bool> status：**用户是否启用防火墙的标志。

③ **unordered_map<string, vector<tuple<string, int, string, string>>> rule：**将每个用户的规则配置文件的数据加载到内存，以便于实现规则的匹配。

（6） **函数设计：**

void *handleProxyRequest(void *arg)：处理客户端请求的子线程。首先，判断客户端的操作，包括登录、注册、注销、修该规则、访问服务器等。针对访问服务器的请求，判断是否满足控制规则，若是，则允许访问，否则拒绝请求。



（7） **设计难点与解决办法：**

难点：大部分服务器在给出所有响应之后没有明确的结束标志，这时如果采用完全阻塞的方式来从服务器获取报文，则会大大增加等待时长，使防火墙的效率明显降低。

解决办法：设置服务器套接字的最大阻塞时长为 1 秒，当阻塞时长超过一秒的情况下结束套接字，在网络良好的情况下，该方法具有比较好的合理性同时很大程度上提升了防火墙的通信效率。

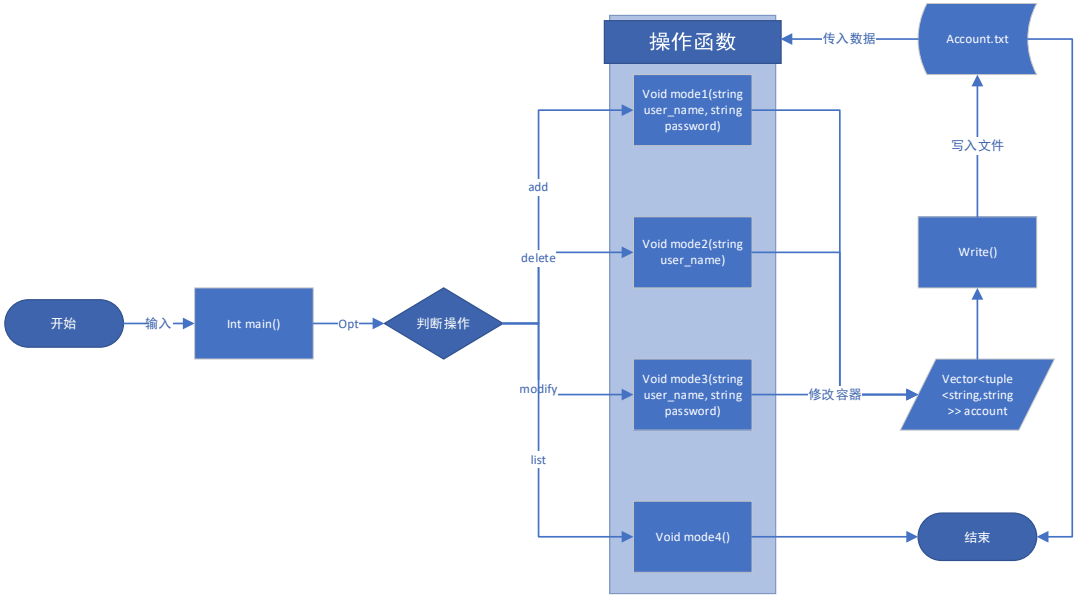
4. 用户管理模块

（1） **功能：**实现服务器使用命令行对用户账户的直接操作，包括创建账户，删除账户，修改账户密码，展示所有账户等操作。

（2） **输入数据：**命令行输入

（3） **输出数据：**输入正确命令时输出修改后的用户信息，输入错误命令时输出错误信息。

（4） **数据处理流程：**



（5） **主要数据结构的设计：**

① vector 容器 **account**，储存账户以及对应的密码。

② int 变量 **opt**，用于读取用户命令。

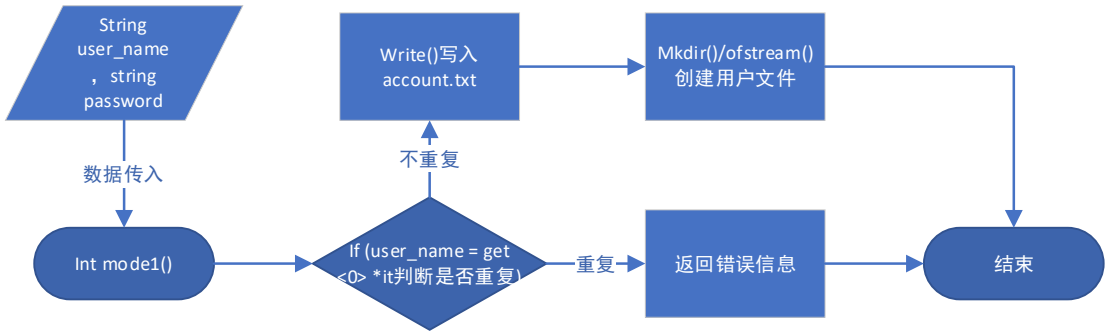
③ string 变量 **user_name** 储存输入的用户名。

④ string 变量 **password** 储存输入的密码。

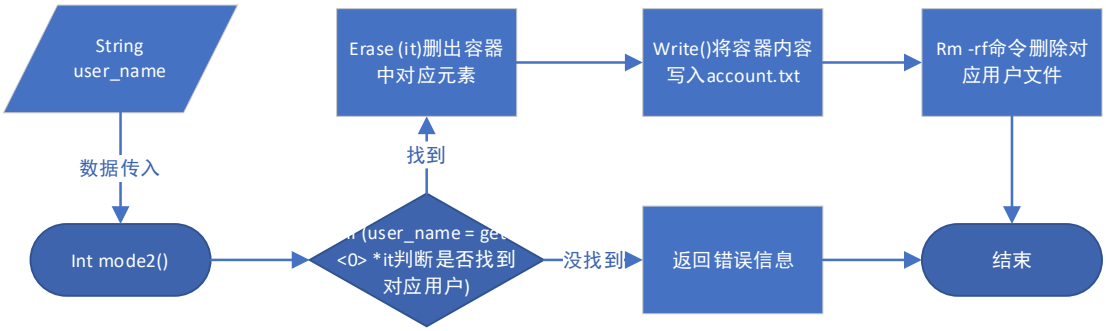
（6） **函数设计：**

① **int main()**：输入操作类型、具体内容。Opt 储存操作类型，user_name 储存输入的用户名，password 储存输入的密码，根据不同的操作类型，执行不同函数。

② **void mode1 (string user_name, string password)**：输入 user_name 与 password，执行加入新账户功能，将输入的内容放入容器中，并写入配置文件。



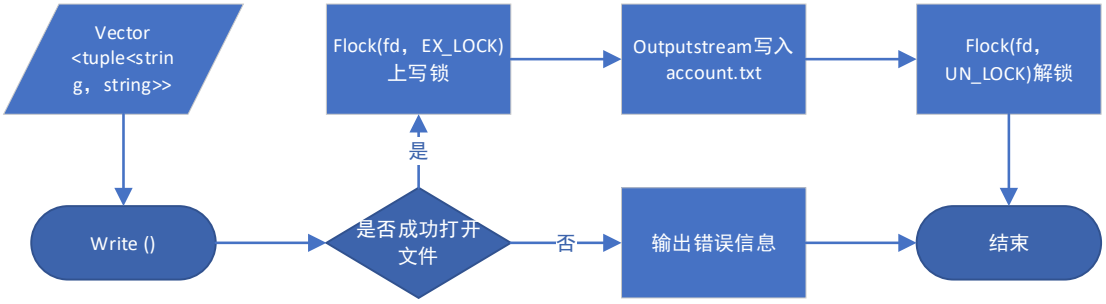
③ **void mode2 (string user_name)**：执行删除功能，根据输入的 user_name，删除容器中对应的账户，并写入配置文件。



④ void mode3 (string user_name, string password)：执行修改功能，根据输入的 user_name，将容器中对应的密码改为输入的 password，并写入配置文件。

⑤ void mode4 ()：执行展示功能，按顺序输出配置文件中的账户密码。

⑥ write()：执行将容器中的内容写入配置文件。



5. 内容过滤模块

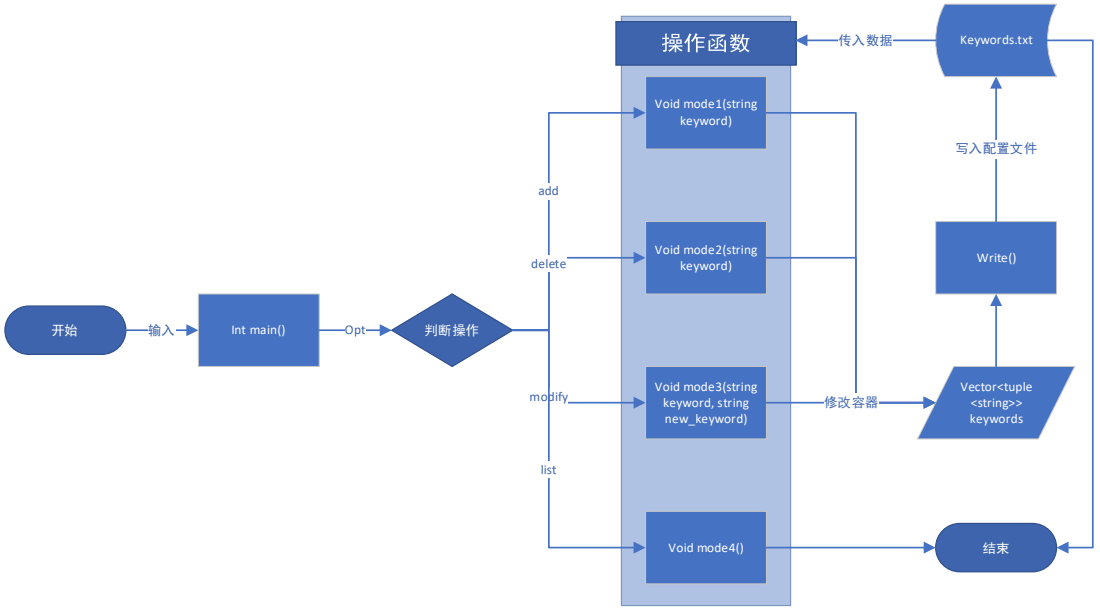
（1） **功能**：实现返回报文进行内容检测，针对特定关键词，做出拒绝传输与通信的决策。

（2） **输入数据**：命令行输入，服务器返回报文与缓存中的返回报文。

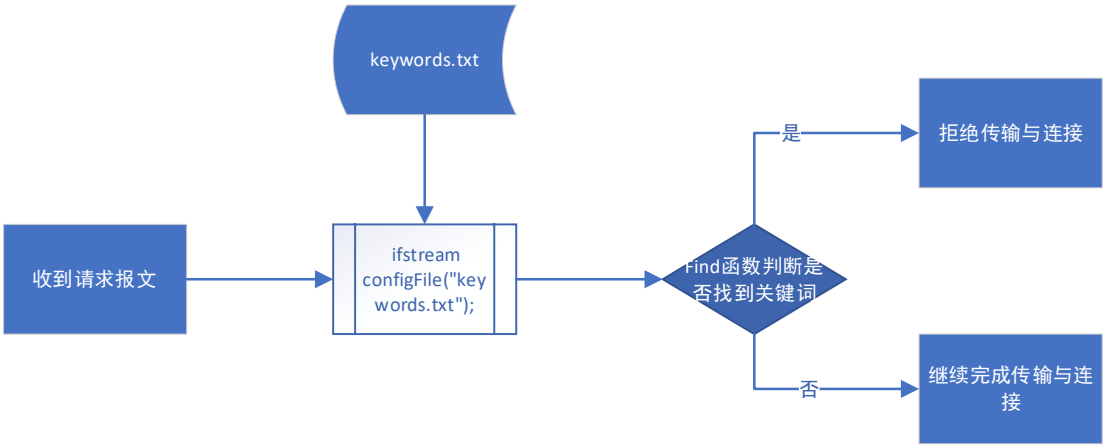
（3） **输出数据**：找到关键词时输出错误信息，配置文件 keywords.txt

（4） **数据处理流程**：

关键词管理程序流程：



报文传输程序流程：



（5） 主要数据结构的设计：

① vector 容器 keywords，储存关键词。

② string keyword，储存输入的关键词

（6） 函数设计：

① **int main()**，输入操作类型、具体内容。Opt 储存操作类型，keyword 储存输入的关键词，new_keyword 储存新关键词，根据不同的操作类型，执行不同函数。

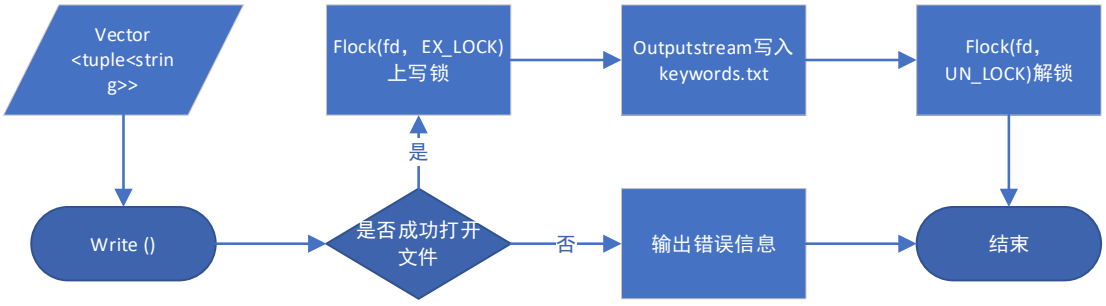
② **void model (string keyword)**：输入 keyword，执行加入新关键词功能，将输入的内容放入容器中，并写入配置文件。Mkdir()函数创建用户文件夹，ofstream()命令创建用户规则和操作日志文件。

③ **void mode2 (string keyword)**：执行删除功能，根据输入的 user_name，删除容器中对应的关键词，并更新配置文件。System()函数利用 rm -rf 命令删除用户对应的用户文件夹。

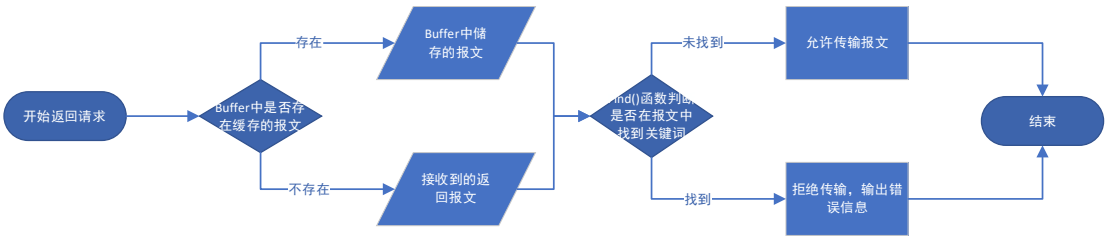
④ **void mode3 (string keyword, string new_keyword)**：执行修改功能，根据输入的 keyword，将容器中对应的关键词改为输入的 new_keyword，并写入配置文件。

⑤ **void mode4 ()**：执行展示功能，按顺序输出配置文件中的关键词。

⑥ **write()**：执行将容器中的内容写入配置文件。



⑦ **void *handleProxyRequest(void *arg)**，在收到服务器返回报文时，读取配置文件，利用 find（）函数判断报文中是否有关键词，如果有，终止连接，并输出错误信息。



6. 日志模块与传输模块

（1） **功能：**在客户端和代理之间传输信息，包括客户端处用户的操作例如创建、注销用户，增删查改控制规则等，并且实时记录用户对于相应规则的修改。

（2） **输入：**用户名、用户进行的具体操作以及相应操作对应的报文中的关键词。

（3） **输出：**不同操作对应的报文以及将用户的操作写入到指定的文件当中。

（4） **主要数据结构的设计：**

日志功能的实现主要包括对于文件控制规则和 http 代理控制规则的记录，主要包括件名称(file_name)、文件类型(file_type)、文件传输方向(file_dire)、文件大小(file_size)和地址(address)、端口号(port)、使用的代理模式(mode)、允许传输的标志(enable)；

传输功能的实现主要包括客户端处用户的相关操作，主要的数据结构包括用户名(username)、用户密码(password)、代理客户端信息(clientSocket)、报文内容(requestBuffer)、客户端 IP 地址(clientIP)。

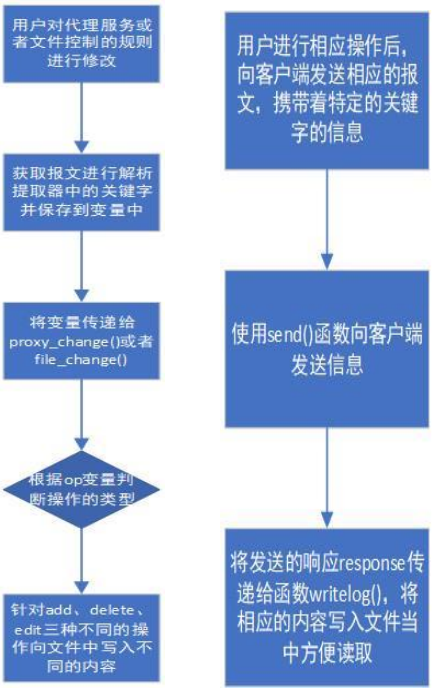
（5） **主要函数设计：**

① **proxy_change()**函数用于记录用户对于 http 代理服务规则的更改的记录，主要包括对三个部分的记录：增添规则、更改规则、删除规则。输入参数为 string 类型的地址、int 类型端口号、string 类型的使用的代理模式、string 类型的允许传输的标志，输出为在文件中写入内容；

② **file_change()**函数用于记录用户对于文件控制的规则的更改的记录，主要包括对三个部分的记录：增添规则、更改规则、删除规则。输入参数为 string 类型的文件传输方向、int 类型的文件大小、string 类型的文件名称、string 类型的文件类型，输出为在文件中写入内容；

③ **writelog()**函数用于把用户进行相关操作后的响应写入到文件中进行保存并方便后续的操作。输入参数为 string 类型的文件路径和 string 类型的响应内容，输出为在文件中写入内容。

④ 依赖的外部函数包括 socket 库中的 send 函数、对文件进行操作的函数、获取字符串长度的函数等。



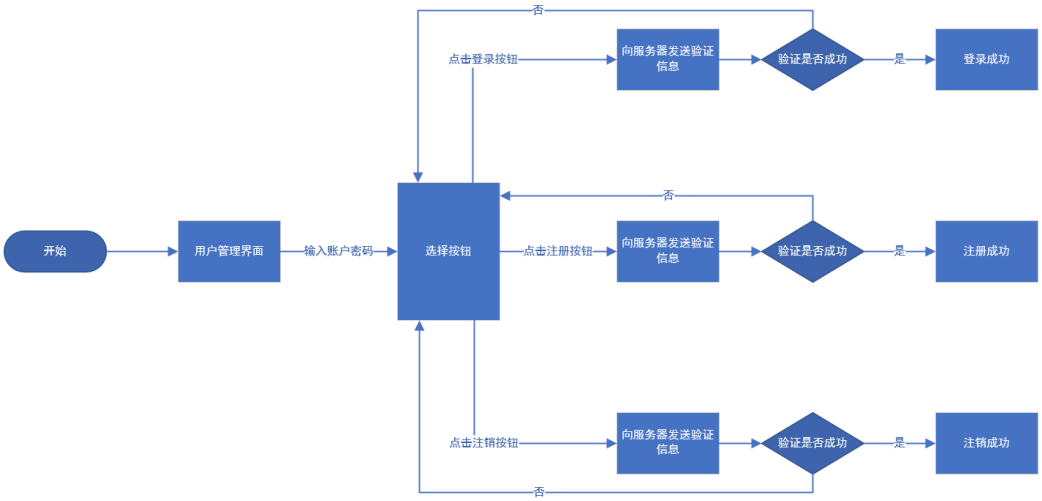
7. 客户端图形界面

客户端图形界面分为**用户管理界面**和**用户使用界面**

（1） **用户管理界面：**

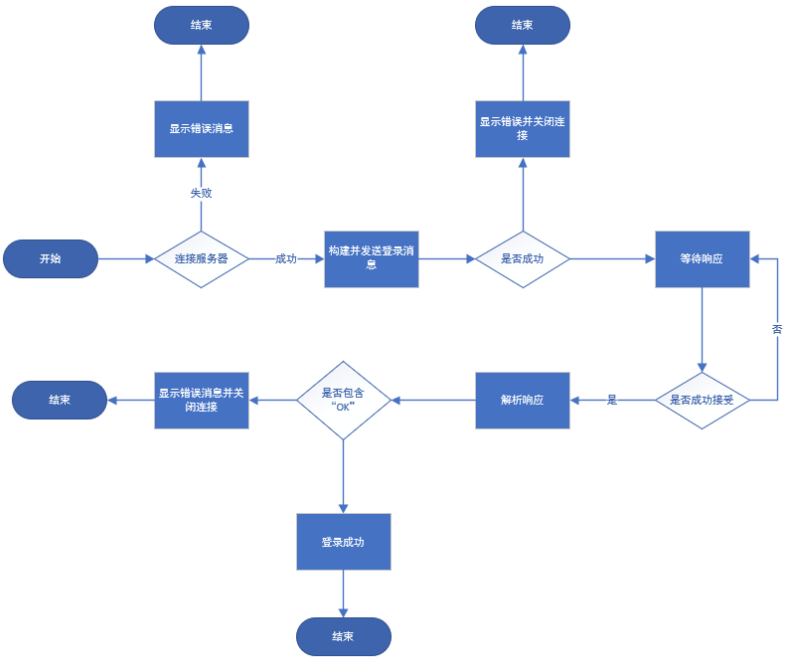
①功能：用户登录，用户注册，用户注销

②数据处理流程：



③主要功能函数：

- `void UserWindow::setupUI()`：目的是创建一个登录页面的用户界面，包括用户名输入框、密码输入框、登录按钮、注册按钮和注销按钮，并利用布局管理器将它们以合适的布局方式显示在窗口上。
- `bool UserWindow::validateLogin(const QString& username, const QString& password)`：目的是用于验证用户登录的有效性



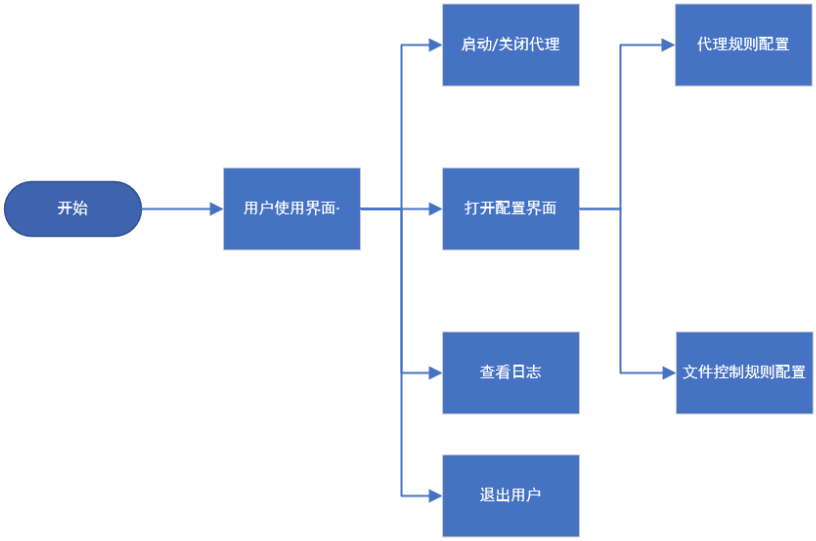
- bool UserWindow::registerUser(const QString& username, const QString& password)：目的是用于验证用户注册的有效性
- bool UserWindow::unregisterUser(const QString& username, const QString& password)：目的是用于验证用户注销的有效性

验证注册和注销有效性的函数逻辑与验证登录有效性函数流程基本一致，不再赘述。

（2） 用户使用界面

①功能：控制规则的启动与停止，查看服务器工作日志，配置代理规则和文件控制规则。

②数据处理流程：

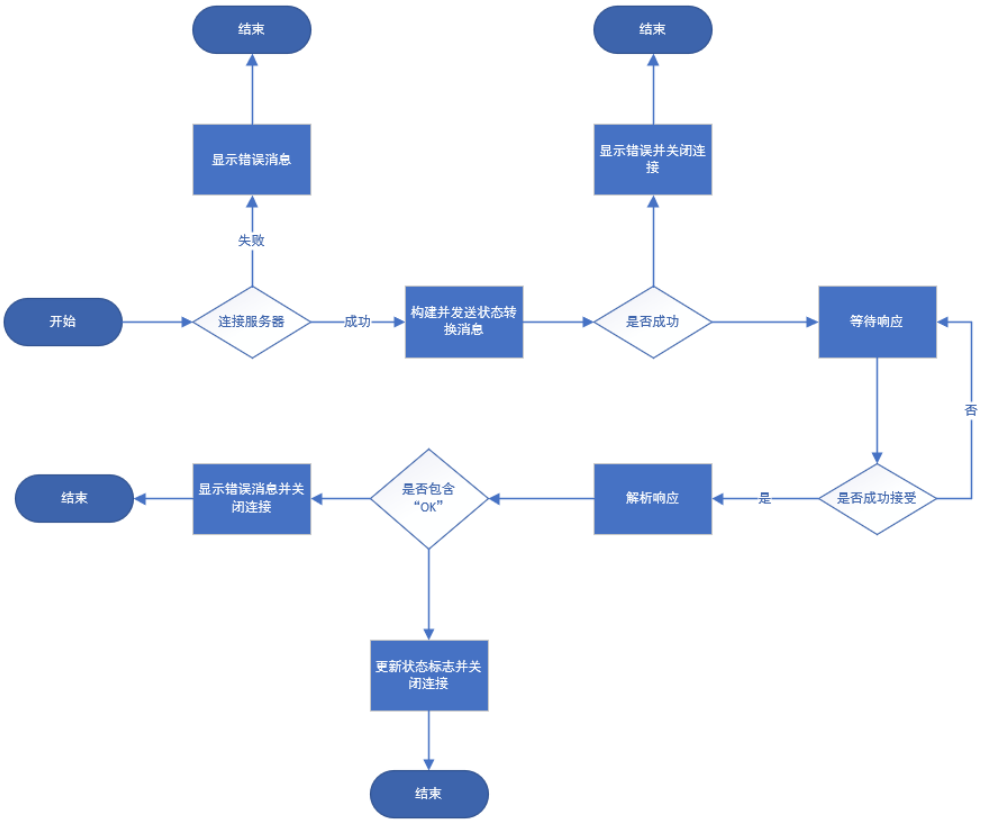


③主要功能函数：

- void MainWindow::createMainPage()：

  主要功能如下：

  1. 设置窗口标题和大小。
  2. 设置背景图，并将其设置为中心部件。
  3. 创建顶部导航栏，并设置导航栏的布局。
  4. 创建状态栏布局，包括当前代理状态的标签和状态图标。
  5. 创建按钮布局，包括启动/停止代理按钮、查看日志按钮、打开配置文件按钮和退出登录按钮。
  6. 设置按钮的样式。
  7. 将导航栏布局和按钮布局添加到主布局中。
  8. 连接启动/停止代理按钮和退出登录按钮的点击事件到对应的槽函数。
  9. 最后将主布局设置为中心部件。
- void MainWindow::toggleProxyStatus()：通过与服务器通信，实现了切换代理状态的功能，并根据服务器的响应更新了代理状态图标

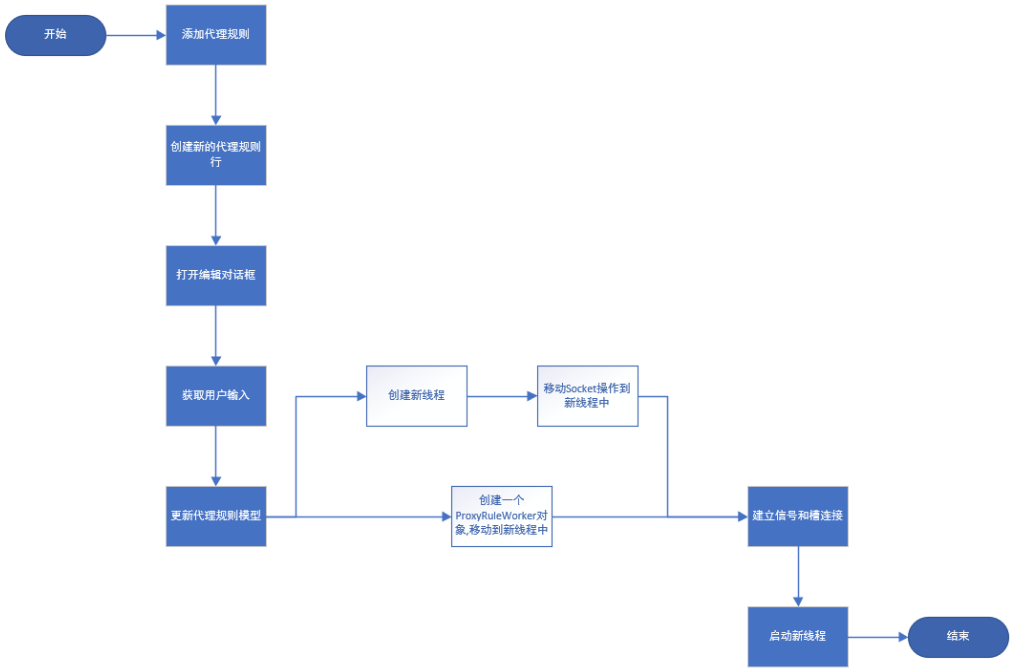


- **void MainWindow::createConfigPage()**：该函数的功能是创建一个配置页面的对话框，其中包含两个选项卡：代理规则和文件控制规则。在代理规则选项卡中，用户可以添加、编辑和删除代理规则。在文件控制规则选项卡中，用户可以添加、编辑和删除文件控制规则。该函数最后显示配置对话框并等待用户交互。
- **void ProxyRuleWorker::runWithSocket()**：该函数主要用来调用以下函数：
  i. void ProxyRuleWorker::addProxyRule(const QString& username,const QString& address, const QString& port, const QString& internet, const QString& enable)：用在服务器添加代理规则，实现流程与 void MainWindow::toggleProxyStatus()一致
  ii. void ProxyRuleWorker::editProxyRule(const QString& username,const QString& address, const QString& port, const QString& internet, const QString& enable)：用在服务器编辑代理规则，实现流程与 void MainWindow::toggleProxyStatus()一致
  iii. void ProxyRuleWorker::deleteProxyRule(const QString& username,const QString& address, const QString& port, const QString& internet, const QString& enable)：用在服务器删除代理规则，实现流程与 void MainWindow::toggleProxyStatus()一致
- **void FileRuleWorker::runWithSocket()**：该函数主要用来调用以下函数：
  i. void FileRuleWorker::addFileRule(const QString& username, const QString& dire, const QString& name, const QString& type, const QString& size)：用在服务器添加文件控制规则，实现流程与 void MainWindow::toggleProxyStatus()一致

ii. void FileRuleWorker::editFileRule(const QString& username, const QString& dire, const QString& name, const QString& type, const QString& size)：用在服务器编辑文件控制规则，实现流程与 void MainWindow::toggleProxyStatus()一致

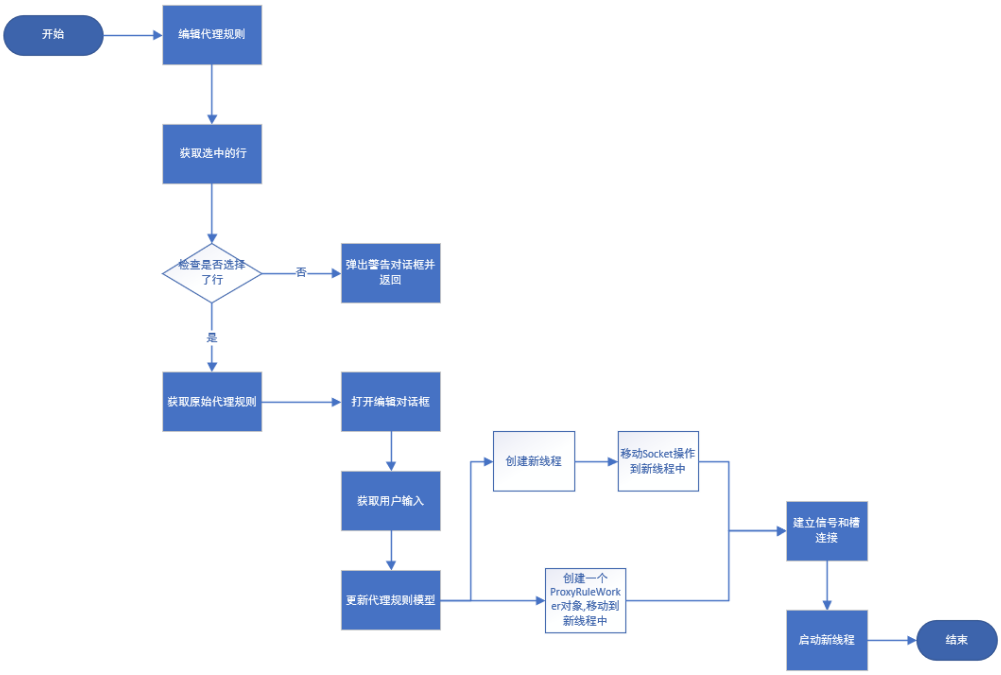
iii. void FileRuleWorker::deleteFileRule(const QString& username, const QString& dire, const QString& name, const QString& type, const QString& size)：用在服务器删除文件控制规则，实现流程与 void MainWindow::toggleProxyStatus()一致

- **void MainWindow::addProxyRule()**：主要功能是添加新的代理规则。它首先创建一个新的代理规则行，并将其添加到数据模型中。然后，它弹出一个编辑对话框，允许用户填写新的代理规则信息。如果用户确认对话框的添加操作，它将获取用户输入的代理规则信息，并将其更新到数据模型中。最后，它将在新线程中执行代理规则操作。

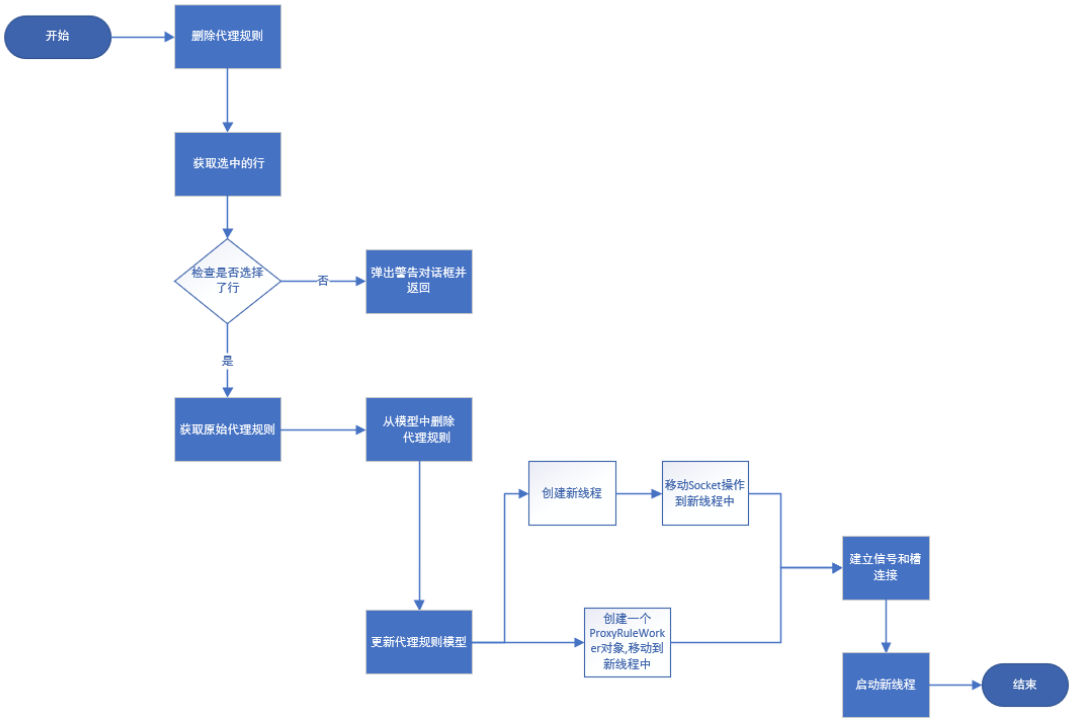


依赖函数：void ProxyRuleWorker::runWithSocket()

- **void MainWindow::editProxyRule()**：主要功能是编辑选定的代理规则。它首先获取选定行的索引，然后从数据模型中获取原始代理规则的信息。接下来，它弹出一个编辑对话框，允许用户编辑代理规则。如果用户确认对话框的更改，它将获取新的代理规则，并将其与原始代理规则进行比较。如果有更改，它将更新数据模型中相应的索引位置。最后，它将在新线程中执行代理规则操作。
  依赖函数：void ProxyRuleWorker::runWithSocket()



- **void MainWindow::deleteProxyRule()**：主要功能是删除选定的代理规则。它首先获取选中行的索引，并从代理规则模型中获取选中行的数据。然后，它从数据模型中删除选中的代理规则行。接下来，它创建一个新的线程和代理规则工作对象 `worker`，并将代理规则操作移动到新线程中。在这里，它设置 `worker` 的删除模式，并建立信号和槽连接。最后，它启动新线程，执行代理规则操作。
  依赖函数：`void ProxyRuleWorker::runWithSocket()`



- **void MainWindow::addFileRule()**：主要功能是添加新的文件控制规则。流程与 `void MainWindow::addProxyRule()` 一致
  依赖函数：`void FileRuleWorker::runWithSocket()`

- void MainWindow::editFileRule()：主要功能是编辑文件控制规则。流程与 void MainWindow::editProxyRule()一致
  依赖函数：void FileRuleWorker::runWithSocket()
- void MainWindow::deleteFileRule()：主要功能是删除文件控制规则。流程与 void MainWindow::deleteProxyRule()一致
  依赖函数：void FileRuleWorker::runWithSocket()

8. 文件控制模块

（1） **功能：**主要的功能为用户在通过 HTTP 方式上传和下载文件时，根据用户所设定的规则对于文件的名称、类型、传输方向、大小进行限制。

（2） **输入及输出：**输入为 HTTP 请求的报文，输出为与用户规则进行匹配，符合规定返回 bool 值 true，若不符合规定则返回 bool 值 false，同时在判断文件传输方向时会返回方向为 upload 或者 download。

（3） **数据结构：**主要包括文件**名称**(file_name)、文件**类型**(file_type)、文件**传输方向**(file_dire)、文件**大小**(file_size)、文件**最大大小**(max_size)、HTTP **请求报文**(buffer)、保存正则表达式**匹配结果**的数组(result)。

（4） **主要功能函数：**

利用正则表达式及 string 中自带的 find 函数与报文中携带的信息进行匹配，没有具体依赖的外部函数。主要包括以下四个函数：

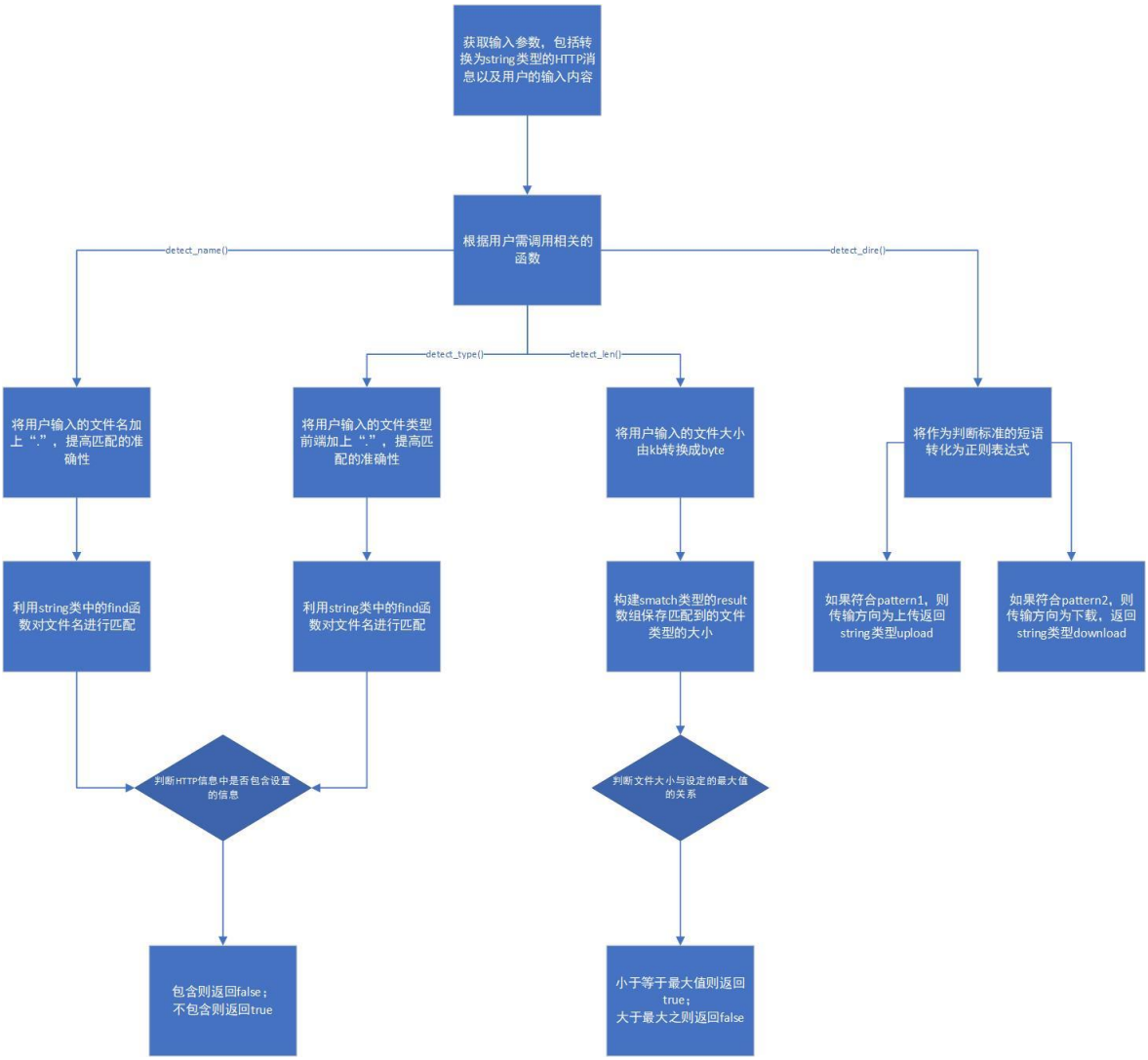
① **bool Filecontrol::detect_name(string message, string filename)**函数用于检测文件传输中文件名称是否符合规则，即是否包含禁止出现的文件名，输入参数为 string 类型的用户设置的文件名和 string 类型的 HTTP 请求报文，输出类型为 bool；

② **string Filecontrol::detect_dire(string message)**函数用于检测文件传输方向，通过 HTTP 消息中特定部分的内容判断，并检测是否与设置的传输方向规则匹配，输入参数为 string 类型的用户设置的文件传输方向和 string 类型的 HTTP 请求报文，输出类型为 string；

③ **string Filecontrol::detect_dire(string message)**函数用于检测文件的类型，通过 HTTP 消息中 Content-Type 头部字段的内容来判断，并检测是否包含禁止出现的类型，输入参数为 string 类型的用户设置的文件类型和 string 类型的 HTTP 请求报文，输出类型为 bool；

④ **bool Filecontrol::detect_len(string message, float length1)**函数用于检测文件大小，通过 HTTP 消息中的 Content-Length 头部字段内容判断，并判断是否超出设置的最大传输大小，输入参数为 float 类型的用户设置的文件大小和 string 类型的 HTTP 请求报文，返回类型为 bool。

以上四个函数的处理流程如下图所示：



9. HTTP 消息解析模块

（1） **功能：**解析 HTTP 报文的信息，将 HTTP 报文的信息保存在一个类中，并提供便于其他模块获取信息的接口。

（2） **输入：**客户端和服务器发送的 HTTP 消息

（3） **输出：**报文数据中的常见内容，包括请求方法、请求 URL、请求协议及版本、请求头部各字段、响应协议及版本、响应状态码、响应状态消息、响应头部各字段。

模块的数据

（4） **主要数据结构：**

HttpParser 类是该源文件的主要类，其将解析结果存储在一个 map 容器中。其私有变量 http 为一个 map 容器，其中每一个键值对均为 string 类型。该类通过解析 HTTP 消息中的内容将每个部分都分为一个键值对，例如在某一 HTTP 请求消息的请求行中，请求方法的键值对为 method 和 GET，请求头部中某一键值对为 Content-Length 和 256。同时该类还包含 HTTP 消息请求行和响应行中各部分的私有变量，包括请求方法、请求 URL、请求协议及版本、

响应协议及版本、响应状态码、响应状态消息。

```
class HttpParser
{
private:
    map<string, string> http;
    string format_key(string &str);
    string method;
    string url;
    string version;
    string status_code;
    string status;

public:
    HttpParser(char *buf); // 解析函数
    ~HttpParser();
    void show(char *msg); // 展示 http 请求报文的所有内容
    string operator[](string str);
    void get_method();  // 获取请求方法
    string get_URL();   // 获取 URL
    void get_version(); // 获取协议及版本
    void get_status_code();
    void get_status();
};
```

（5） 主要的函数：

① HttpParser 类的构造函数 **HttpParser(char *buf)**：

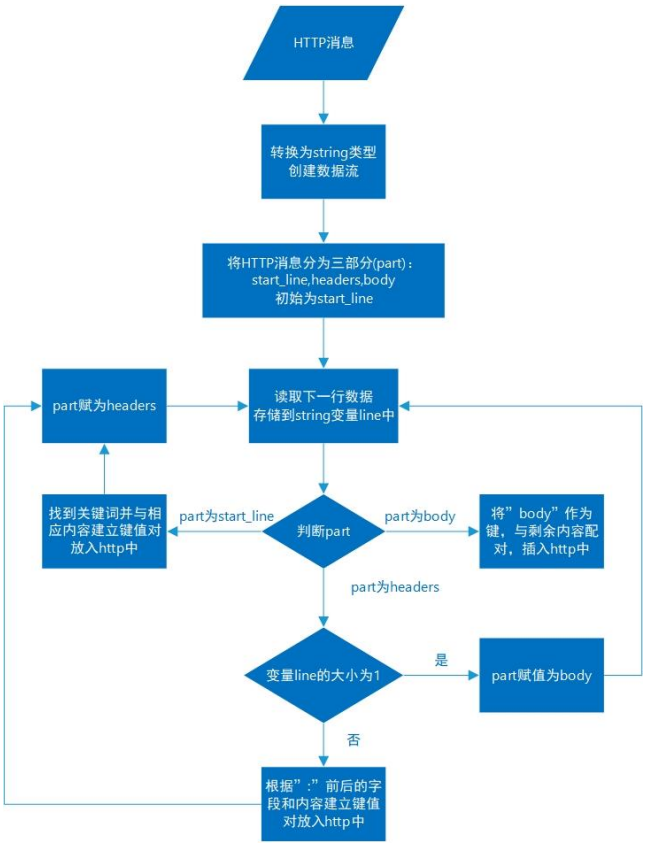
输入：char 类型的指针，其指向 HTTP 消息存储的起始位置。

输出：map 容器私有变量 http。

依赖关系：调用了 map 标准库中的函数，包括查找、配队、插入等函数。

功能：解析 HTTP 消息，根据固定的 HTTP 消息格式将其解析并存储在 http 变量中。

流程图：



② **operator[]重载：**

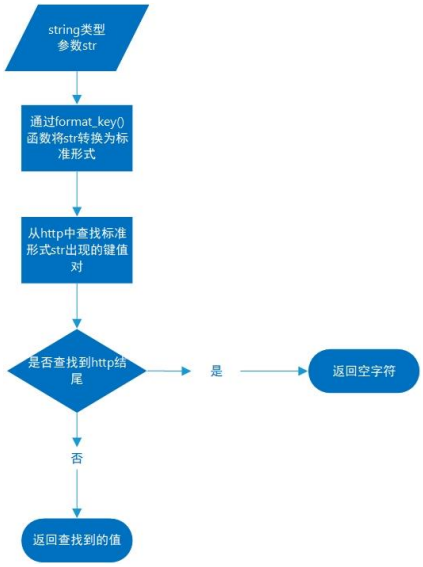
输入：string 类型变量 str。

输出：查找到的字段的键对应的值，即响应字段在 HTTP 消息中的内容。

依赖关系：调用 format_key()函数，将传入的参数转换为标准的形式（单个单词开头字母大写），如将”content-type”转换为”Content-Type”。

功能：在该类的 http 变量的每一个键值对中寻找匹配的键，同时返回该键对应的值。

流程图：



③ **string dom_ip(string hostname)**域名转换 IPv4：

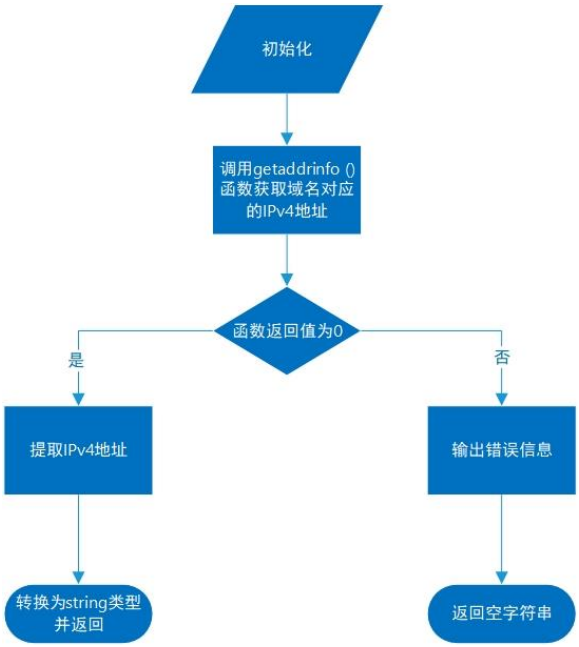
输入：IPv4 地址或域名。

输出：若输入为域名，则输出为对应的 IPv4 地址；若输入本身为 IPv4 地址，则输出为该地址。

依赖关系：在实际调用中需要配合 extract_dom_ip()函数（从 Host 中解析出域名并返回）和 remove_carriage_return()函数（去掉解析出的 string 类型 Host 中的'\r'并返回）使用。

功能：将域名转换为对应的 IPv4 地址。

流程图：



第四章 系统实现

1. 实现环境

（1） 操作系统：Linux
（2） 开发框架：Qt 框架
（3） 编程语言：C++

2. 开发工具

（1） Qt Creator：Qt Creator 是 Qt 官方提供的集成开发环境（IDE），适用于 C++和 Qt 应用程序的开发。它提供了丰富的特性，如代码编辑器、可视化界面设计器、调试器等，可帮助开发者发防火墙应用程序。
（2） Visual Studio Code：由微软开发的免费、跨平台的源代码编辑器。它于 2015 年首次发布，目的是提供一个轻量级但功能强大的开发工具，适用于不同的编程语言和技术堆栈，可以跨平台使用。
（3） CMake：CMake 是一个跨平台的构建工具，可以简化 C++应用程序的构建过程。

3. 源文件情况

（1） **HttpParser.cpp 和 HttpParser.h**

源文件 HttpParser.cpp 和 HttpParser.h 共 6169 字节，244 行，共 37 个符号（包括变量、函数、类、结构体、枚举等）。代码实现了通过传入的 HTTP 消息，解析消息中常见的内容包括请求行中的请求方法、URL 和 HTTP 协议版本、请求头部中的各个字段对应的内容，以及响应行中的协议版本、状态码和状态消息、响应头部中的各个字段对应的内容，并提供输出每个部分的接口，同时包括其他辅助函数。

- **主要的数据结构：**

HttpParser 类是该源文件的主要类，将解析结果存储在一个 map 容器中。其私有变量 http 为一个 map 容器，其中每一个键值对均为 string 类型。该类通过解析 HTTP 消息中的内容将每个部分都分为一个键值对，例如在某一 HTTP 请求消息的请求行中，请求方法的键值对为 method 和 GET，请求头部中某一键值对为 Content-Length 和 256。同时该类还包含 HTTP 消息请求行和响应行中各部分的私有变量。

- **主要的函数：**

① **HttpParser 类**的构造函数接受指向 HTTP 消息存储位置的 char 类型指针，解析消息内容并存储在一个 map 容器中。
② **show()**函数用于打印 HTTP 报文中的所有内容；
③ **operator[]**重载用于获取指定键对应的值，即请求头部和响应头部中各字段对应的内容；
④ **get_method()**函数获取 HTTP 请求行中的请求方法；
⑤ **get_URL()**函数用于获取请求行中的 URL；
⑥ **get_version()**函数用于获取 HTTP 请求行中的协议及版本；
⑦ **get_status_code()**函数用于获取 HTTP 响应的状态码；
⑧ **get_status()**函数用于获取 HTTP 响应的状态消息。
下列函数不属于 HttpParser 类，为辅助其他功能的函数：

① **extract_dom_host()**函数用于从 Host 中解析出域名；
② **remove_carriage_return()**函数用于去除解析出的字符串中的'\r'（由于 HttpParser 类解析出的字段内容最后包括'\r'，需要去除）；
③**dom_ip()**函数用于将解析出的域名解析为 IPv4 地址并返回，若传入的参数已经为 IPv4 地址则返回该 IPv4 地址。

（2） file_transfer.cpp 和 file_transfer.h

源文件 file_transfer.cpp 和 file_transfer.h 共 6662 字节，136 行，共 15 个符号（包括变量、函数、类、结构体、枚举等）。该源文件定义了一个 Filecontrol 类，该类包括四个函数，分别判断传入文件的文件名、文件传输方向、文件类型、文件大小是否符合设置的规则。

- **主要的函数：**

① **detect_name()**函数用于检测文件传输中文件名称是否符合规则，即是否包含禁止出现的文件名；
② **detect_dire()**函数用于检测文件传输方向，通过 HTTP 消息中特定部分的内容判断，并检测是否与设置的传输方向规则匹配；
③ **detect_type()**函数用于检测文件的类型，通过 HTTP 消息中 Content-Type 头部字段的内容来判断，并检测是否包含禁止出现的类型；
④ **detect_len()**函数用于检测文件大小，通过 HTTP 消息中的 Content-Length 头部字段内容判断，并判断是否超出设置的最大传输大小。

（3） file_control.cpp 和 file_control.h

源文件 file_control.cpp 和 file_control.h 共 29862 字节，364 行，共 14 个符号（包括变量、函数、类、结构体、枚举等）。

主要的函数：

① **file_read()**函数通过读取配置文件规则的文件获取文件规则内容，第二个参数为每个元素均为一个 tuple 元组的容器，通过引用将获取的文件规则内容存储于其中。
② **add_file_rule()**函数用于添加一个特定用户的文件规则并保存至相应的文件中，并将各种情况对应的响应报文返回给客户端。
③ **delete_file_rule()**函数用于删除一个特定用户的文件规则并保存至相应的文件中，并将各种情况对应的响应报文返回给客户端。
④ **edit_file_rule()**函数用于修改一个特定用户的文件规则并保存至相应的文件中，并将各种情况对应的响应报文返回给客户端。
⑤ **load_file_rule()**函数用于加载一个指定用户的文件规则，并将规则内容发送给客户端。

（4） account.h 和 account.cpp

源文件 account.h 与 account.cpp 共 18135 字节 ，586 行。该源文件实现了一个简单的 C++网络应用程序，使用 HTTP 协议进行客户端和服务器之间的通信。它包含一些基本的网络功能，如处理注册、登录、注销等请求，并读写文件进行用户账户信息、代理规则等的存储和管理。

- **主要数据结构：**

① **unordered_map<string, string> cip:** 用于存储客户端 IP 地址与用户名之间的映射关系。
② **unordered_map<string, bool> status:** 用于存储用户状态，标记用户是否处于活动会话状态。
③ **unordered_map<string, vector<tuple<string, int, string, string>>> rule:** 存储代理规则，

以用户为键，代理规则为值。

④ **unordered_map<string, vector<tuple<string, string, string, float>>> rule2:** 存储文件规则，以用户为键，文件规则为值。

主要的函数：

① **bool userExists(const string &username):** 检查指定用户名是否已存在于"account.txt"文件中。

② **void writeLog(const string &path, const string &response):** 将响应信息写入指定文件。

③ **string getCurrentTimestamp():** 获取当前的时间戳（格式化为字符串）。

④ **string formatLogMessage(const string &timestamp, const string &level, const string &message, const string &user):** 格式化日志消息。

⑤ **string createRegistrationLog(const string &user):** 创建注册成功的日志消息。

⑥ **string createLoginLog(const string &user):** 创建登录成功的日志消息。

⑦ **string createLogoutLog(const string &user):** 创建注销成功的日志消息。

⑧ **string createExitLog(const string &user):** 创建退出成功的日志消息。

⑨ **bool userRight(const string &username, const string &password):** 检查用户名和密码是否匹配。

⑩ **void handleRegisterRequest(int clientSocket, char *requestBuffer):** 处理注册请求。

⑪ **void handleExitRequest(int clientSocket, char *requestBuffer, string clientIP):** 处理退出请求。

⑫ **void removeUser(const string &username, const string &password):** 从账户文件和磁盘中删除指定用户。

⑬ **void handleLogoutRequest(int clientSocket, char *requestBuffer):** 处理注销请求

⑭ **void handleLoginResponse(int clientSocket, char *requestBuffer, string clientIP):** 处理登录请求。

⑮ **void handleloadRequest(int clientSocket, char *requestBuffer):** 处理加载请求。

⑯ **void handleLogRequest(int clientSocket, char *requestBuffer):** 处理日志请求。

⑰ **void LogClear(int clientSocket, char *requestBuffer):** 清除代理日志文件内容。

（5） rule.h 和 rule.cpp

源文件 rule.h 与 rule.cpp 共 16071 字节，389 行。该源文件中实现的功能是一个简单的代理服务器，具有基于规则的过滤功能。它允许用户通过 HTTP 请求添加、编辑、删除和切换代理规则，并将规则更改记录在日志文件中。

- **主要数据结构：**

① **unordered_map<string, bool> status：**用于存储每个用户的状态，表示代理规则是否启用。

② **unordered_map<string, vector<tuple<string, int, string, string>>> rule：**用于存储每个用户的代理规则，其中每个规则由四个元素组成：地址、端口、模式和启用状态。

- **主要函数：**

① **void proxy_change(string user, string address, int port, string mode, string enable, string op)：**将规则更改记录写入日志文件中。

② **void read(string Filename, vector<tuple<string, int, string, string>> &rule)：**从文件中读取代理规则，并打印每个用户的规则。

③ **void handleAddRuleRequest(int clientSocket, char *requestBuffer)：**处理添加新代理规则的 HTTP 请求，更新规则并保存到文件中。

④ **void handleEditRuleRequest(int clientSocket, char *requestBuffer)**：处理编辑现有代理规则的 HTTP 请求，更新规则并保存到文件中。

⑤ **void handleDeleteRuleRequest(int clientSocket, char *requestBuffer)**：处理删除现有代理规则的 HTTP 请求，从文件中删除规则。

⑥ **void handleToggleRequest(int clientSocket, char *requestBuffer)**：处理切换用户代理规则状态的 HTTP 请求。

（6） account_manage.cpp

源文件 account_manage.cpp 共 7969 字节，315 行。代码实现了通过命令行的方式，在服务器直接对用户进行修改，包括创建用户，删除用户，修改用户密码，展示所有用户的功能，并在创建用户时创建用户文件夹，在删除用户时，删除用户文件夹，同时实现对 account.txt 的加锁，防止多个程序同时使用 account.txt。输入 account.txt 的内容，并将改动内容写入 account.txt

- **主要的数据结构：**

Vector <tuple<string，string>> account 是储存用户名与密码的容器，开始运行时，程序读取 account.txt 的内容，并存入容器中，此后对容器进行操作，完成后，在将容器覆写到 account.txt 中。

- **主要的函数：**

① **int mode1(string user_name, string password)**函数用于在容器中加入新元素，包括用户名与密码，并将容器中元素写入 account.txt。

② **int mode2(string user_name)**函数用于在容器中删除新元素，根据用户名，找到容器中对应元素并删除，并将容器中元素写入 account.txt。

③ **int mode3(string user_name, string password)**函数用于在容器中修改元素，找到用户名对应元素，并将输入的 password 代替容器中储存的 password，并将容器中元素写入 account.txt。

④ **int mode4()**函数用于输出容器中储存的所有元素。

⑤ **write()**函数用于将容器中元素写入配置文件。

⑥ **string pas_sha256(const string password)**函数实现将输入的密码进行加密后以 16 进制形式储存。

- **依赖的外部函数**

① 包括<vector>内的操作函数，如<类名>.push_back(), <类名>.emplace_back()等，用于实现对 account 容器的写入删除等操作。

② <fstream>内的函数，如 ifstream(Filename), ofstream(Filename), isstringstream(Filename)等函数，用于实现对 account.txt 的操作，包括读取内容，写入内容等操作，还包括创建用户文件的部分操作。

③ <sys/stat.h>中的 mkdir(PATH)函数，实现创建用户文件夹。

④ <stdlib.h>中的 system(command)函数，利用“rm -rf”命令，实现删除用户文件夹的功能。

⑤ <sys/file.h>中的 flock(fd，EX_LOCK)函数和 flock(fd, UN_LOCK)函数，实现对 account.txt 的加解锁。

⑥ <openssl/sha.h>中的 SHA256_Init()，SHA256_Update()，SHA256_Final()函数，实现对密码的 sha256 加密。

（7） keywords_manage.cpp

源文件 keywords_manage.cpp 共 6554 字节，254 行。代码实现了利用命令行工具实现

对关键词文件 keywords.txt 的操作，包括增加、删除、修改、展示等功能。

- **主要的数据结构：**

  Vector<tuple<string>>容器，用于储存关键词。

- **主要的函数：**

① **int mode1(string keyword)**函数用于在容器中加入新元素，即 keyword，并将容器中元素写入 keywords.txt。

② **int mode2(string keyword)**函数用于在容器中删除新元素，根据关键词，找到容器中对应元素并删除，并将容器中元素写入 keywords.txt。

③ **int mode3(string keyword, string new_ keyword)**函数用于在容器中修改元素，找到用户名对应元素，并将输入的 new_keyword 代替容器中储存的 keyword，并将容器中元素写入 keywords.txt。

④ **int mode4()**函数用于输出容器中储存的所有元素。

⑤ **write()**函数用于将容器中元素写入配置文件。

- **依赖的外部函数**

① 包括<vector>内的操作函数，如<类名>.push_back(), <类名>.emplace_back()等，用于实现对 keywords 容器的写入删除等操作。

② <fstream>内的函数，如 ifstream(Filename), ofstream(Filename), isstringstream(Filename)等函数，用于实现对 keywords.txt 的操作，包括读取内容，写入内容等操作，还包括创建用户文件的部分操作。

③ <sys/file.h>中的 flock(fd，EX_LOCK)函数和 flock(fd, UN_LOCK)函数，实现对 keywords.txt 的加解锁。

④ <string>中的 find()函数，用于搜索报文中是否存在关键词。

⑤ <cctype>中的 isalpha()函数，用于判断，如果找到关键词后，关键词前后是否为字母，以实现全字匹配。

（8） proxy.cpp

源文件 proxy.cpp 共 19741 字节，565 行。代码实现了防火墙主体逻辑的实现，包括监听端口、为每个请求创建子线程进行处理、缓存逻辑、以及整个子线程的处理逻辑，将各个功能模块联系与结合起来。

- **主要的数据结构：**

① **buff 类：**存储缓存具体内容、启用标志、请求次数

② **unordered_map<string, buff> cache：**存储请求报文与缓存类 buff 的二元组。

③ **extern unordered_map<string, string> cip：**存储用户和 IP 地址二元组

④ **extern unordered_map<string, bool> status：**存储用户名与客户端启用状态二元组

⑤ **extern unordered_map<string, vector<tuple<string, int, string, string>>> rule：**存储用户名与其对应的控制规则。

⑥ **extern unordered_map<string, vector<tuple<string, string, string, float>>> rule2：**存储用户名与其对应的文件控制规则二元组。

- **主要的函数：**

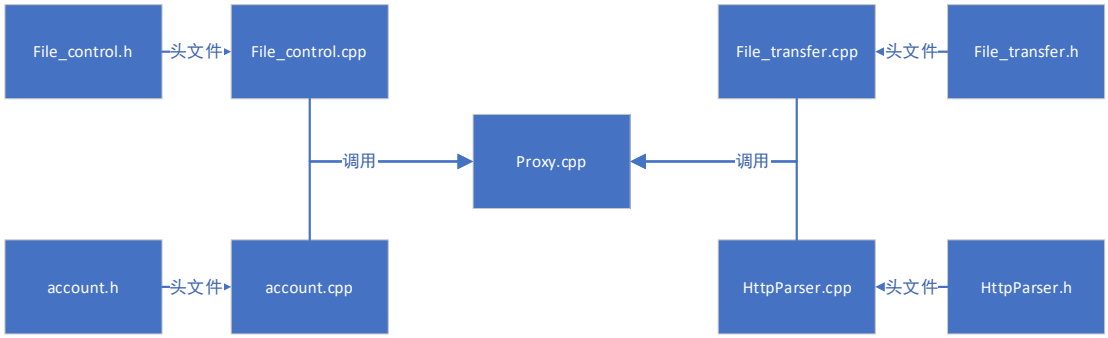
① **int main()** 输出防火墙 logo 到交互界面，绑定监听套接字，创建缓存频数清零线程，为每个监听到的请求创建子线程来处理。

② **void LFU()** 用于实现 LFU 缓存淘汰算法。

③ **void *handleProxyRequest(void *arg)** 处理代理请求的子线程，调用每个模块的接口函数，实现具体的用户操作、规则设置、访问请求控制。

④ **void *zero(void *arg)** 用于定时清零缓存频数的线程。

- **与其他源文件之间的调用关系：**



（9） user.h 和 user.cpp

源文件 user.h 和 user.cpp 包含共 10992 字节，359 行代码，实现了创建一个用户登录界面,包含用户注册、登录、注销等功能。使用 QTcpSocket 与服务器通信验证用户信息。登录成功后启动主窗口界面。

- **主要的数据结构：**

① **UserWindow 类:** 具有设置界面和连接信号槽的方法，并提供了验证登录、注册用户、注销用户的功能。
② **QLabel *backgroundLabel:**背景图片。
③ **QLabel *usernameLabel:**用户名标签。
④ **QLineEdit *usernameLineEdit:**用户名输入框。
⑤ **QLabel *passwordLabel:**密码标签。
⑥ **QLineEdit *passwordLineEdit:**密码输入框。
⑦ **QPushButton *loginButton:**登录按钮。
⑧ **QPushButton *registerButton:**注册按钮。
⑨ **QPushButton *unregisterButton:**注销按钮。

- **主要的函数**

① **void UserWindow::setupUI():**构建界面布局和组件。
② **void UserWindow::setupConnections():**连接信号槽。
③ **QString sha256Hash(const QString &password):** SHA256 哈希运算。
④ **bool UserWindow::validateLogin(const QString& username, const QString& password):**登录验证。
⑤ **bool UserWindow::registerUser(const QString& username, const QString& password):** 用户注册。
⑥ **bool UserWindow::unregisterUser(const QString& username, const QString& password):**用户注销。
⑦ **void UserWindow::handleLogin():**处理登录点击事件。
⑧ **void UserWindow::handleRegister():**处理注册点击事件。
⑨ **void UserWindow::handleUnregister():**处理注销点击事件。

（10） mainwindow.cpp

源文件 mainwindow.cpp，包含共 10846 字节，319 行代码，实现了创建主界面布局的功能,包含导航栏、状态栏、按钮等；通过设置布局、创建各种组件、连接信号和槽,实现了主窗口的基本结构和功能,包括代理开关、规则配置、日志查看等。

- **主要数据结构:**

① **MainWindow 类：**用于创建应用程序的主界面，包括设置应用程序的窗口布局和组件创建配置页面、日志页面和主页面、切换代理状态、处理配置页面的点击事件等功能。
② **QLabel:**显示文本、图标等。
③ **QPushButton:**按钮。
④ **QPixmap:**图片。
⑤ **QVBoxLayout:**垂直布局。
⑥ **QHBoxLayout:**水平布局。

- **主要函数:**

① **MainWindow::createMainPage():**创建主页面布局和内容。
② **MainWindow::toggleProxyStatus():**切换代理状态。
③ **MainWindow::setupConnections():**连接信号和槽。
④ **MainWindow::createConfigPage():**创建配置页面。
⑤ **MainWindow::createLogPage():**创建日志页面。
⑥ **MainWindow::logout():**用户登出。
⑦ **MainWindow::setupname(QString name):**设置用户名。
⑧ **MainWindow::closeEvent(QCloseEvent *event):**窗口关闭事件。

（11） configwindow.cpp 和 logwindow.cpp

源文件 configwindow.cpp 和 logwindow.cpp 包含 25963 字节，约 700 行代码，实现了创建规则配置界面布局和日志查看界面的布局功能。

- **主要数据结构:**

① **ProxyRuleDialog 类：**用于显示一个对话框窗口，让用户输入代理规则的相关信息。
② **FileRuleDialog 类:** 用于显示一个对话框窗口，让用户输入文件控制规则的相关信息。
③ **MainWindow 类：**用于创建应用程序的主界面，包括设置应用程序的窗口布局和组件创建配置页面、日志页面和主页面、切换代理状态、处理配置页面的点击事件等功能
④ **QLabel:**用于显示文本。
⑤ **QLineEdit:**用于输入文本。
⑥ **QComboBox:**下拉列表。
⑦ **QPushButton:**按钮。
⑧ **QGridLayout:**网格布局。

- **主要函数:**

a) ProxyRuleDialog 类:

① **ProxyRuleDialog::ProxyRuleDialog(QWidget *parent, const QString &title, const QString &address, const QString &port, const QString &internet,bool enabled):**构造函数
② **QString ProxyRuleDialog::getAddress() const:**获取地址。
③ **QString ProxyRuleDialog::getPort() const:**获取端口。
④ **QString ProxyRuleDialog::getInternet() const:**获取网络协议。
⑤ **bool ProxyRuleDialog::getEnabled() const:**获取启用状态。

b) FileRuleDialog 类:

① **FileRuleDialog::FileRuleDialog(QWidget *parent, const QString &title, const QString &dire, const QString &name, const QString &type,const QString &size):**构造函数。

② **QString FileRuleDialog::getDire() const:**获取传输方向。
③ **QString FileRuleDialog::getName() const:**获取文件名。
④ **QString FileRuleDialog::getType() const:**获取文件类型。
⑤ **QString FileRuleDialog::getSize() const:**获取文件大小。

c) MainWindow 类：

① **void MainWindow::loadproxyrules():**从服务器加载代理规则,并显示到界面上。
② **void MainWindow::loadfilerules():**从服务器加载文件规则,并显示到界面上。
③ **void MainWindow::createConfigPage():**创建配置界面
④ **void MainWindow::addFileRule():**添加文件规则。
⑤ **void MainWindow::editFileRule():**编辑文件规则。
⑥ **void MainWindow::deleteFileRule():**删除文件规则。
⑦ **void MainWindow::addProxyRule():**添加代理规则。
⑧ **void MainWindow::editProxyRule():**编辑代理规则。
⑨ **void MainWindow::deleteProxyRule():**删除代理规则。
⑩ **void MainWindow::createLogPage():**创建日志界面。

（12） ruleworker.cpp

源文件 ruleworker.cpp 共 11364 字节，331 行，通过面向对象的方式封装了代理规则和文件规则的增删改操作,以异步的方式与服务器交互,实现了规则管理的功能。

- **主要数据结构：**

① **ProxyRuleWorker 类:**用于处理代理规则的类。
② **FileRuleWorker 类:**用于处理文件规则的类。

- **主要函数：**

① **ProxyRuleWorker::addProxyRule():**添加代理规则。
② **ProxyRuleWorker::editProxyRule():**修改代理规则。
③ **ProxyRuleWorker::deleteProxyRule():**删除代理规则。
④ **FileRuleWorker::addFileRule():**添加文件规则。
⑤ **FileRuleWorker::editFileRule():**修改文件规则。
⑥ **FileRuleWorker::deleteFileRule():**删除文件规则。

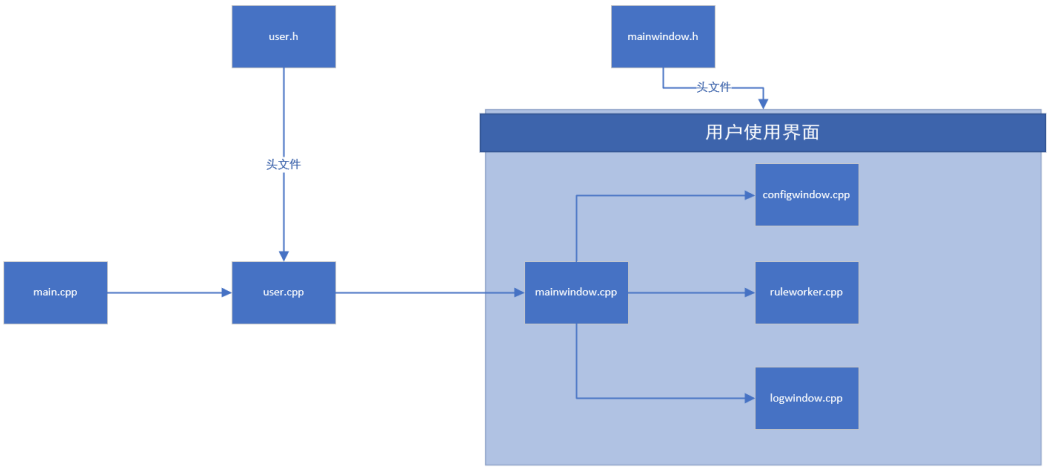
（13） mainwindow.h

源文件 mainwindow.h 共 5617 字节，204 行，声明**类 MainWindow、ProxyRuleDialog、ProxyRuleWorker、FileRuleDialog、FileRuleWorker** 及其成员变量和函数，具体数据结构包含在对源文件 **mainwindow.cpp、ruleworker.cpp、configwindow.cpp 和 logwindow.cpp 的说明中，不再赘述。**

（14） main.cpp

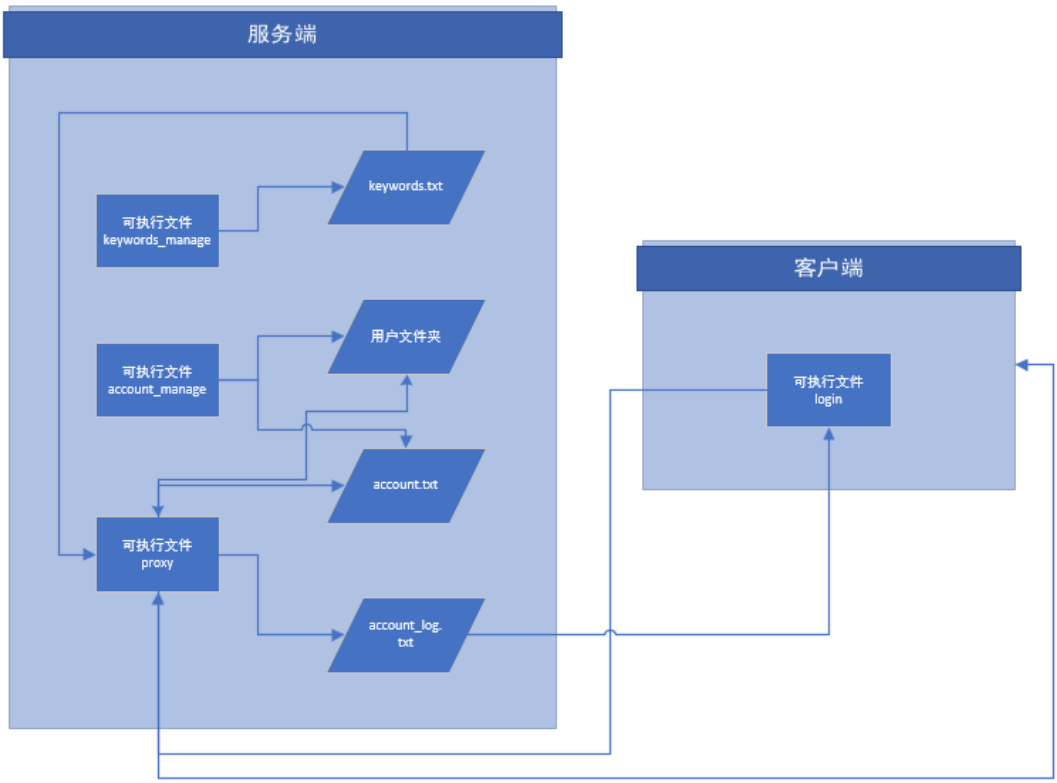
源文件 main.cpp 共 212 字节，14 行，是整个客户端程序的入口。

- **与其他源文件之间的调用关系：**



4. 程序组成与调用流程

目标程序由服务端和客户端**两部分共四个可执行文件**组成，由于文件锁的运用，四个可执行文件可以同时运行，以达到管理员可以在用户使用的同时对关键词、用户进行管理。同时，程序允许存在**多个客户端同时向服务端请求服务**。



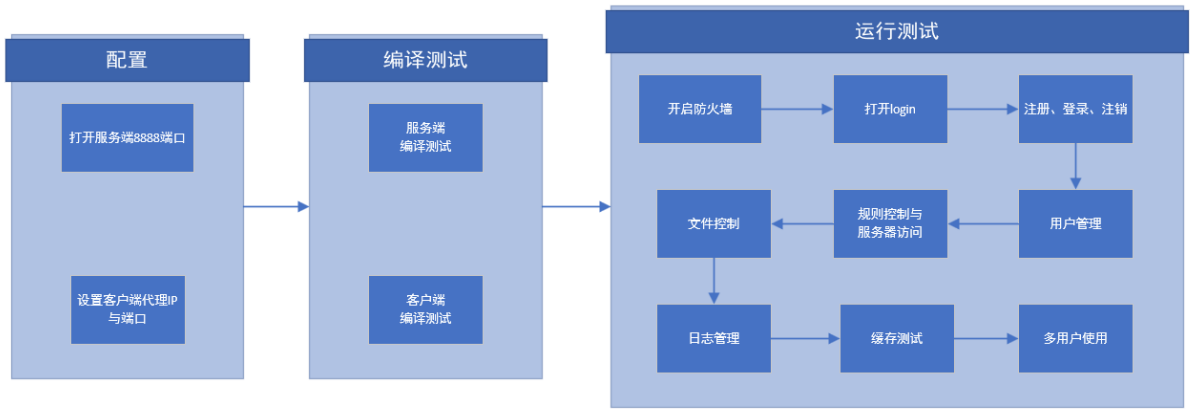
第五章 系统测试

对于以下的所有测试，客户端部署在虚拟机，防火墙及测试网站分别部署在两个不同的服务器，其具体配置信息如下：

功能	操作系统	网卡设置	相关软件及版本	用户及密码
客户端	Ubuntu 18.04	NAT 模式	**Qt** 5.12.8	
服务端	Ubuntu 22.04	公网 IP: 120.27.142.144	**g++** 11.3.0 **cmake** 3.22.1	**用户：** root **密码：** Abc123456
网页部署端	Ubuntu 22.04	公网 IP: 124.221.74.62	**Apache** 2.4.1 **PHP** 7.4.3	**用户：** ubuntu **密码：** Abc123456-

注：服务端和网页部署端可以根据提供的用户名和密码通过 xhsell 远程登录进行测试。其中，防火墙源码在 120.27.142.144 服务器/root/proxy 目录下，网页部署在 124.221.74.62 服务器/var/www/html 目录下。

1. 测试流程



2. 编译测试

（1） 服务器的编译：

防火墙主体：输入 *sh install.sh* 编译源代码。

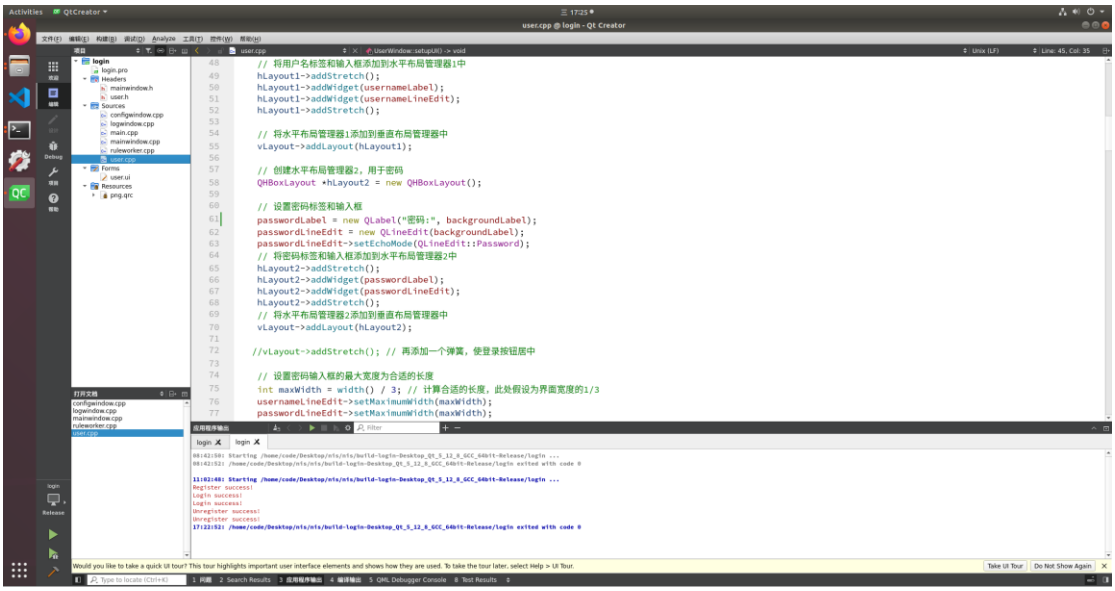
```
root@iZbp148ioksqn21y5djizpZ:~/proxy-test# sh install.sh
CMake Deprecation Warning at CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 2.8.12 will be removed from a future version of
  CMake.

  Update the VERSION argument <min> value or use a ...<max> suffix to tell
  CMake that the project does not need compatibility with older versions.


-- The C compiler identification is GNU 11.3.0
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /root/proxy-test
Consolidate compiler generated dependencies of target proxy
[ 14%] Building CXX object CMakeFiles/proxy.dir/HttpParser.cpp.o
[ 28%] Building CXX object CMakeFiles/proxy.dir/account.cpp.o
[ 42%] Building CXX object CMakeFiles/proxy.dir/file_control.cpp.o
[ 57%] Building CXX object CMakeFiles/proxy.dir/file_transfer.cpp.o
[ 71%] Building CXX object CMakeFiles/proxy.dir/proxy.cpp.o
[ 85%] Building CXX object CMakeFiles/proxy.dir/rule.cpp.o
[100%] Linking CXX executable proxy
[100%] Built target proxy
```

内容过滤模块以及用户管理模块：

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# g++ account_manage.cpp -o account_manage
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# g++ keywords_manage.cpp -o keywords_manage
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ls
account_manage  account_manage.cpp  keywords_manage  keywords_manage.cpp
```

（2） 客户端的编译：

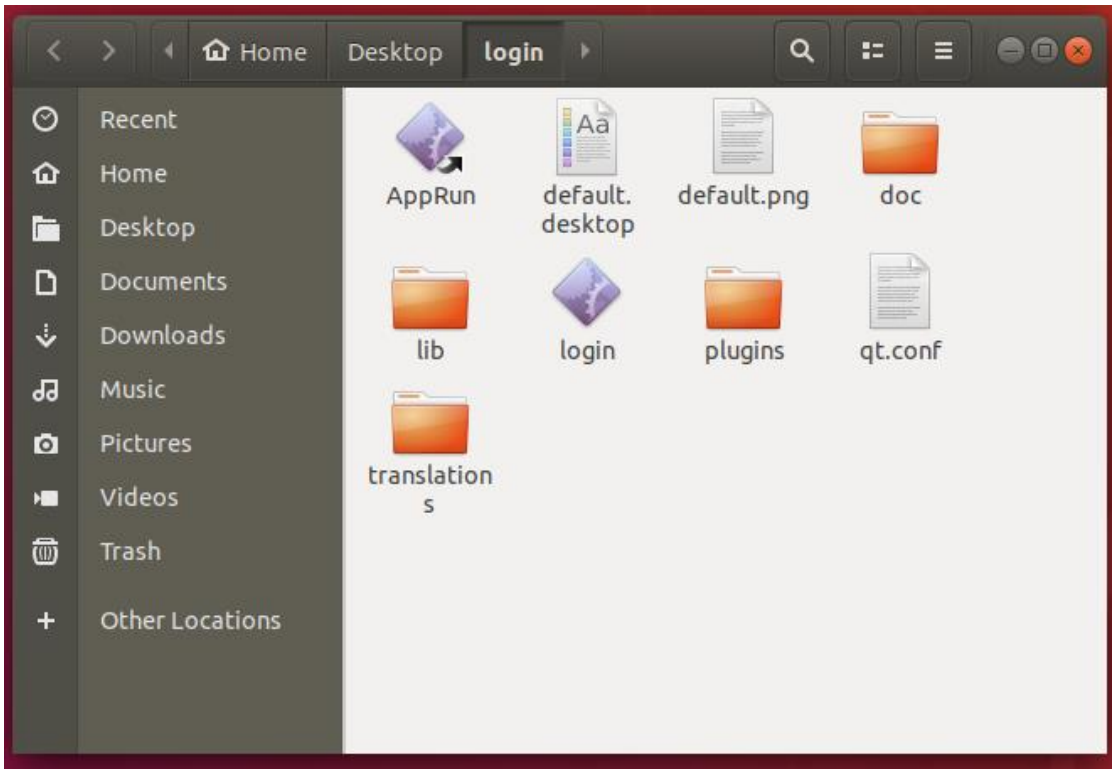
通过 Qt 直接生成可执行文件：



将可执行文件放入一个空白文件夹，使用命令 **linuxdeployqt login -appimage -always-overwrite**，将所有依赖和可执行文件都打成可执行文件，使得程序运行环境无需安装 Qt。

```
code@code:~/Desktop/login$ linuxdeployqt login -appimage -always-overwrite
linuxdeployqt  (commit 8428c59), build 47 built on 2023-04-23 17:29:33 UTC
Not using FHS-like mode
app-binary: "/home/code/Desktop/login/login"
appDirPath: "/home/code/Desktop/login"
relativeBinPath: "login"
Keeping existing AppRun
qmakePath 3= ""
appimagetool, continuous build (commit 8bbf694), build <local dev build> built o
n 2020-12-31 11:48:33 UTC
fatal: not a git repository (or any of the parent directories): .git
Failed to run 'git rev-parse --short HEAD: Child process exited with code 128 (c
ode 128)
Desktop file: /home/code/Desktop/login/default.desktop
Categories entry not found in desktop file
.desktop file is missing a Categories= key
```

打包结果如图:



3. 运行测试

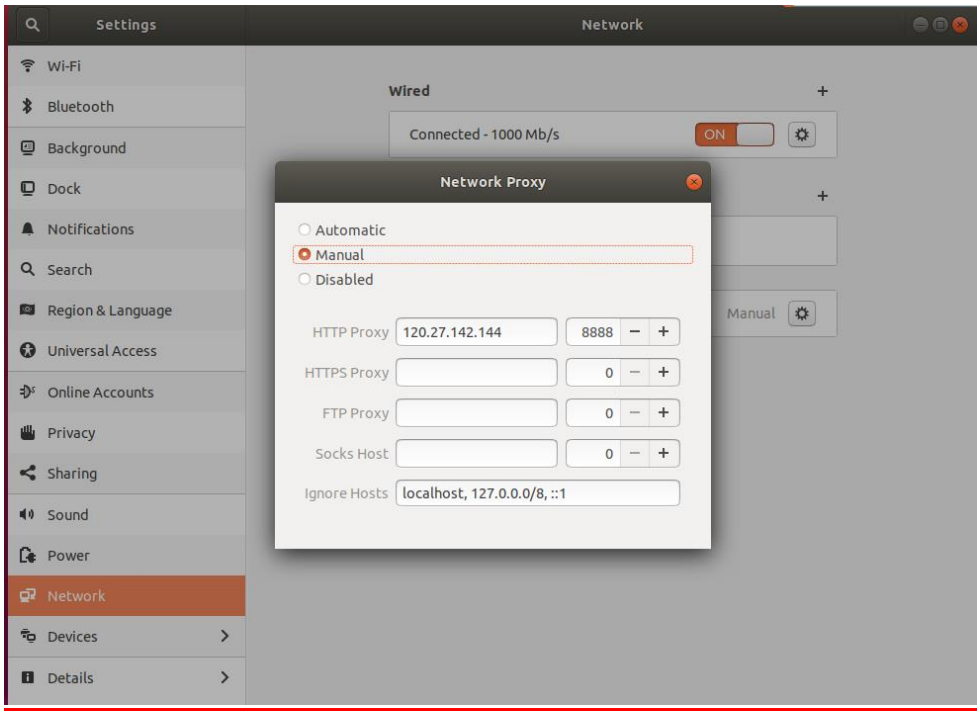
所有测试进行前都必须在服务端启动应用代理防火墙：

在使用 Ctrl+C 关闭防火墙之后，需要等待监听套接字断开连接才能进行下一次测试！

```
root@iZbp148ioksqn21y5djizpZ:~/proxy# ./proxy
Server started.
```

所有测试进行前都必须设置客户端的代理地址为服务端公网 IP，端口为 8888：

最好关闭客户端浏览器缓存功能，方法见 README.md。



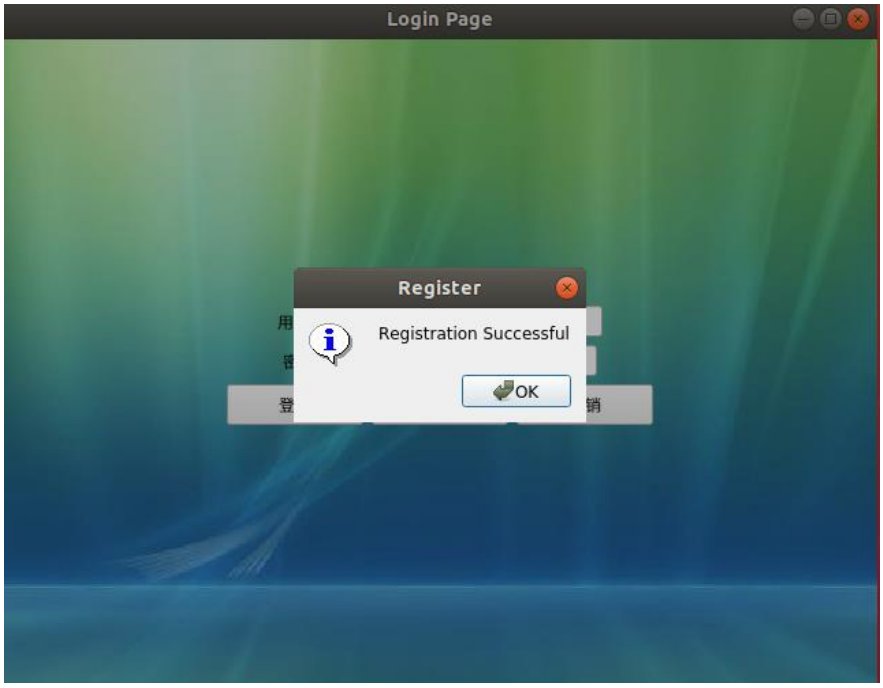
（1）注册、登录、注销功能：

① 正常注册、登录测试

打开客户端应用程序，进入用户管理界面

在输入框内输入账户密码，注册一个**用户名、密码均为** cgy 的用户

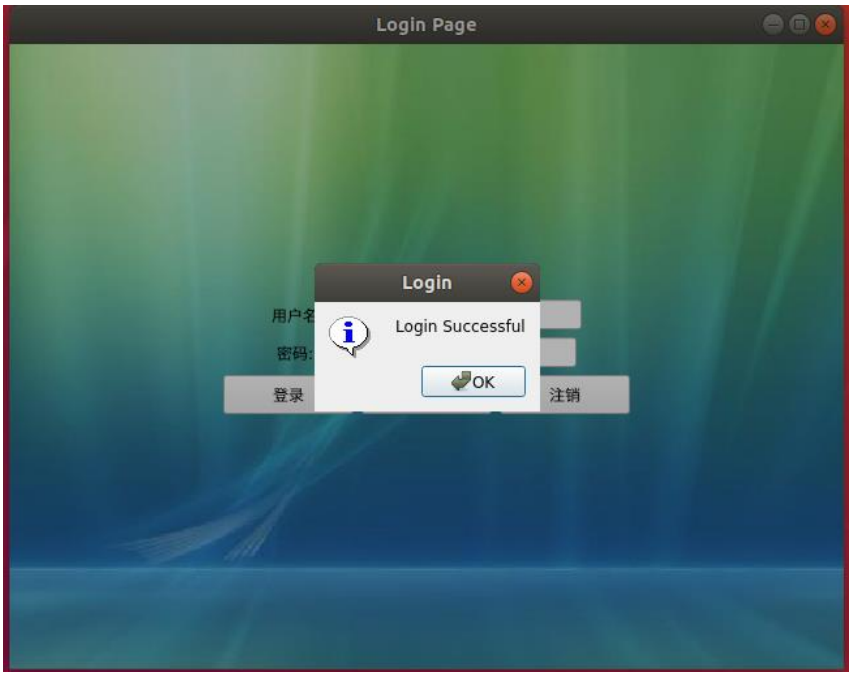
点击**注册**按钮，观察客户端和服务器的响应：




```
Server started.
POST /register HTTP/1.1
Host: 120.27.142.144
Content-Length: 69

username=cgy&password=0d5ec275c1539445da1249fd2b3d30631984b6878a838304c549e361c96e98f3
File created successfully: ./cgy/proxy_rules.txt
File created successfully: ./cgy/proxylog.txt
File created successfully: ./cgy/file_rules.txt
```

可以看到**用户已经注册成功**。

点击**登录**按钮：

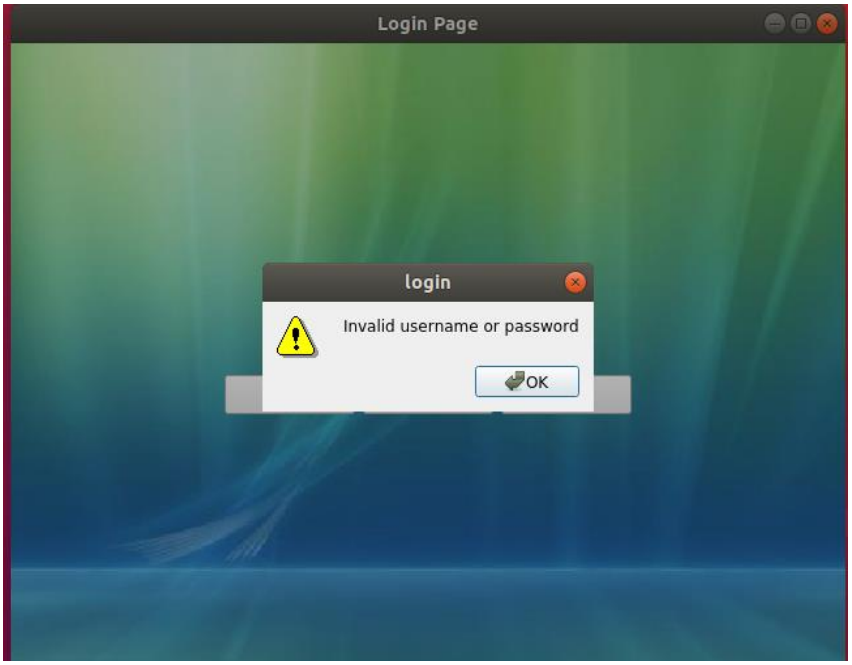



```
POST /login HTTP/1.1
Host: 120.27.142.144
Content-Length: 69

username=cgy&password=0d5ec275c1539445da1249fd2b3d30631984b6878a838304c549e361c96e98f3
./cgy/proxy_rules.txt
./cgy/file_rules.txt
```

成功登录并**进入用户使用界面**：

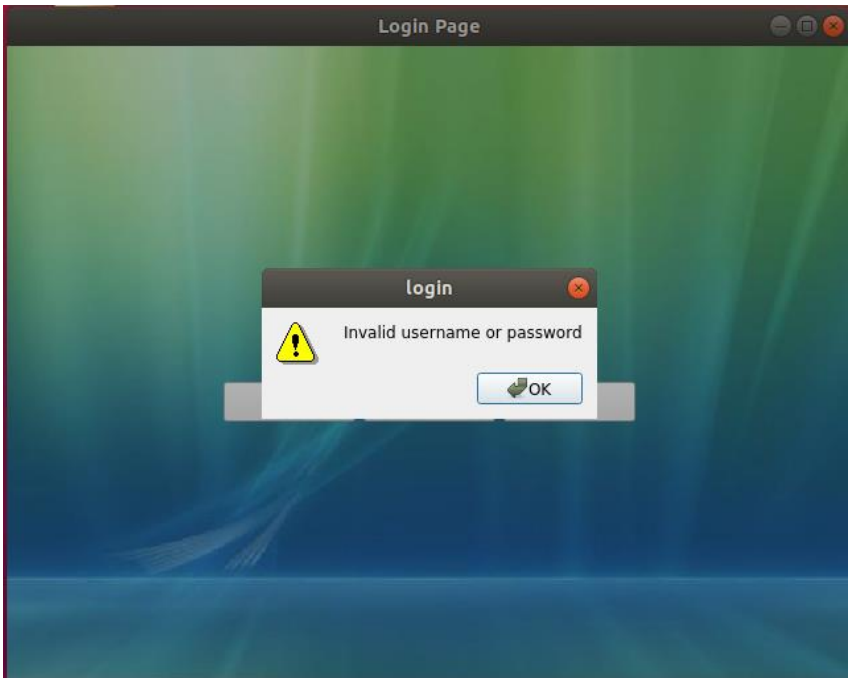
② **错误密码测试：**

还是对于用户 cgy 测试，输入错误密码 xixi，客户端响应如下：



③ **未注册用户测试：**

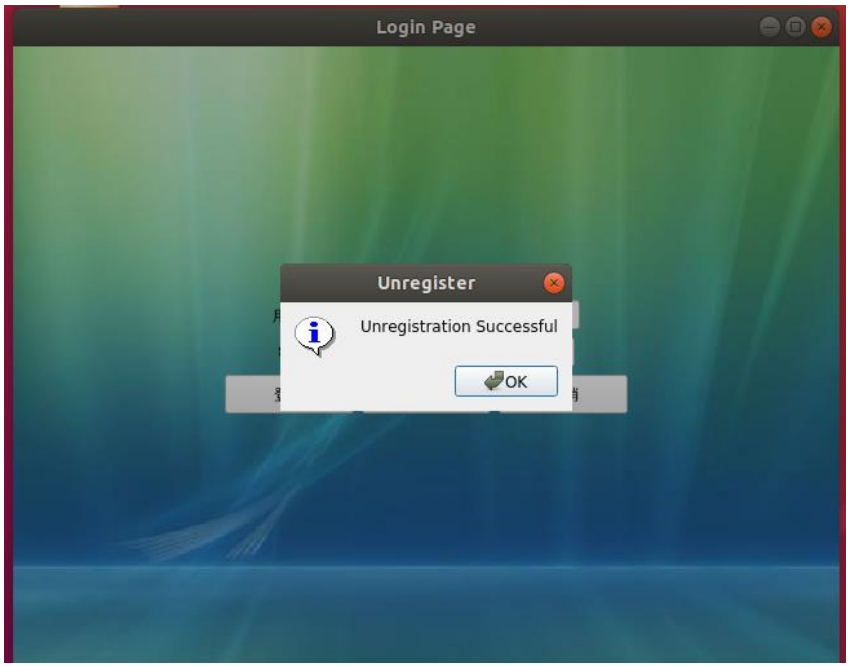
输入未注册用户 xyf，密码 xyf：



注册后即可成功登录（结果与测试①一致，不再赘述）

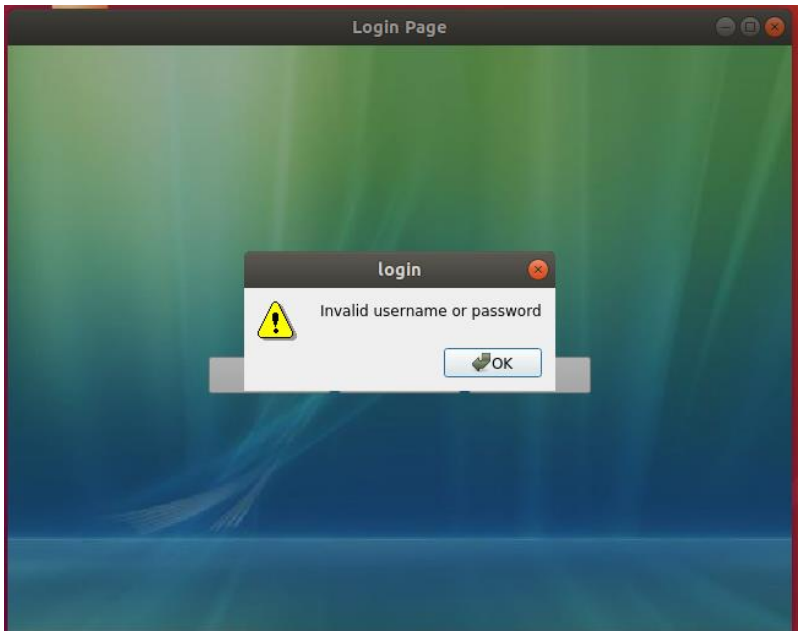
④ 注销测试：

注销用户 cgy：




```
POST /logout HTTP/1.1
Host: 120.27.142.144
Content-Length: 69

username=cgy&password=0d5ec275c1539445da1249fd2b3d30631984b6878a838304c549e361c96e98f3
User removed
```

再次登录显示失败：



⑤ **多用户注册测试**

一次性注册三个用户，见下表：

用户	密码	加密存储
123	123	a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
abc	abc	ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
test	test	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08

其中，用户名和加密后的密码会被存储到文件 **account.txt** 中：

```
proxy > account.txt
1  Username: 123, Password: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
2  Username: abc, Password: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
3  Username: test, Password: 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
4
```

结论：软件能够正确实现用户的登录、注册、注销与错误处理。

（2）用户管理功能：

实现在服务器上用命令行实现对所有账户的管理。

进入**~/proxy/manage**

输入指令**./account_manage -o <操作> -u <用户名> -p <密码>**进行用户管理，具体参数及用法如下：

```
Usage: ./account_manage
 -o add -u <user_name>, -p <password> ---add a account
 -o delete -u <user_name> ---delete a account
 -o modify -u <user_name>, -p <new_password> ---change password
 -o list ---list all accounts
```

为方便测试，已添加账户 **test1**、**test2**，其中，**test2** 的密码为 **abc123456**。而用户 123、abc、test 是后面的测试需要用到的。

```
account.txt
1  Username: 123, Password: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
2  Username: abc, Password: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
3  Username: test, Password: 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
4  Username: test1, Password: f6e0a1e2ac41945a9aa7ff8a8aaa0cebc12a3bcc981a929ad5cf810a090e11ae
5  Username: test2, Password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
```

① **添加账户**

使用如下指令添加账户 **test3**，密码为：**abc123456。**

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o add -u test3 -p abc123456
Done!
```

进入客户端，登录账户，发现可以正常登录，查看 **account.txt**：

```
account.txt
1  Username: 123, Password: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
2  Username: abc, Password: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
3  Username: test, Password: 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
4  Username: test1, Password: f6e0a1e2ac41945a9aa7ff8a8aaa0cebc12a3bcc981a929ad5cf810a090e11ae
5  Username: test2, Password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
6  Username: test3, Password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
7
```

也成功写入，加密后的密码与 test2 也一样。

② **删除账户**

输入如下指令删除账户 test2

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o delete -u test2
Done!
```

查看 account.txt：

```
account.txt
1   Username: 123, Password: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
2   Username: abc, Password: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
3   Username: test, Password: 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
4   Username: test1, Password: f6e0a1e2ac41945a9aa7ff8a8aaa0cebc12a3bcc981a929ad5cf810a090e11ae
5   Username: test3, Password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
6
```

成功删除，在客户端再次登录，发现登录失败。

③ 修改密码

将 **test1** 的密码修改为 **abc123456**

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o modify -u test1 -p abc123456
Done!
```

查看 account.txt：

```
account.txt
1   Username: 123, Password: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
2   Username: abc, Password: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
3   Username: test, Password: 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
4   Username: test1, Password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
5   Username: test3, Password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
6
```

test1 的密码也与 **test3** 一样。

④ 展示 account.txt 内容

输入如下指令：

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o list
Username: 123, password: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3
Username: abc, password: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Username: test, password: 9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
Username: test1, password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
Username: test3, password: a03c32fcd351cba2d9738622b083bed022ef07793bd92b59faea0207653f371d
```

输出与 **account.txt** 内容一致。

⑤ 错误处理

包括：**创建相同用户名账户、删除不存在的用户、修改后的密码与原密码相同。**

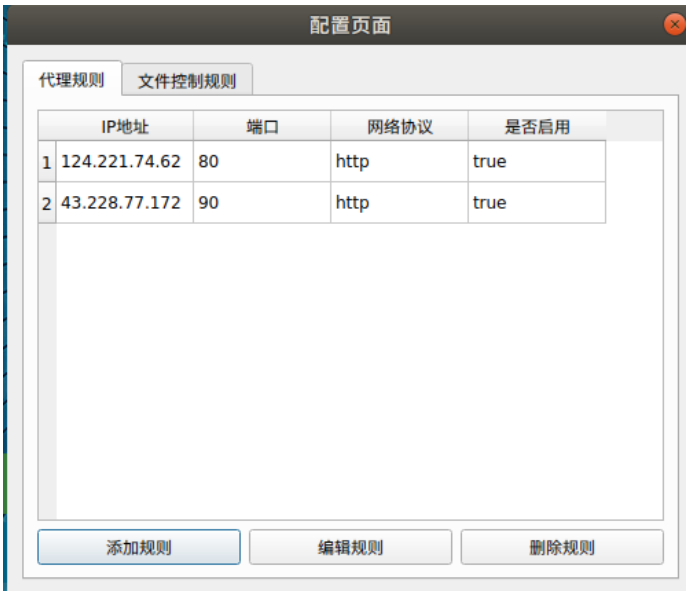
```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o add -u test3 -p abc123456
Account already exists!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o delete -u test2
Not found!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./account_manage -o modify -u test1 -p abc123456
No changes!
```

结论：账户管理模块能够正常运行并达到预期效果。

（3）规则控制与服务器访问：

① 添加规则：

在图形界面点击打开配置文件，添加如下规则：



服务端用户的配置文件也同样得到了更新:

```
proxy > test > proxy_rules.txt
1   Address: 124.221.74.62, Port: 80, Mode: http, Enable: true
2   Address: 43.228.77.172, Port: 90, Mode: http, Enable: true
3
```

② 修改规则：

将第二条规则的端口号修改为 80:

配置文件同样得到了更新:

```
proxy > test > proxy_rules.txt
1   Address: 124.221.74.62, Port: 80, Mode: http, Enable: true
2   Address: 43.228.77.172, Port: 80, Mode: http, Enable: true
3
```

③ 删除规则：

删除第二条规则，配置文件更新同步:

```
proxy > test > proxy_rules.txt
1   Address: 124.221.74.62, Port: 80, Mode: http, Enable: true
2
```

④ **连通性测试**:

访问测试网站 http://124.221.74.62/，**连接成功**。

将规则启用模式改为 **false**，再次访问，**请求被拒绝**。

再将模式改为 true，点击***停止代理按钮***，此时代理状态为**“×”**，同样无法访问:

结论：规则管理以及连通性测试的结果都符合预期，效果良好。

（4）文件控制：

① 添加规则：

添加如下规则：

服务端用户的配置文件也同样得到了更新：

```
proxy > test > file_rules.txt
1  Filedire: 上传, Filename: rule.cpp, Filetype: cpp, Filelen: 2
2  Filedire: 上传, Filename: hello.txt, Filetype: txt, Filelen: 1
3
```

② 修改规则：

将第二条规则中的上传改为**下载**：

服务端用户的配置文件同样得到了更新：

```
proxy > test > file_rules.txt
1  Filedire: 上传, Filename: rule.cpp, Filetype: cpp, Filelen: 2
2  Filedire: 下载, Filename: hello.txt, Filetype: txt, Filelen: 1
3
```

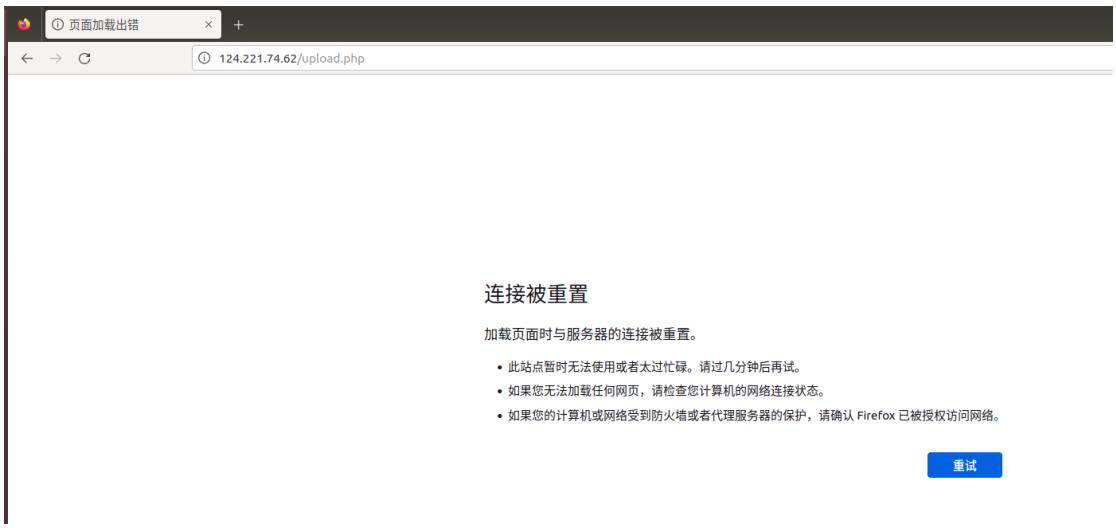
③ 删除规则：

删除**第二条规则**，服务端用户的配置文件得到了更新：

```
proxy > test > file_rules.txt
1   Filedire: 上传, Filename: rule.cpp, Filetype: cpp, Filelen: 2
2
```

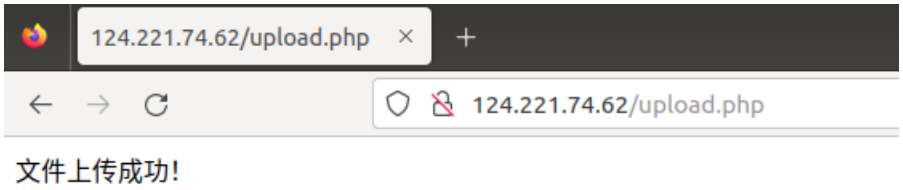
④ **规则有效性测试**:

尝试上传文件 **rule.cpp**，发现上传失败:



再将第二条规则加上，尝试下载文件 **hello.txt**，同样下载失败:

删除第一条规则，尝试上传文件 **rule.cpp**，文件上传成功:



可以在网页部署端看到上传的文件，前缀为时间重命名:

```
ubuntu@VM-4-3-ubuntu:~/uploads$ ls
64b7a69ab457c_rule.cpp
```

使用 vim 打开：

```
int main()
~
~
```

这与虚拟机中文件内容完全相同：

```
home > code > Downloads > rule.cpp
  1   int main()
  2
```

删除第二条规则，尝试**下载文件 hello.txt**

打开文件，显示如下：

原神，启动！！！

这与网页部署端文件内容一致：

```
原神，启动！！！
~
```

结论：与规则控制不同，文件传输控制设置为黑名单形式，根据详细设计中的功能设计，该结果与之相符，实验成功。

（5）日志管理：

① 日志查看：

在客户端点击***查看日志***，可以看到我们刚才的所有操作：

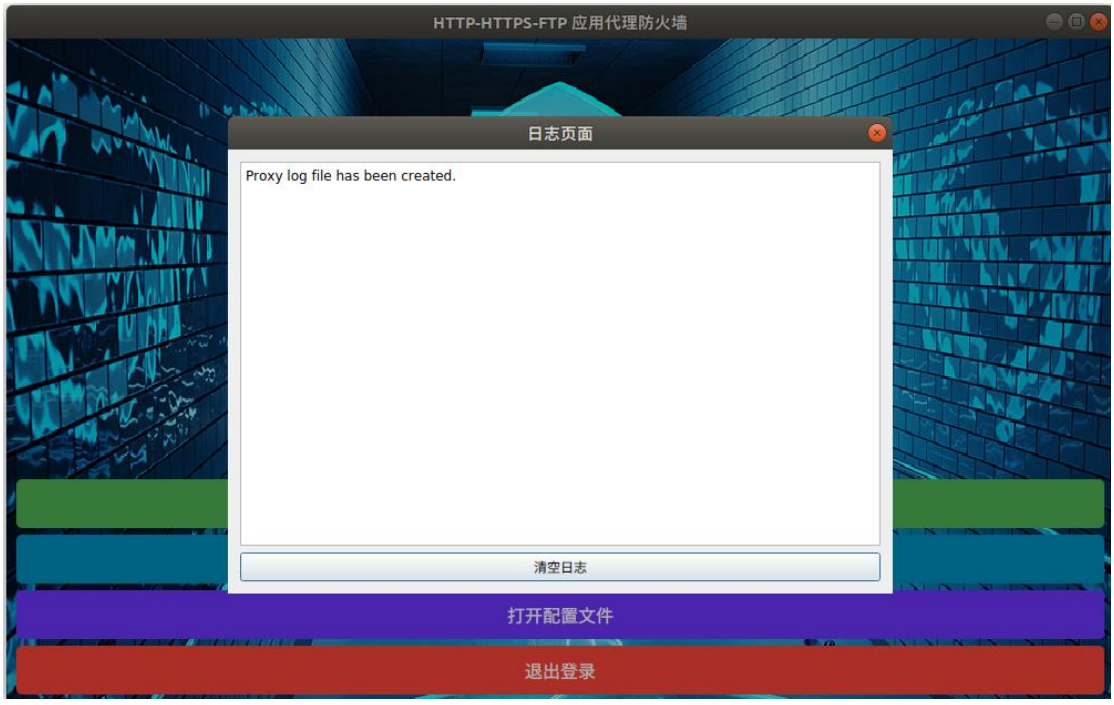
这里的显示与服务端记录完全相同：

proxy > test > proxylog.txt

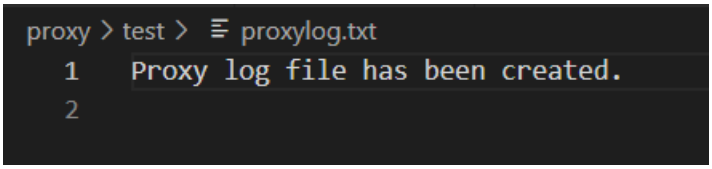
```
1   Proxy log file has been created.
2   Wed Jul 19 16:05:49 2023
3    test add the proxy_rule. Address: 124.221.74.62 Port: 80 Mode: http Enable: true
4   Wed Jul 19 16:06:15 2023
5    test add the proxy_rule. Address: 43.228.77.172 Port: 90 Mode: http Enable: true
6   Wed Jul 19 16:06:22 2023
7    test edit the proxy_rule. Address: 43.228.77.172 Port: 90 Mode: http Enable: true
8   The proxy_rule change to Address: 43.228.77.172 Port: 80 Mode: http Enable: true
9   Wed Jul 19 16:07:45 2023
10   test delete the proxy_rule. Address: 43.228.77.172 Port: 80 Mode: http Enable: true
11  Wed Jul 19 16:10:02 2023
12   test edit the proxy_rule. Address: 124.221.74.62 Port: 80 Mode: http Enable: true
13  The proxy_rule change to Address: 124.221.74.62 Port: 80 Mode: http Enable: false
14  Wed Jul 19 16:10:43 2023
15   test edit the proxy_rule. Address: 124.221.74.62 Port: 80 Mode: http Enable: false
16  The proxy_rule change to Address: 124.221.74.62 Port: 80 Mode: http Enable: true
17  Wed Jul 19 16:18:21 2023
18   test add the file_rule.Filedire: 上传, Filename: rule.cpp, Filetype: cpp, Filelength: 2
19  Wed Jul 19 16:18:58 2023
20   test add the file_rule.Filedire: 上传, Filename: hello.txt, Filetype: txt, Filelength: 1
21  Wed Jul 19 16:21:37 2023
22   test edit the file_rule.Filedire: 上传, Filename: hello.txt, Filetype: txt, Filelength: 1
23  The file_rule change to Filedire: 下载, Filename: hello.txt, Filetype: txt, Filelength: 1
24  Wed Jul 19 16:22:47 2023
25   test delete the file_rule.Filedire: 下载, Filename: hello.txt, Filetype: txt, Filelength: 1
26  Wed Jul 19 16:34:30 2023
27   test add the file_rule.Filedire: 下载, Filename: hello.txt, Filetype: txt, Filelength: 1
28  Wed Jul 19 16:59:49 2023
29   test delete the file_rule.Filedire: 上传, Filename: rule.cpp, Filetype: cpp, Filelength: 2
30  Wed Jul 19 16:59:51 2023
31   test delete the file_rule.Filedire: 下载, Filename: hello.txt, Filetype: txt, Filelength: 1
32
```

② 日志清空：

点击清空按钮，再次打开：



服务端同样得到更新：



结论：日志的存储与管理符合预期。

（6）内容过滤：

① 内容过滤规则管理：

实现在服务器用命令行方式**添加**、**删除**、**修改**关键词。

首先，进入**~/proxy/manage**，为测试方便，在上一级目录中的关键词存储文件**kewords.txt** 中已存在 **111**，**222**，**333** 三个关键词。

运行可执行文件 **keywords_manage** 查看语法格式：

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage
Usage: ./keywords_manage
-o add -k <keyword> ---add a keyword
 -o delete -k <keyword> ---delete a keyword
 -o modify -k <keyword>, -n <new_keyword> ---change keyword
 -o list ---list all keywords
```

- **添加关键词**

添加关键词 **aaa**。

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o add -k aaa
Done!
```

- **删除关键词**

删除关键词 **111**。

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o delete -k 111
Done!
```

- **修改关键词**

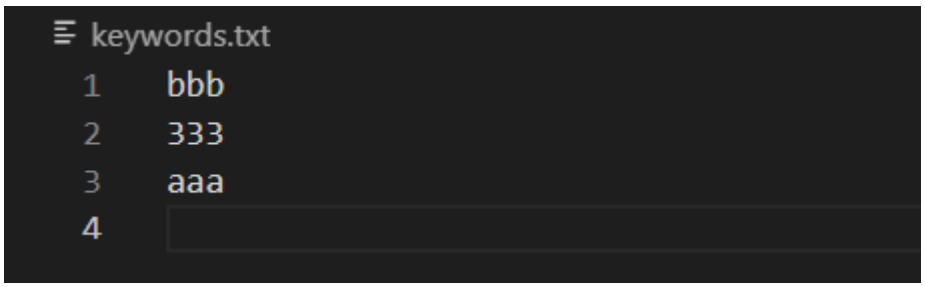
将 **222** 修改为 **bbb**。

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o modify -k 222 -n bbb
Done!
```

- **展示所有关键词**

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o add -k aaa
Done!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o delete -k 111
Done!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o modify -k 222 -n bbb
Done!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o list
bbb
333
aaa
```

查看配置文件，发现与配置文件内容一致



- **错误处理**

包括：**输入重复关键词**，**删除不存在的关键词**，**修改后的词与原词一样**，修改不存在的**关键词**

```
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o add -k aaa
keyword already exists!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o delete -k 111
Not found!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o modify -k bbb -n bbb
No changes!
root@iZbp148ioksqn21y5djizpZ:~/proxy/manege# ./keywords_manage -o modify -k 111 -n bbb
keyword is not existed!
```

② 内容过滤实现

开始测试，此时关键词列表为 **bbb**，**333**，**aaa**，客户端浏览器访问测试网站，反馈如下：

可以正常连接，此时，根据返回报文中的`Accept-Encoding: gzip, deflate`

加上关键词**“gzip”**，此时再连接网站，**服务端**返回如下：

```
Error: Found keyword 'gzip' in server response.
```

客户端浏览器反馈如下：

服务器显示错误信息，客户端无法连接到目标网站。

（7）缓存测试：

① 缓存存储与使用：

登录后刷新页面，观察客户端返回：`Cache used!`，说明缓存存储成功并且成功被使用。

② **缓存淘汰：**

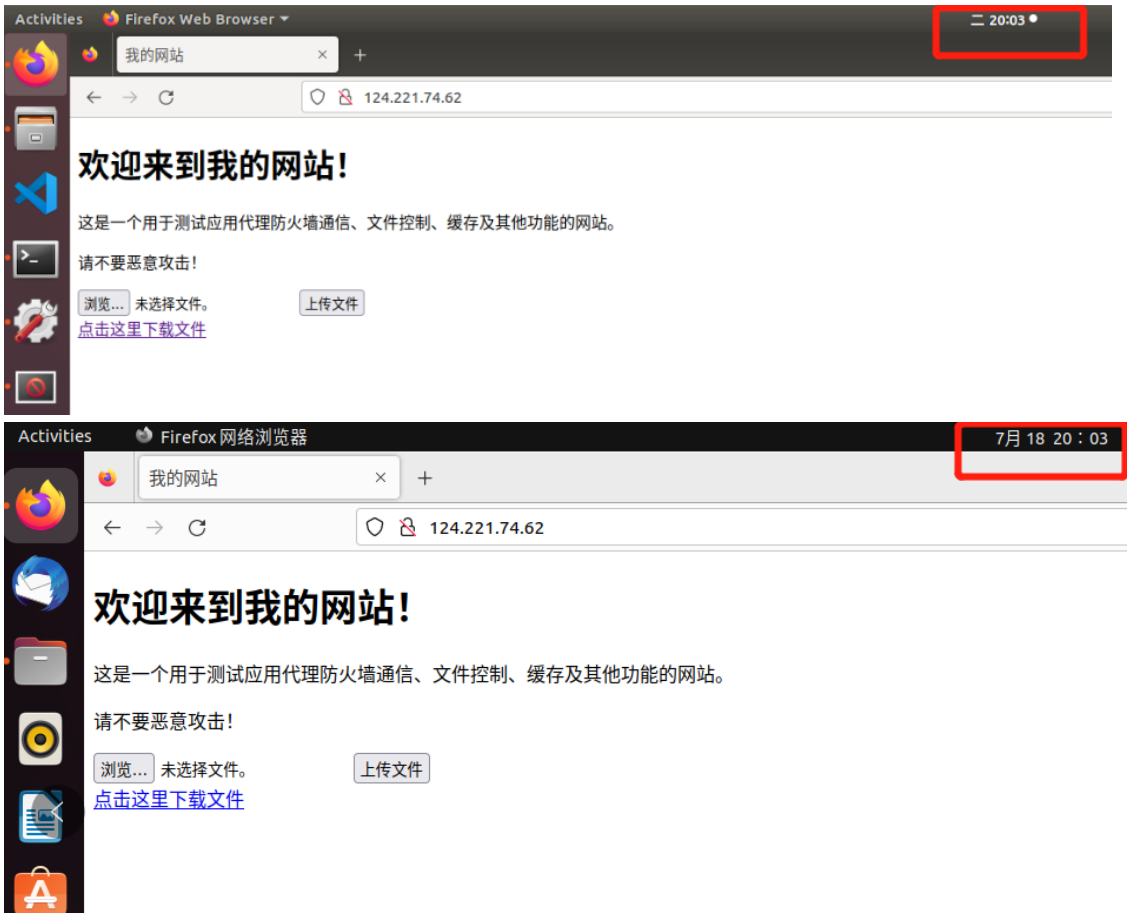
修改源代码，设置缓存最大数为 **3**，同时，使之在交互界面不输出报文，并使存储缓存时，**输出当前缓存数量**。连接测试网站后，连续上传**不同文件，**发现缓存总数稳定在 3（显示 Cache used！是因为需要返回原网页，并且我们关闭了浏览器原本的缓存功能）：

```
0
1
Cache used!
2
Cache used!
3
Cache used!
Cache used!
3
3
Cache used!
```

结论：缓存功能与淘汰功能都可以正常使用。

（8）多用户使用：

使用**分别部署在两台主机上**的虚拟机同时运行客户端，添加相应规则并上网（这样可以保证即使虚拟机是 NAT 模式，客户端识别到的 IP 地址也不一样）。测试结果如下：



结论：可以看到两台不同的虚拟机在**相同时间不同 ip、不同账号**的情况下，都成功访问了测试网站。成功实现了多用户同时使用的功能。

4. 测试总结

综上所述，本次测试验证了软件的主要功能和性能，**覆盖了软件的各个关键模块**,包括**用户管理**、**规则控制**、**文件传输控制**、**日志管理**、**内容过滤**等。测试用例设计合理，覆盖了功能的常规用例、边界用例、异常用例,测试覆盖率高。测试执行顺利，结果良好。尽管存在一些小问题，但整体来说，软件质量良好，达到了预期的效果。我们还提出了改进建议，包括完善 FTP 消息和 HTTPS 消息的有效控制，优化文件控制逻辑、控制范围和消息解析的细节等部分功能,以进一步提升软件的质量和稳定性。在后续的工作中，我们将继续组织测试以监控质量，并根据反馈的问题进行改进和优化。

第六章 项目小结

项目的实施概况：

在本次项目中，我们致力于开发一个应用代理防火墙，以增强网络安全并保护用户免受各类网络威胁。我们已经成功实现了多个重要功能模块，包括 HTTP 消息解析、HTTP 通信、缓存与内容过滤、用户管理、规则控制、传输和日志管理，以及一个友好的图形界面模块。我们的项目包括服务端和客户端。其中，服务端为防火墙主体，主要以命令行的形式控制，实现用户的管理以及代理上网、文件控制、存储配置文件与日志文件等功能，在交互界面返回用户请求与请求结果，可以实现多个用户同时使用代理。而客户端主要提供友好的图形界面，让用户更方便地进行控制规则的配置与代理防火墙的使用。

不足与展望：

然而，我们也要承认在项目实施中存在一些不足。首先，我们未能实现 FTP 和 HTTPS 的代理功能，这对于现代网络环境来说是一个重要的缺失。FTP 和 HTTPS 协议广泛应用于文件传输和加密通信，因此，为了进一步提高防火墙的实用性和全面性，我们需要在未来的工作中加入对这两个协议的支持。其次，由于项目开发周期较紧凑，有些功能可能没有充分优化和测试。这可能导致一些潜在的漏洞和错误未被发现，影响系统的稳定性和安全性。因此，我们应该在项目规划阶段更加充分地考虑测试和优化的时间，确保项目交付的质量和可靠性。

体会与感受：

在项目实施过程中，我们的团队收获了许多宝贵的经验和感悟。在合作中不免会存在一些意见上的冲突，在前期我们也因为没有明确的目标进行了无用的尝试，但在发现问题之后我们很快调整了过来。我们采取讨论协商的方式，每个模块由一位同学主导，所有成员在课堂之余聚在一起交流，每个人都充分给出自己的见解，最后由模块负责人选定实施方式，这样可以确保每个人都知道其他模块要干什么，使得每个模块之间的接口更加合理。因此，在项目实施过程中，团队合作的重要性得到了充分认识，大家的共同努力和紧密配合使得项目得以顺利完成。同时，我们也深刻认识到信息安全科技创新的重要性和挑战性，网络安全已经成为全球关注的焦点，在上完本课程之后，我们对信息安全相关软件的开发目标与流程有了比较清晰的认识，开发软件的能力与编程解决实际问题的能力也有了很大的提升。我相信无论是对今后的科研还是工作，这都将是一次宝贵的经历！

建议与意见：

针对信息安全科技创新这门课程，我们对其表示感谢，并提出一些建议。首先，希望在课程中能够增加一个完全自愿的项目展示交流环节，可以在课程结束之后邀请愿意参与展示的小组进行汇报展示，邀请老师与同学点评并提出建议，这样可以让我们更好认识到自己项目的亮点与不足，也可以在交流中吸取其他团队在分工、安排与设计上的宝贵经验。同时，课程还可以引导学生关注信息安全领域的前沿发展和热点问题，激发学生的创新思维，先由学生想一下有没有可以研究与创新的方向，再给出已经确定好的几个项目，这样可能会更能激发学生的创新思维。

总而言之，我们认为本课程是我们在大学学习中甚至是人生成长中不可或缺的一门课程，我们的专业能力与团队合作能力、沟通能力得到了巨大提升，为我们更好地适应未来工作中的合作环境提供了巨大帮助。

最后，衷心感谢每一位老师与助教在项目开发中提供的宝贵建议、帮助与指导！

附：项目组成员贡献表（由项目组长填写，如不写，各成员之间平分工作量）

成员姓名	是否项目组长	具体承担任务	组长评分（百分制）
徐宇飞	是	控制规则实现、子线程处理、缓存管理、调试	20%
陈冠宇	否	图形界面设计、报文传输、请求接收、调试	20%
林中阳	否	控制规则管理、用户管理、内容过滤、调试	20%
柴继晨	否	报文解析、文件控制管理、日志管理、调试	20%
闫梓涵	否	报文解析、文件控制实现、请求接收、调试	20%