**httpgate详细设计说明书**

**版本 1.0**

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **发布日期** | **修订内容** | **主要修订人** |
| 1.0 | 2012/2/16 | 创建 | 萧石 |
|  |  |  |  |
|  |  |  |  |

**目 录**

[1 引言 3](#_Toc314752059)

[1.1 编写目的 3](#_Toc314752060)

[1.2 背景 3](#_Toc314752061)

[1.3 简介 3](#_Toc314752062)

[2 总体设计 3](#_Toc314752063)

[2.1 系统的总体功能需求 3](#_Toc314752064)

[2.1.1 系统输入数据 3](#_Toc314752065)

[2.1.2 系统输出数据 3](#_Toc314752066)

[2.1.3 系统处理功能 4](#_Toc314752067)

[2.1.4 系统性能要求 4](#_Toc314752068)

[2.2 系统结构 4](#_Toc314752069)

[2.2.1 物理结构 4](#_Toc314752070)

[2.2.2 体系结构 4](#_Toc314752071)

[2.3 外部接口设计 5](#_Toc314752072)

[3 模块设计 5](#_Toc314752073)

[3.1 初始化 6](#_Toc314752074)

[3.2 事件驱动 6](#_Toc314752075)

[3.3 通用内存池模块(genpool.c) 7](#_Toc314752076)

[3.4 防攻击模块(ipfilter.c cookiefilter.c) 10](#_Toc314752077)

[3.5 连接池管理模块(connection.c) 12](#_Toc314752078)

[3.6 健康检查模块(ups.c) 14](#_Toc314752079)

[3.7 ip黑白名单模块(iprange.c) 14](#_Toc314752080)

[3.8 http分析模块(http.c) 15](#_Toc314752081)

[3.9 状态机(state.c) 15](#_Toc314752082)

[3.10 定时器(timer.c) 16](#_Toc314752083)

[3.11 网络通讯模块(sock.c) 17](#_Toc314752084)

[4 配置文件 18](#_Toc314752085)

[5 系统限制 20](#_Toc314752086)

[6 后续改进 20](#_Toc314752087)

# 引言

## 编写目的

本文对httpgate系统进行详细设计，主要包括系统各模块将采用的技术、程序的关键数据结构，程序的接口以及程序的架构设计、程序的模块功能设计。

此文档提供系统的详细设计，即系统的实现技术方案，为系统的编码提供基础，作为今后系统开发和维护的指导性文档，也为测试人员系统测试提供依据。

## 背景

本程序正式名称为httpgate，顾名思义http的大门，所有的http请求都要通过httpgate。httpgate为web server及后端的业务逻辑提供良好的保护功能，最大限度屏蔽恶意访问。同时，针对apache，httpgate能够与客户端保持大量的长连接，提高了系统的整体性能。

## 简介

httpgate是一个http 7层代理软件，专注于http代理及http防攻击，具备如下特点：

1. 采用事件驱动、异步、非阻塞的设计模式。性能优秀，开启http防攻击的情况下，性能接近nginx
2. 对客户端ip和客户端cookie的访问情况进行统计，支持基于客户端ip和客户端cookie的防攻击。
3. 代理层采用读取转发的方式，有效保护后端web server
4. 支持ip白名单，黑名单
5. 支持细致的客户端连接参数配置，可针对不同的应用类型、可能的攻击方式，配置不同的参数，提升攻击的代价。比如客户端连接超时、客户端写入超时、客户端读取超时。
6. 支持后端web server健康检查、故障冗余、rr/iphash负载均衡
7. 支持客户端长连接
8. 支持基于域名,url,cookie分流

# 总体设计

## 系统的总体功能需求

### 系统输入数据

1. 用户访问web服务时，由httpgate抽取用户访问应用服务器时相关信息,包括:IP址地，代理IP地址,用户的cookie，用户的连接时间，读取时间，写入时间，连接频度等
2. 白名单、黑名单列表

### 系统输出数据

1. 关闭客户端的连接
2. 重置客户端的连接
3. 输出客户端被封禁的信息

### 系统处理功能

1. 抽取用户的访问规则，包括ip的访问频度，cookie的访问频度
2. 对抽取的数据进行实时分析/计算，并根据预先定义的规则采取封禁或放行
3. 对抽取的ip数据进行过滤，如果ip在白名单种，则进入cookie数据过滤。如果ip在黑名单中，则直接重置连接。
4. ip白名单、黑名单修改之后，发送SIGUSR1信号可以重载。

### 系统性能要求

1. 效率要求：单机支持20亿以上的日pv，qps达到3w以上
2. 可靠性要求：判断误差率<0.05%
3. 稳定性要求：稳定性99.96%以上
4. 可扩展性：

## 系统结构

### 物理结构



### 体系结构



**说明：**

httpgate系统是一个完整的独立的程序。它介于用户和webserver之间。由httpgate接受并保持与用户的连接，将用户的请求转发给webserver，然后将webserver响应的结果返回给用户。

**交互流程：**

* 与用户的交互
* 接收用户的http请求
* 与用户保持keep alive连接
* 返回请求的结果给用户
* 关闭用户连接
* 重置用户连接
* 与 web server的交互
* 检查可用的web server
* 发送用户的请求给web server
* 接收web server返回的结果

## 外部接口设计



**说明：**

用户将请求发送给httpgate，httpgate分析用户的请求，对用户的请求进行过滤，然后将请求发送给web server，httpgate获取web server返回的结果，将结果返回给用户。

# 模块设计

httpgate的功能由如下各模块互相协助共同完成。



## 初始化

**功能说明：**

httpgate在处理用户请求之前，需要对服务进行初始化。初始化的操作包括命令行参数解析，日志文件初始化，信号初始化，定时器初始化，内存池初始化等。

流程：



## 事件驱动

**功能说明：**

httpgate采用事件驱动、异步、非阻塞的设计模式，事件驱动是整个服务的核心，由事件驱动来调度除定时器以外的所有模块。

**流程：**



## 通用内存池模块(genpool.c)

**功能说明:**

通用内存池模块是httpgate的基础，为提高httpgate整体的性能，减少显式的内存分配，设计了一个通用的内存池genpool，在httpgate里面，用户的连接池，数据buffer，ip连接状态统计，cookie连接状态统计等均用到了该内存池模块。

通用内存池将内存划分为块(chunk)，块再划分为页(page)，每个内存块中含有固定数量的内存页，但是内存页的大小可以根据应用来决定。

通用内存池有3个链表， used\_chunks,free\_chunks,full\_chunks，分别代表使用中的内存块链表，空闲内存块链表，无内存页可用的内存块链表。链表的头部分别为used\_chunks\_head,free\_chunks\_head,full\_chunks\_head。

**通用内存分配表：**



**重要数据结构说明：**

通用内存池句柄，用于描述当前内存池的使用状态。

typedef struct \_genpool\_handler\_t{

uint32\_t free\_chunks, max\_free\_chunks; /\* 当前空闲数据块，最大空闲块数 \*/

uint32\_t prealloc\_chunks; /\* 预分配的数据块数量 \*/

uint32\_t page\_size, pages\_per\_chunk; /\* 页大小，块中的页数 \*/

uint32\_t total\_chunks, max\_total\_chunks; /\* 块的总数， 最大可申请块数 \*/

struct list\_head used\_chunks\_head; /\* 使用中的块链表 \*/

struct list\_head free\_chunks\_head; /\* 空闲的块链表 \*/

struct list\_head full\_chunks\_head; /\* 无空闲页的块链表 \*/

} genpool\_handler\_t;

**接口：**

1. 初始化内存池

genpool\_handler\_t \*genpool\_init(size\_t size, size\_t max);

2. 分配内存页

inline void \*genpool\_alloc\_page(genpool\_handler\_t \*g);



3. 释放内存页

inline int genpool\_release\_page(genpool\_handler\_t \*g, void \*mem);



4. 输出内存池状态

int genpool\_status(genpool\_handler\_t \*g);

## 防攻击模块(ipfilter.c cookiefilter.c)

**功能说明：**

httpgate的防攻击都在该模块里实现，该防攻击模块包含2个部分，来源ip过滤和用户cookie过滤。防攻击模块对用户每一个请求的来源ip，用户cookie进行实时的统计，当一个请求的统计数据达到或超过预订阈值的时候，httpgate会关闭或重置该请求所在的连接。

流程：



**接口：**

1. ip连接状态池初始化

int ip\_pool\_init(size\_t max);

2. ip连接状态池过期数据清理

int ip\_pool\_clean\_timer(int max);

3. ip过滤

inline int ip\_filter(uint32\_t ip);

4. ip连接状态更新

inline int ip\_stat\_refresh(uint32\_t ip);

5. ip封禁策略配置

int ipfilter\_conf\_init(time\_t t1, time\_t t2, uint32\_t s1, uint32\_t s2, time\_t p1, time\_t p2);

6. cookie连接状态池初始化

int cookie\_pool\_init(size\_t max);

7. cookie连接状态池过滤数据清理

int cookie\_pool\_clean\_timer(int max);

8. cookie过滤

inline int cookie\_filter(const char \*cookie, int len);

9. cookie封禁策略配置

int cookiefilter\_conf\_init(time\_t t1, time\_t t2, uint32\_t s1, uint32\_t s2, time\_t p1, time\_t p2);

## 连接池管理模块(connection.c)

**功能说明：**

连接池保存的是当前所有用户的连接，每个连接都会有一个相应的数据结构描述，它是httpgate里最为重要的数据结构。主要包括，客户端的句柄、web server的句柄，连接当前处理状态，数据缓存，后端webserver，epoll回调接口等。

**重要数据结构说明：**

连接描述符用于管理客户端的连接情况，包括连接句柄，连接状态，事件回调等

typedef struct \_connection\_t{

uint32\_t seqno; /\* 该连接的序号 \*/

uint32\_t clientip; /\* 客户端ip \*/

uint32\_t clientport; /\* 客户端端口 \*/

ups\_t upstream; /\* upstream服务器 \*/

ep\_handler\_t cli; /\* 客户端事件回调接口 \*/

ep\_handler\_t ups; /\* ups事件回调接口 \*/

struct list\_head link; /\* 链接指针 \*/

time\_t state\_start\_time; /\* 当前连接状态开始事件，用于超时判断 \*/

buf\_t inbuf; /\* 读buffer \*/

buf\_t outbuf; /\* 写buffer \*/

int state; /\* 连接状态，用于状态机 \*/

int closing; /\* 连接正在关闭的标志 \*/

http\_req\_t request; /\* http请求数据结构 \*/

http\_resp\_t response; /\* http响应数据结构 \*/

int reset\_client; /\* 客户端重置标志，当客户端被封禁时，该标志被置1 \*/

int keepalive; /\* keepalive连接标志 \*/

conn\_stat\_t conn\_stat; /\* 当前连接统计 \*/

} connection\_t;

**接口：**

1. 连接池初始化

int connection\_init(size\_t count);

2. 分配连接数据结构

connection\_t \*connection\_alloc(int clientfd, uint32\_t clientip, uint32\_t clientport);

3. 释放连接数据结构

int connection\_dealloc(connection\_t \*c);

4. 关闭连接，包括客户端和webserver

int connection\_close(connection\_t \*c);

5. 关闭与webserver的连接

int connection\_close\_half(connection\_t \*c);

6. 将连接状态置为关闭

int connection\_set\_closing(connection\_t \*c);

7. 将连接状态置为关闭，并设置reset标示，表示当真正关闭客户端连接的时候直接reset

int connection\_set\_closing\_reset\_client(connection\_t \*c);

8. keepalive超时清理

int connection\_keep\_alive\_clean\_timer(int max);

9. 读客户端超时清理

int connection\_read\_client\_clean\_timer(int max);

10. 连接webserver超时清理

int connection\_connect\_ups\_clean\_timer(int max);

11. 写webserver超时清理

int connection\_write\_ups\_clean\_timer(int max);

12. 连接webserver重试

int connection\_connect\_ups\_retry\_timer(int max);

13. 读webserver写客户端超时清理

int connection\_read\_ups\_write\_client\_clean\_timer(int max);

14. 关闭连接

int connection\_closing\_clean\_timer(int max);

15. 将连接加入read\_ups\_write\_client队列

inline int add\_connection\_read\_ups\_write\_client\_queue(connection\_t \*c);

16. 将连接加入connectups队列

inline int add\_connection\_connectups\_queue(connection\_t \*c);

17. 将连接加入writeups队列

inline int add\_connection\_writeups\_queue(connection\_t \*c);

18. 将连接加入connectups\_retry队列

inline int add\_connection\_connectups\_retry\_queue(connection\_t \*c);

19. 将连接加入readclient队列

inline int add\_connection\_readclient\_queue(connection\_t \*c);

20. 将连接加入keepalive队列

inline int add\_connection\_keepalive\_queue(connection\_t \*c);

## 健康检查及分流(ups.c)

**功能说明：**

健康检查用于对后端webserver的健康状态进行统计，根据一定的分流机制为httpgate提供一个可用的webserver。该模块采用了打分的机制来描述后端webserver的健康状态，当后端webserver的失败分数(fail\_score)达到一定阈值时，webserver将被描述为不可用。

**接口：**

1. 后端webserver列表初始化

int ups\_tree\_init(void);

1. ups初始化

int ups\_init(ups\_t \*u);

1. ups注册。根据host,uri进行注册

int ups\_register(char \*host, char \*uri, char \*srv, char \*port);

1. ups查询。可根据ip,host,uri进行查询，查询方式由balance方式决定。

inline int ups\_search(ups\_t \*u, uint32\_t ip, char \*host, char \*uri, char \*cookie);

1. 查找下一个ups(当前一个ups连接失败)

int ups\_search\_next(ups\_t \*u, uint32\_t ip);

1. 设置ups负载均衡方式

int ups\_set\_balance(char \*host, char \*uri, int balance);

1. 设置最大重试次数

int ups\_set\_maxretry(char \*host, char \*uri, int retry);

1. ups连接成功，当前ups的fail\_score减1

inline int ups\_set\_success(ups\_t \*u);

1. ups连接失败，当前ups的fail\_score加2

inline int ups\_set\_fail(ups\_t \*u);

## ip黑白名单模块(iprange.c)

**功能说明：**

该模块对用户定义的ip黑白名单进行有效的管理，httpgate接收到一个请求之后，会提取请求的来源ip，然后将该ip提交给ip黑白名单进行分析， 如果来源ip在白名单里，则允许该连接进入cookie过滤。如果来源ip在黑名单里，则该连接被重置。如果来源不在黑白名单里，则该连接进入来源ip过滤。

ip列表文件由ip段组成，每个ip段一行，格式为起始ip空格终止ip，例如：

# comment line

10.23.0.1 10.23.0.255

202.102.154.3 202.102.154.3

**接口：**

ip黑名单、ip白名单使用的是同一套接口函数。

1. ip列表初始化

iprange\_t \*iprange\_init(const char \*filename, int max);

2. 检查ip是否在ip列表中

int ipaddr\_in\_range(iprange\_t \*handler, uint32\_t addr);

3. 输出ip列表

int iprange\_dump(iprange\_t \*handler);

4. 释放ip列表

int iprange\_release(iprange\_t \*handler);

5. 释放ip列表，并初始化新的ip列表，当httpgate捕获到SIGUSR1的时候会调用该接口

iprange\_t \*iprange\_reload(iprange\_t \*handler, const char \*filename, int max);

## http分析模块(http.c)

**功能说明：**

http模块对客户端请求及webserver返回的结果进行分析，主要有3个功能

1. 对客户端的请求是否完整进行判断，并提取http头的部分信息，
2. 将请求转发给web server之前，增加x-forward-ip字段，将来源ip加到http头
3. 去掉webserver返回的http头中的”connection: close”，以保持httpgate与客户端之间的keep alive

**接口：**

1. 初始化http request结构体

int http\_req\_init(http\_req\_t \*r);

1. 初始化http response结构体

int http\_resp\_init(http\_resp\_t \*r);

1. 获取http header的元素

int get\_header\_elem(http\_req\_t \*r, int elemseq, char \*buf, int len);

1. 探测http request请求是否接收完毕，并分析http request

int req\_test\_and\_parse(buf\_t \*buf, http\_req\_t \*r);

1. 输出http request header

int http\_req\_header\_dump(http\_req\_t \*r, char \*buf, size\_t len);

1. 在http request header加上客户端ip, X-Forwarded-For: xxx.xxx.xxx.xxx

int http\_req\_append\_clientip(http\_req\_t \*r, uint32\_t ip);

1. 删除http response的Connection: close

int http\_resp\_clean\_conntype(buf\_t \*buf);

## 状态机(state.c)

**功能说明：**

状态机用于对当前连接的状态进行判断，并根据状态机决定下一步的动作。状态机是httpgate最复杂的一个模块。

**接口：**

1. 当前连接的状态机

int state(connection\_t \*c, int \*retcode)



## 定时器(timer.c)

**功能说明：**

httpgate是一个基于事件的、异步的服务器程序，对于一些超时任务及周期性的任务，必须且只能使用定时器进行调度。例如客户端连接超时，keepalive超时，过期数据清理及释放等等。

目前使用定时器进行调度的任务有：

1. keepalive超时清理connection\_keep\_alive\_clean\_timer

2. 读客户端超时清理connection\_read\_client\_clean\_timer

3. 连接upstream超时清理connection\_connect\_ups\_clean\_timer

4. ups失败重连connection\_connect\_ups\_retry\_timer

5. 写ups超时清理connection\_write\_ups\_clean\_timer

6. 写client超时清理connection\_ write\_client\_clean\_timer

7. 关闭连接定时清理connection\_closing\_clean\_timer

8. 过期ip池数据清理ip\_pool\_clean\_timer

9. 过期cookie池数据清理cookie\_pool\_clean\_timer

**接口：**

1. 定时器初始化

int timer\_init(void);

2. 定时器注册

int timer\_register(timer\_func\_t func, unsigned long arg, char \*info, int interval);

3. 定时器执行

int timer(void);

## 网络通讯模块(sock.c)

**功能说明：**

网络通讯模块涵盖了所有的网络操作，包括监听端口，接收客户端请求，连接webserver，以及webserver的读写操作。具体包括如下几个函数。

**接口：**

1. 监听端口

inline int make\_listen\_nonblock(const char \*host, const char \*serv);

2. 连接webserver

inline int connect\_nonblock(const char \*host, const char \*serv, int \*flag);

3. 接受客户端连接

inline int accept\_client(int sockfd, struct sockaddr\_in \*cliaddr, socklen\_t \*len);

4. 读客户端

inline int read\_client\_prepare(connection\_t \*c);

inline int read\_ups\_prepare(connection\_t \*c);

inline int read\_client(connection\_t \*c, int \*retcode);

5. 写客户端

inline int write\_client(connection\_t \*c, int \*retcode);

6. 写webserver

inline int write\_ups(connection\_t \*c, int \*retcode);

7. 读webser

inline int read\_ups(connection\_t \*c, int \*retcode);

8. 其它

inline int setnonblock(int fd);

# 配置文件

global{

# daemon: on=1, off=0

daemon 1

# max connections

max\_connections 1000000

# buffer size (B), suggest bigger than 8192

buffer\_size 16384

# max buffer, equal or bigger than max\_connections

max\_buffer 1000000

# number of work processes

workers 2

# 设置cpu亲和性，开启后性能略有下降...

cpu\_attach 0

# connection keepalive timeout (s)

keepalive\_timeout 20

# max request per keepalive connection

max\_keepalive\_requests 200

# connection read client timeout (s)

read\_client\_timeout 30

# connect ups timeout (s)

connect\_ups\_timeout 2

# write ups timeout (s)

write\_ups\_timeout 10

# connect write client timeout (s)

write\_client\_timeout 60

# listen address

listen\_addr 0.0.0.0

listen\_port 9001

# log file

log\_path ../../logs/httpgate.log

# log level: none, log, debug, info

log\_level log

}

filter{

# ip filter: on=1, off=0

ipfilter 1

# ip filter configuation,

# ip filtered ipfilter\_time1 secs when ip qps exceed --

# ipfilter\_threshold1 in ipfilter\_cycle1 secs

ipfilter\_cycle1 10

ipfilter\_threshold1 150

ipfilter\_time1 10

# ip filtered ipfilter\_time2 secs when ip qps exceed --

# ipfilter\_threshold2 in ipfilter\_cycle2 secs

ipfilter\_cycle2 60

ipfilter\_threshold2 400

ipfilter\_time2 30

# cookie filter: on=1, off=0

cookiefilter 0

# cookie filter configuation, use carefully

# cookie filtered cookiefilter\_time1 secs when cookie qps --

# exceed cookiefilter\_threshold1 in cookiefilter\_cycle1 secs

cookiefilter\_cycle1 10

cookiefilter\_threshold1 60

cookiefilter\_time1 10

# cookie filtered cookiefilter\_time2 secs when cookie qps --

# exceed cookiefilter\_threshold2 in cookiefilter\_cycle2 secs

cookiefilter\_cycle2 60

cookiefilter\_threshold2 200

cookiefilter\_time2 30

# ip whitelist

whitelist ./conf/whitelist

# ip blacklist

blacklist ./conf/blacklist

}

# www.taobao.com branch

upstream{

#complete compare

host www.taobao.com

#uri:/a/ proxy

proxy{

#prefix match

uri /a/

server 172.10.10.11:80

server 172.10.10.12:80

# balance: rr=0, iphash=1, cookie=2, default=1

balance 0

# connect ups max retry

retry 2

}

#uri:/b/ proxy

proxy{

#prefix match

uri /b/

server 172.11.11.100:80

server 172.11.11.101:80

# balance: rr=0, iphash=1, cookie=2, default=1

balance 0

# connect ups max retry

retry 2

}

#uri default proxy

default{

server 172.22.22.200:80

server 172.22.22.201:80

# balance: rr=0, iphash=1, cookie=2, default=1

balance 0

# connect ups max retry

retry 2

}

}

# lz.taobao.com branch

upstream{

#complete compare

host lz.taobao.com:9999

#uri: /top/ proxy

proxy{

#prefix match

uri /top/

server 172.24.25.26:9999

server 172.24.25.27:9999

# balance: rr=0, iphash=1, cookie=2, default=1

balance 1

# connect ups max retry

retry 2

}

#uri default proxy

default{

server 172.24.25.120:9999

server 172.24.25.121:9999

# balance: rr=0, iphash=1, cookie=2, default=1

balance 0

# connect ups max retry

retry 2

}

}

#default branch

default{

server 127.0.0.1:80

# balance: rr=0, iphash=1, cookie=2, default=1

balance 0

# connect ups max retry

retry 2

}

# 系统限制

目前httpgate可能有如下问题有待改进。

1. 防攻击

Httpgate的防攻击支持基于ip和cookie的实时过滤，但是方式比较粗暴，直接重置客户端的连接，需要提供更细致的支持。另外，httpgate也不支持基于url的过滤。

1. 监控及管理

现在对httpgate运行状态进行监控难度比较大，没有一个统一的接口。

没有提供管理接口，无法对程序做灵活的配置和管理。

# 后续改进

基于httpgate目前存在的一些问题，下一个版本将着重改进如下功能：

1. 增加基于url的防攻击，细化防攻击的方式
2. 提供实时监控接口