第三部分

HTTP 模块

扫码试看/订阅

《Nginx 核心知识100讲》

配置块的嵌套

```
main
http {
        upstream { … }
        split_clients {…}
        map {…}
        geo {…}
        server {
                if () {…}
                location {
                        limit_except {…}
                }
                location {
                        location {
                        }
                }
        }
        server {
        }
}
```

指令的 Context

示例

Syntax: **log_format** *name* [escape=default|json|none] *string* ...;

Default: log_format combined "...";

Context: http

Syntax: **access_log** *path* [*format* [buffer=*size*] [gzip[=*level*]] [flush=*time*] [if=*condition*]];
access_log off;

Default: access_log logs/access.log combined;

Context: http, server, location, if in location, limit_except

指令的合并

值指令：存储配置项的值

- 可以合并
- 示例
  - root
  - access_log
  - gzip

动作类指令：指定行为

- 不可以合并
- 示例
  - rewrite
  - proxy_pass
- 生效阶段
  - server_rewrite 阶段
  - rewrite 阶段
  - content 阶段

存储值的指令继承规则：向上覆盖

子配置不存在时，直接使用父配置块

子配置存在时，直接覆盖父配置块

```
server {
        listen       8080;
        root /home/geek/nginx/html;
        access_log  logs/geek.access.log  main;
        location /test {
                root /home/geek/nginx/test;
                access_log  logs/access.test.log main;
        }
        location /dlib {
                alias dlib/;
        }

        location / {
         }
}
```

HTTP 模块合并配置的实现

指令在哪个块下生效？

指令允许出现在哪些块下？

在 server 块内生效，从 http 向 server 合并指令：

- char *(*merge_srv_conf)(ngx_conf_t *cf, void *prev, void *conf);

配置缓存在内存

- char *(*merge_loc_conf)(ngx_conf_t *cf, void *prev, void *conf);

Listen 指令

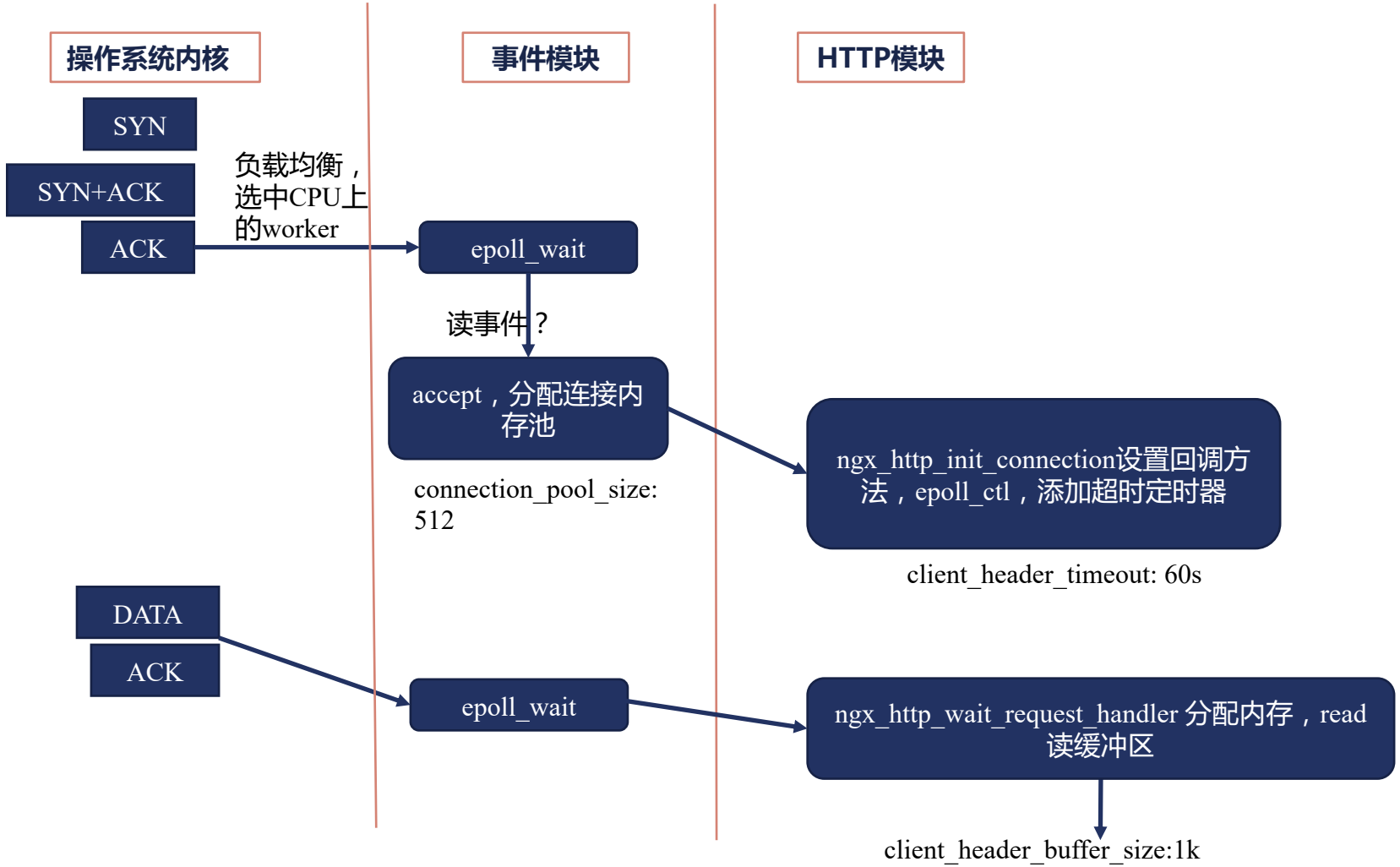
极客时间

Syntax:	**listen** *address*[:*port*] [default_server] [ssl] [http2 | spdy] [proxy_protocol] [setfib=*number*] [fastopen=*number*] [backlog=*number*] [rcvbuf=*size*] [sndbuf=*size*] [accept_filter=*filter*] [deferred] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[*keepidle*]:[*keepintvl*]:[*keepcnt*]]; **listen** *port* [default_server] [ssl] [http2 | spdy] [proxy_protocol] [setfib=*number*] [fastopen=*number*] [backlog=*number*] [rcvbuf=*size*] [sndbuf=*size*] [accept_filter=*filter*] [deferred] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[*keepidle*]:[*keepintvl*]:[*keepcnt*]]; **listen** unix:*path* [default_server] [ssl] [http2 | spdy] [proxy_protocol] [backlog=*number*] [rcvbuf=*size*] [sndbuf=*size*] [accept_filter=*filter*] [deferred] [bind] [so_keepalive=on|off|[*keepidle*]:[*keepintvl*]:[*keepcnt*]];
Default:	listen *:80 | *:8000;
Context:	server

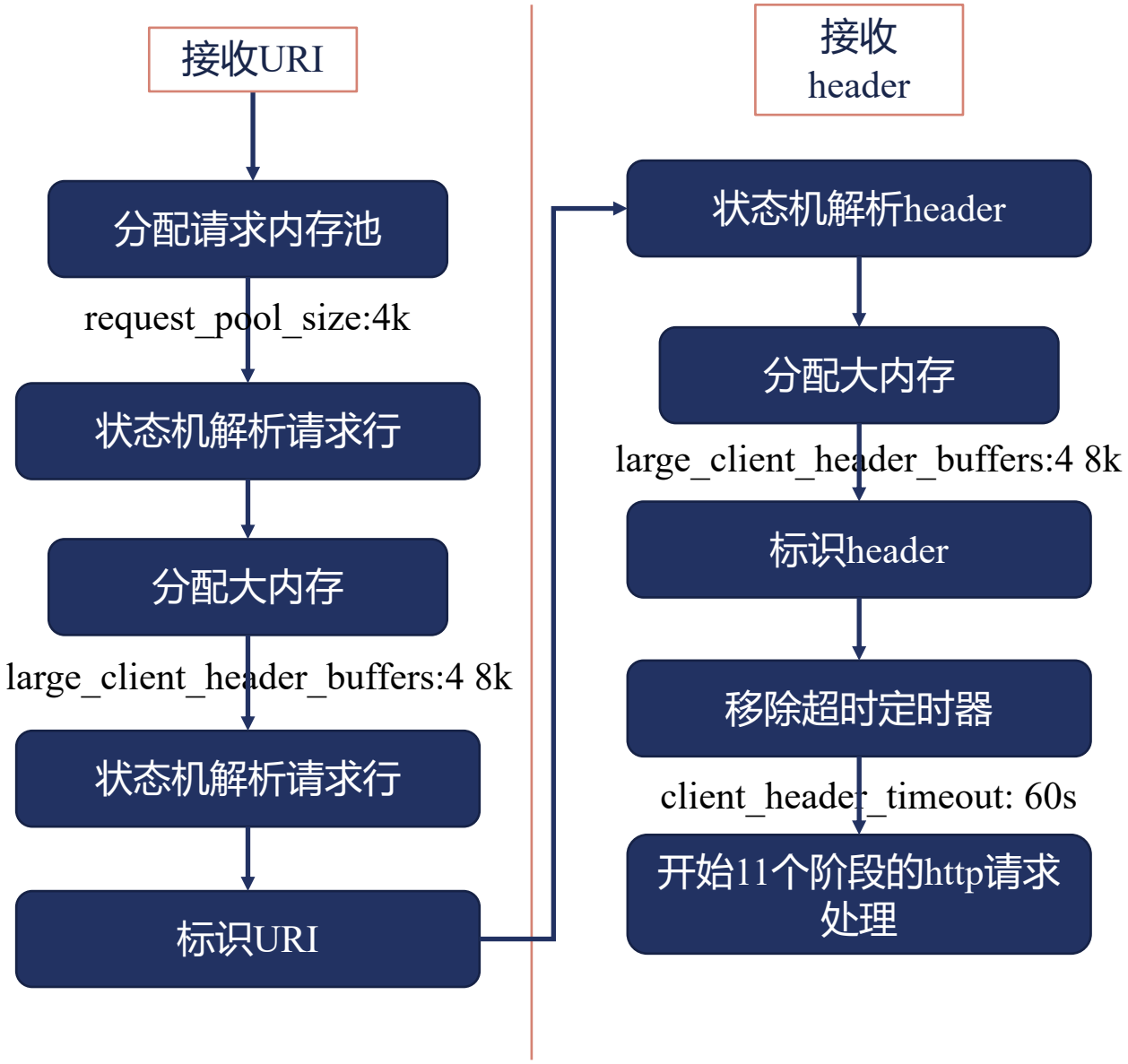
示例

```
listen unix:/var/run/nginx.sock;
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000 bind;
listen [::]:8000 ipv6only=on;
listen [::1];
```

接收请求事件模块


操作系统内核
事件模块
HTTP模块
SYN
SYN+ACK
ACK
负载均衡，选中CPU上的worker
epoll_wait
读事件？
accept，分配连接内存池
connection_pool_size: 512
ngx_http_init_connection设置回调方法，epoll_ctl，添加超时定时器
client_header_timeout: 60s
DATA
ACK
epoll_wait
ngx_http_wait_request_handler 分配内存，read读缓冲区
client_header_buffer_size:1k

接收请求 HTTP 模块


接收URI
分配请求内存池
request_pool_size:4k
状态机解析请求行
分配大内存
large_client_header_buffers:4 8k
状态机解析请求行
标识URI
接收
header
状态机解析header
分配大内存
large_client_header_buffers:4 8k
标识header
移除超时定时器
client_header_timeout: 60s
开始11个阶段的http请求处理

过大的请求头部

Syntax: **client_header_buffer_size** *size*;

Default: client_header_buffer_size 1k;

Context: http, server

Syntax: **large_client_header_buffers** *number size*;

Default: large_client_header_buffers 4 8k;

Context: http, server

正则表达式（一）

元字符

代码	说明
.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线或汉字
\s	匹配任意的空白符
\d	匹配数字
\b	匹配单词的开始或结束
^	匹配字符串的开始
$	匹配字符串的结束

重复

代码	说明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复 n 次
{n,}	重复 n 次或更多次
{n,m}	重复 n 到 m 次

正则表达式（二）

\ 转义符号：取消元字符的特殊含义

() 分组与取值：

示例：
原始url：/admin/website/article/35/change/uploads/party/5.jpg
转换后的url：/static/uploads/party/5.jpg

匹配原始url的正则表达式：
/^\/admin\/website\/article\/(\d+)\/change\/uploads\/(\w+)\/(\w+)\.(png|jpg|gif|jpeg|bmp)$/

rewrite^/admin/website/solution/(\d+)/change/uploads/(.*)\.(png|jpg|gif|jpeg|bmp)$ /static/uploads/$2/$3.$4 last;

server_name 指令

01

指令后可以跟多个域名，第1个是主域名

Syntax　**server_name_in_redirect** on | off;

Default　server_name_in_redirect off;

Context　http, server, location

02

***泛域名：仅支持在最前或者最后**

例如：server_name *.taohui.tech

03

正则表达式：加~前缀

例如：server_name www.taohui.tech ~^www\d+\.taohui\.tech$;

server_name 指令

用正则表达式创建变量：用小括号()
例如:

```
server {
    server_name ~^(www\.)?(.+)$;
    location / { root /sites/$2; }
}
```

```
server {
    server_name ~^(www\.)?(?<domain>.+)$;
    location / { root /sites/$domain; }
}
```

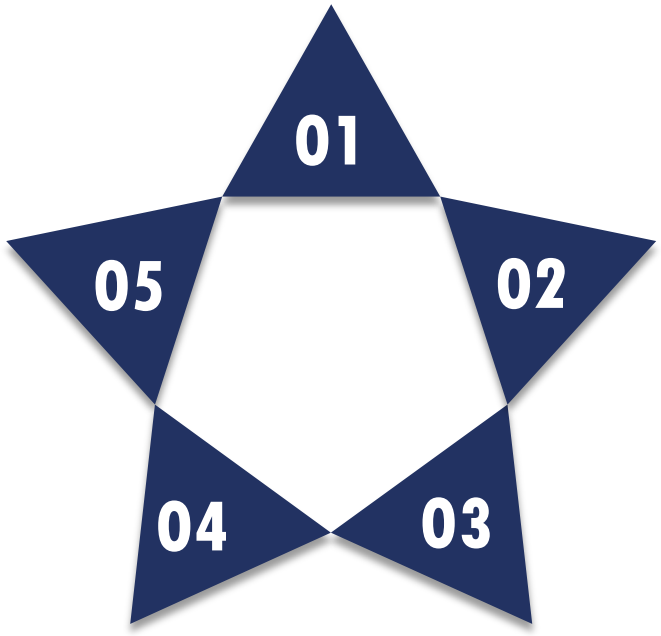
04

05

其他

- .taohui.tech可以匹配 taohui.tech *.taohui.tech
- _匹配所有
- “” 匹配没有传递Host头部

Server 匹配顺序



01 精确匹配

02 *在前的泛域名

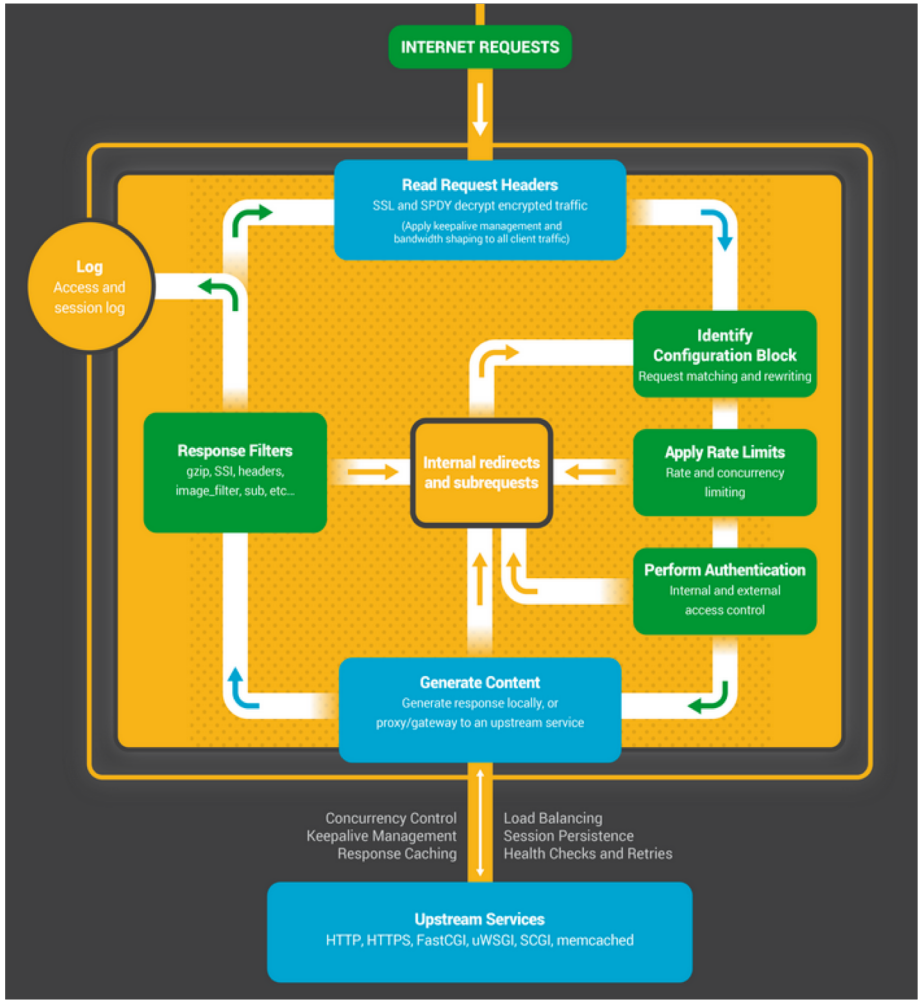
03 *在后的泛域名

04 按文件中的顺序匹配正则表达式域名

05 default server

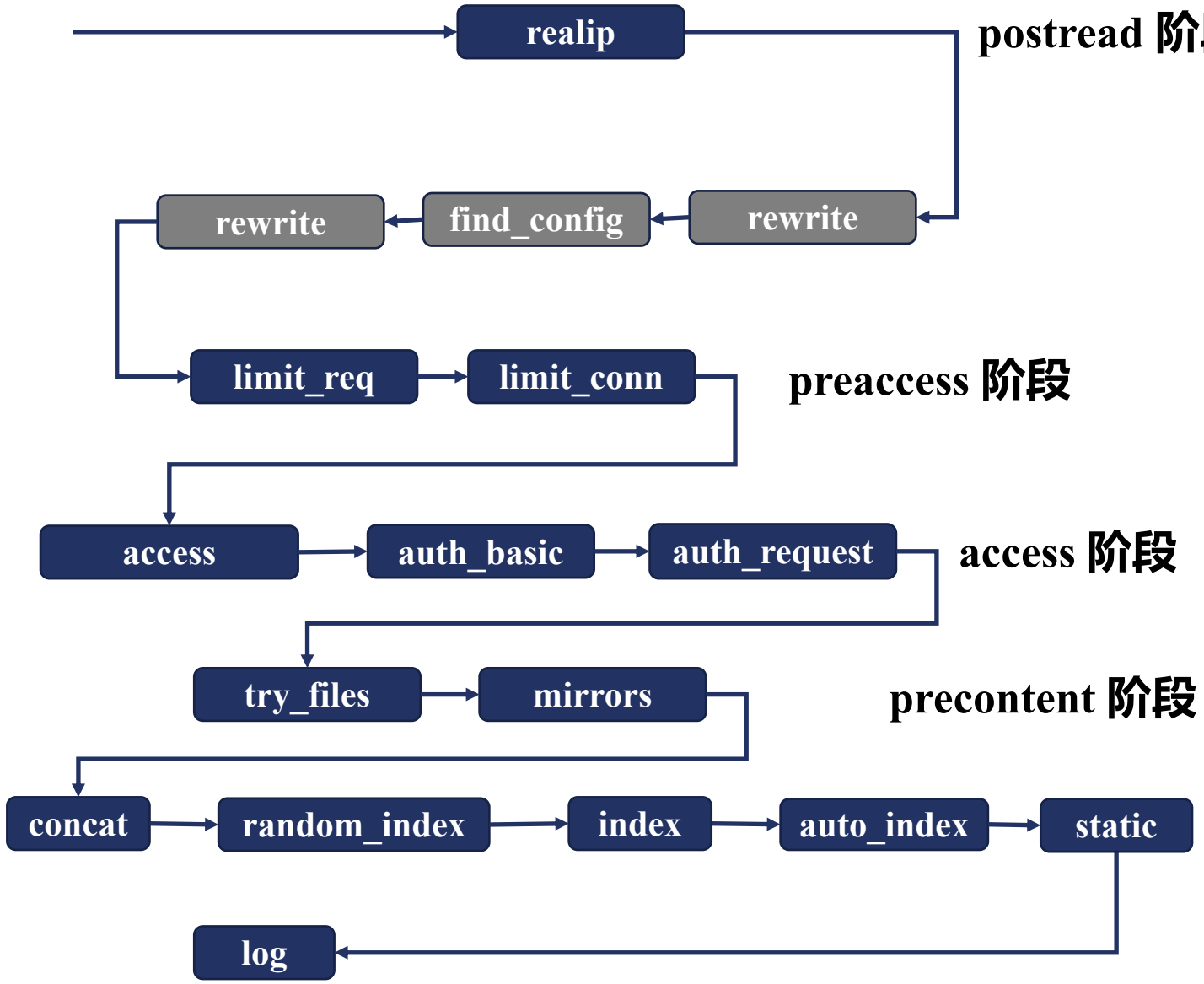
- 第 1 个
- listen 指定 default

HTTP 请求处理时的 11 个阶段



POST_READ	realip
SERVER_REWRITE	rewrite
FIND_CONFIG	
REWRITE	rewrite
POST_REWRITE	
PREACCESS	limt_conn, limit_req
ACCESS	auth_basic, access, auth_request
POST_ACCESS	
PRECONTENT	try_files
CONTENT	index, autoindex, concat
LOG	access_log

11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

问题：如何拿到真实的用户 IP 地址？

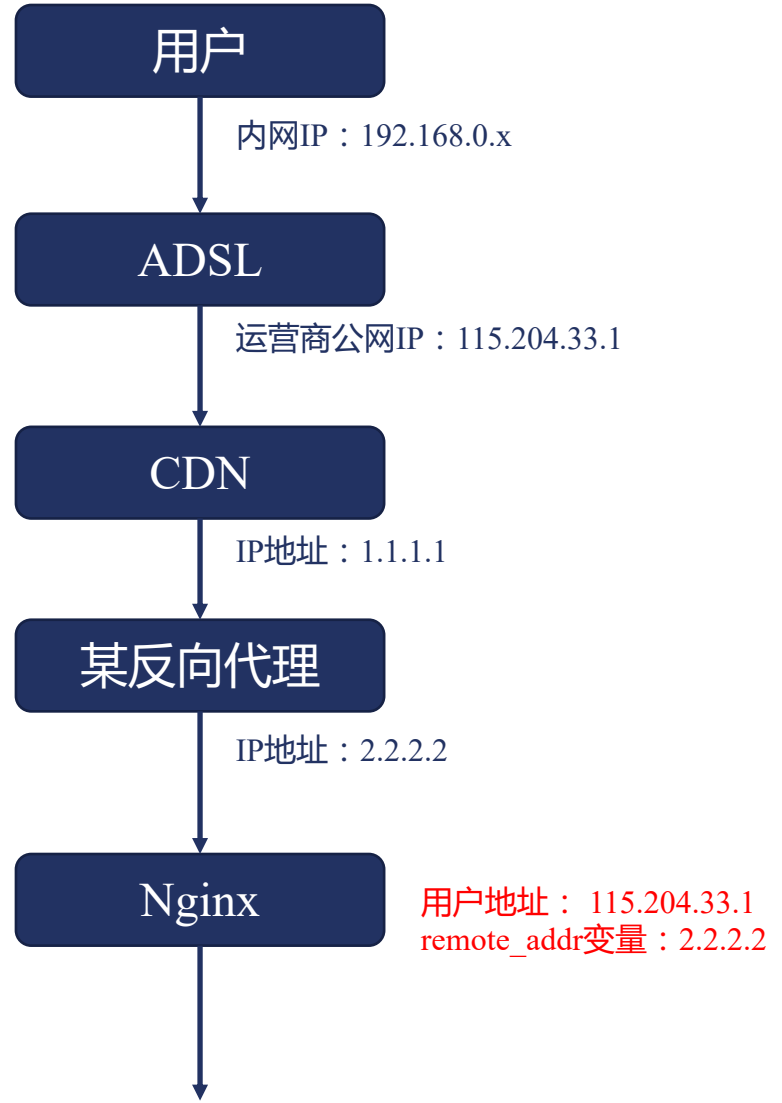
01 TCP 连接四元组（ src ip,src port, dst ip,dst port ）

02 HTTP 头部 X-Forwarded-For 用于传递 IP

03 HTTP 头部 X-Real-IP 用于传递用户 IP

04 网络中存在许多反向代理

前提



X-Forwarded-For： 115.204.33.1
X-Real-IP： 115.204.33.1

X-Forwarded-For： 115.204.33.1 , 1.1.1.1
X-Real-IP： 115.204.33.1

拿到真实用户 IP 后如何使用？

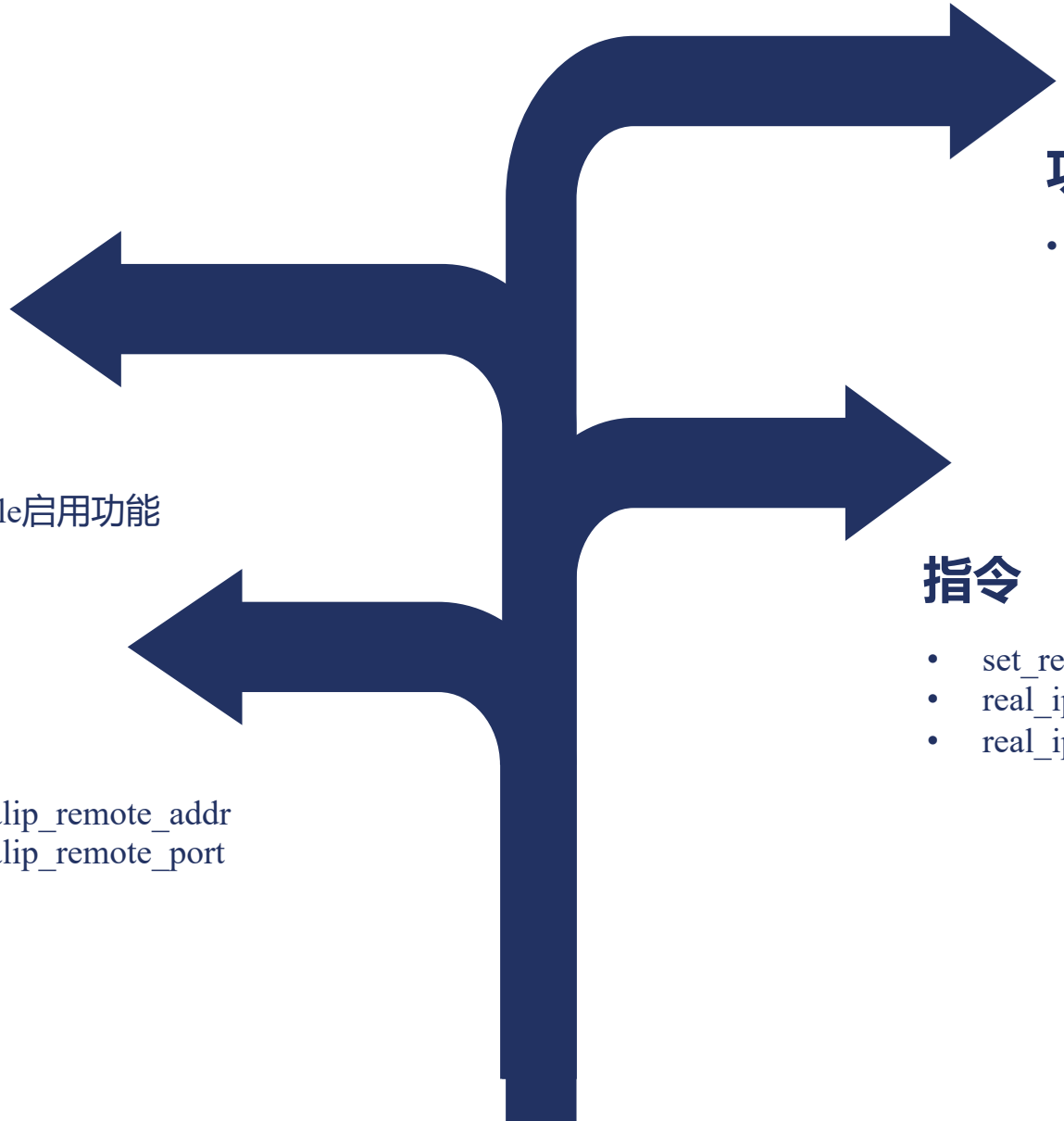
基于变量 ！！！

如 binary_remote_addr、 remote_addr 这样的变量，

其值就为真实的 IP！这样做连接限制（limit_conn模块）

才有意义！

realip 模块



功能

- 修改客户端地址

默认不会编译进 Nginx

- 通过--with-http_realip_module启用功能

指令

- set_real_ip_from
- real_ip_header
- real_ip_recursive

变量

- realip_remote_addr
- realip_remote_port

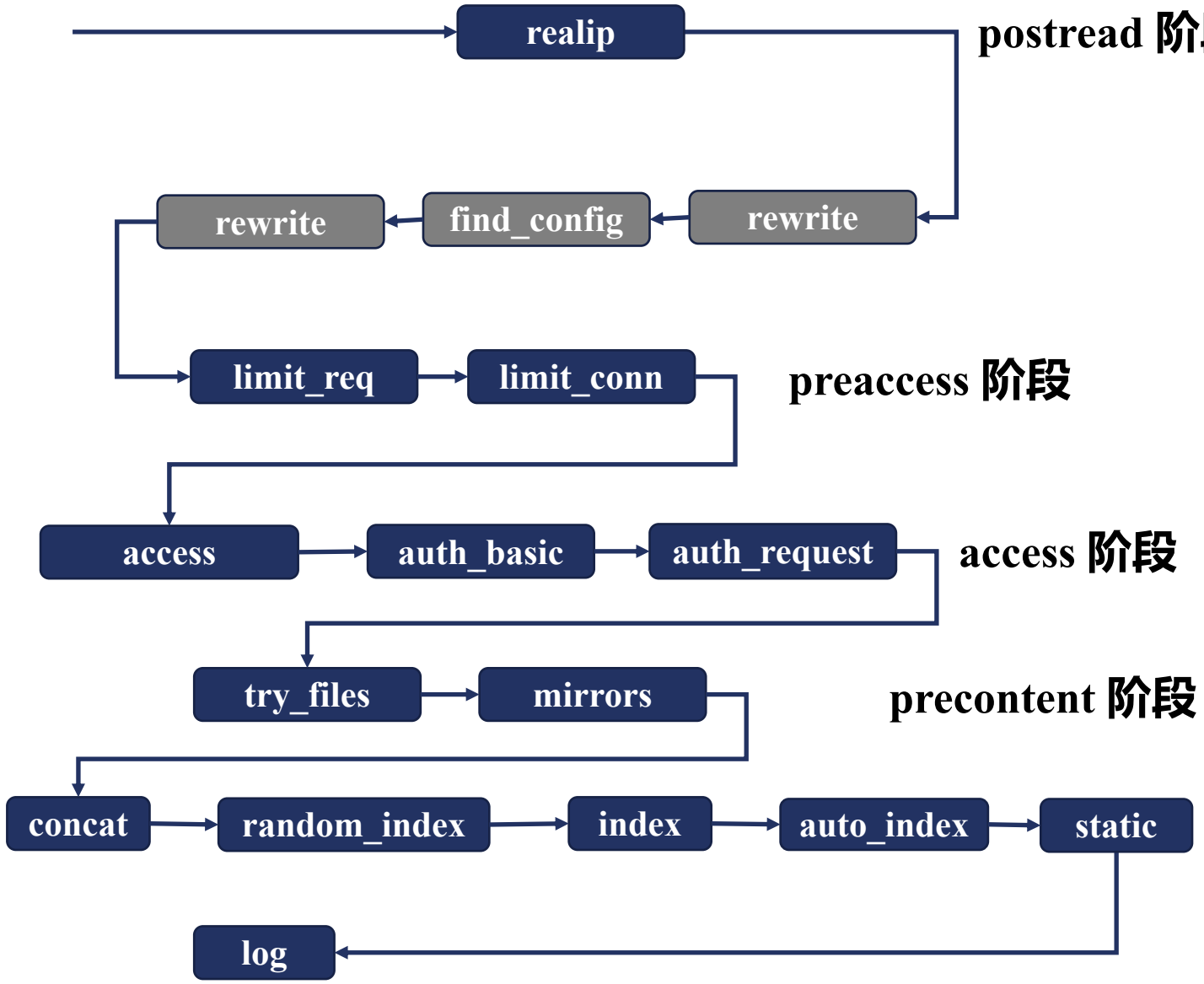
realip 模块的指令

Syntax: **set_real_ip_from** *address* | *CIDR* | unix:;
Default: —
Context: http, server, location

Syntax: **real_ip_header** *field* | X-Real-IP | X-Forwarded-For | proxy_protocol;
Default: real_ip_header X-Real-IP;
Context: http, server, location

Syntax: **real_ip_recursive** on | off;
Default: real_ip_recursive off;
Context: http, server, location

11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

Rewrite 模块：return 指令

Syntax:	**return** *code* [*text*]; **return** *code URL*; **return** *URL*;
Default:	—
Context:	server, location, if

返回状态码

- Nginx自定义
  - 444：关闭连接
- HTTP 1.0标准
  - 301：http1.0永久重定向
  - 302：临时重定向，禁止被缓存
- HTTP 1.1标准
  - 303：临时重定向，允许改变方法，禁止被缓存
  - 307：临时重定向，不允许改变方法，禁止被缓存
  - 308：永久重定向，不允许改变方法

return 示例

```
server {
        server_name return.taohui.tech;
        listen 8080;


        root html/;
        error_page 404 /403.html;
        return 403;
        location /{
                return 404 "find nothing!";
        }
}
```

问题

server与location块下的return指令关系？

return与error_page指令的关系？

rewrite 模块：return 指令与 error_page

Syntax: **error_page** *code* ... [=[*response*]] *uri*;
Default: —
Context: http, server, location, if in location

例子：

1. error_page 404 /404.html;
2. error_page 500 502 503 504 /50x.html;
3. error_page 404 =200 /empty.gif;
4. error_page 404 = /404.php;
5. location / {
   error_page 404 = @fallback;
   }
   location @fallback {
   proxy_pass http://backend;
   }
6. error_page 403 http://example.com/forbidden.html;
7. error_page 404 =301 http://example.com/notfound.html;

rewrite 模块：rewrite 指令

极客时间

Syntax:	**rewrite** regex replacement [flag];
Default:	—
Context:	server, location, if

功能

- **将regex指定的url替换成replacement这个新的url**

  --可以使用正则表达式及变量提取

- **当replacement以http://或者https://或者$schema开头，则直接返回302重定向**
- **替换后的url根据flag指定的方式进行处理**

  --last：用replacement这个URI进行新的location匹配

  --break：break指令停止当前脚本指令的执行，等价于独立的break指令

  --redirect：返回302重定向

  --permanent：返回301重定向

rewrite 指令示例（一）

极客时间

配置指令：

```
root html/;
        location /first {
                rewrite /first(.*) /second$1 last;
                 return 200    'first!';
        }

        location /second {
                rewrite /second(.*) /third$1 break;
                return 200 'second!';
        }

        location /third {
                return 200 'third!';
        }
```

目录结构：

```
html/first/
└── 1.txt
html/second/
└── 2.txt
html/third/
└── 3.txt
```

问题

1. return指令与rewrite指令的顺序关系？
2. 访问/first/3.txt， /second/3.txt，/third/3.txt分别返回的是什么？
3. 如果不携带flag会怎么样？

rewrite 指令示例（二）

```
location /redirect1 {
        rewrite /redirect1(.*) $1 permanent;
}

location /redirect2 {
        rewrite /redirect2(.*) $1 redirect;
}

location /redirect3 {
        rewrite /redirect3(.*) http://rewrite.taohui.tech$1;
}

location /redirect4 {
        rewrite /redirect4(.*) http://rewrite.taohui.tech$1 permanent;
}
```

问题

访问/redirect1/index.html，

返回的是什么？

以及2、3、4分别又是什么？

rewrite 行为记入 error 日志

Syntax: **rewrite_log** on | off;

Default: rewrite_log off;

Context: http, server, location, if

rewrite 模块的 if 指令

Syntax:	**if** (*condition*) { ... }
Default:	—
Context:	server, location

规则

条件 condition 为真，则执行大括号内的指令；遵循值指令的继承规则

if 指令的条件表达式

规则

01 检查变量为空或者值是否为0，直接使用

02 将变量与字符串做匹配，使用=或者 !=

03 将变量与正则表达式做匹配

- 大小写敏感，~或者!~
- 大小写不敏感，~*或者!~*

04 检查文件是否存在，使用-f 或者 !-f

05 检查目录是否存在，使用-d或者!-d

06 检查文件、目录、软链接是否存在，使用-e或者!-e

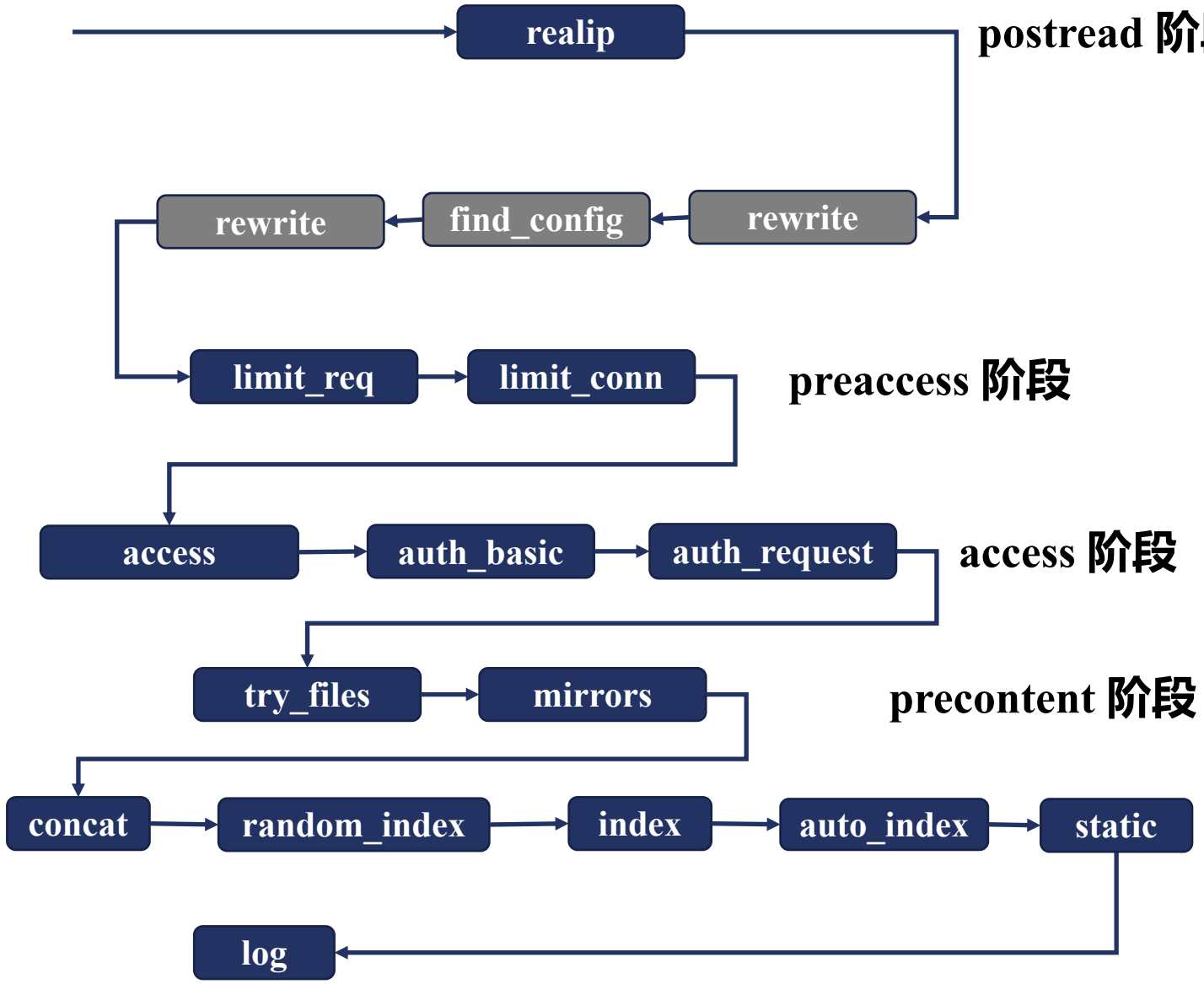
07 检查是否为可执行文件，使用-x或者!-x

if 指令的条件表达式

示例：

```
if ($http_user_agent ~ MSIE) {
        rewrite ^(.*)$ /msie/$1 break;
}
if ($http_cookie ~* "id=([^;]+)(?:;|$)") {
        set $id $1;
}
if ($request_method = POST) {
        return 405;
}
if ($slow) {
        limit_rate 10k;
}
if ($invalid_referer) {
        return 403;
}
```

11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

location 指令

Syntax: **location** [= | ~ | ~* | ^~] *uri* { ... }
location *@name* { ... }

Default: —

Context: server, location

Syntax: **merge_slashes** on | off;

Default: merge_slashes on;

Context: http, server

location 匹配规则：仅匹配 URI，忽略参数

合并连续的/符号

- merge_slashes on

前缀字符串

- 常规
- =：精确匹配
- ^~：匹配上后则不再进行正则表达式匹配

正则表达式

- ~：大小写敏感的正则匹配
- ~*：忽略大小写的正则匹配

用于内部跳转的命名location

- @

问题

以下 URL 会返回什么内容?

/Test1

/Test1/

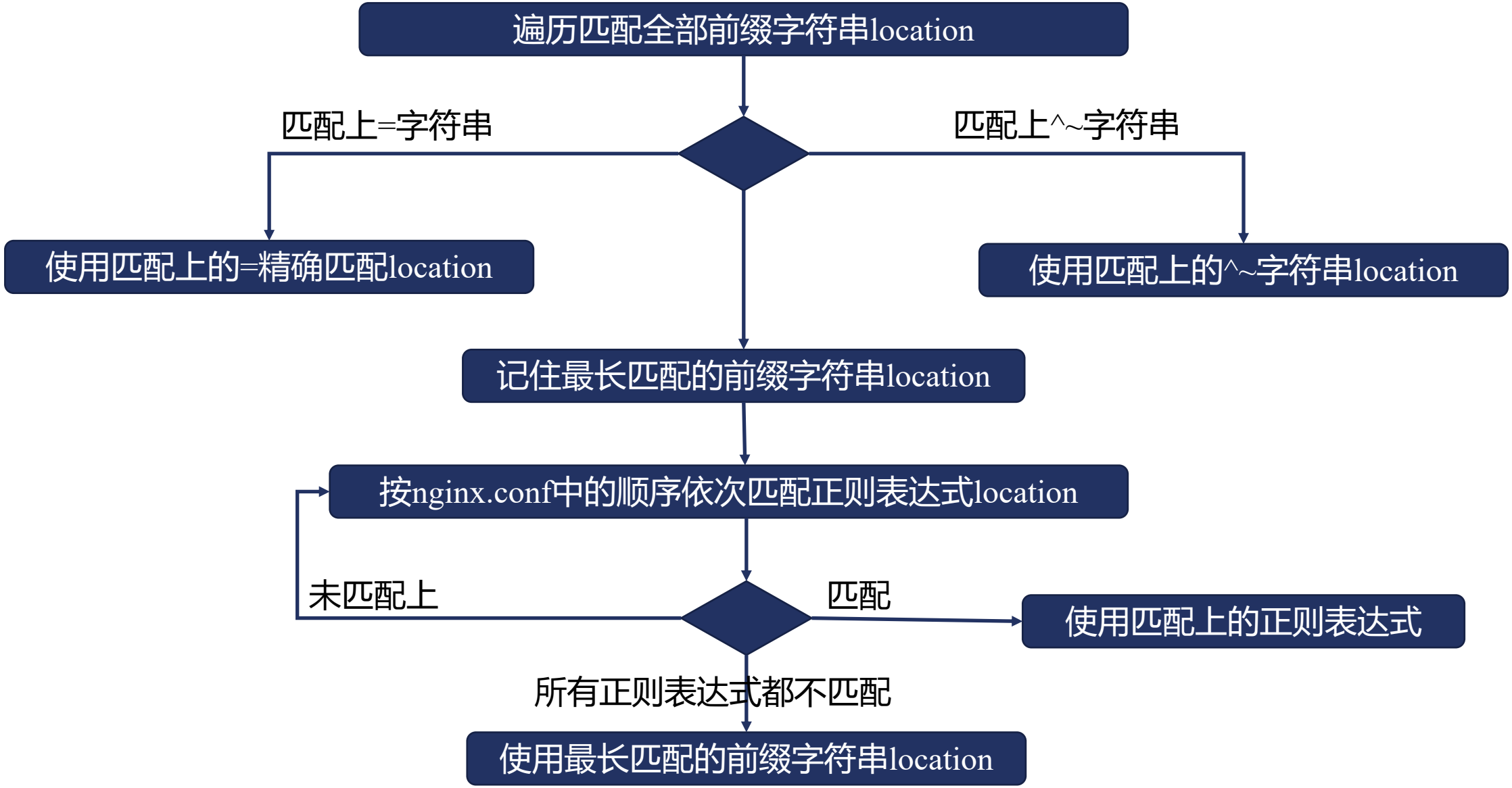
/Test1/Test2

/Test1/Test2/

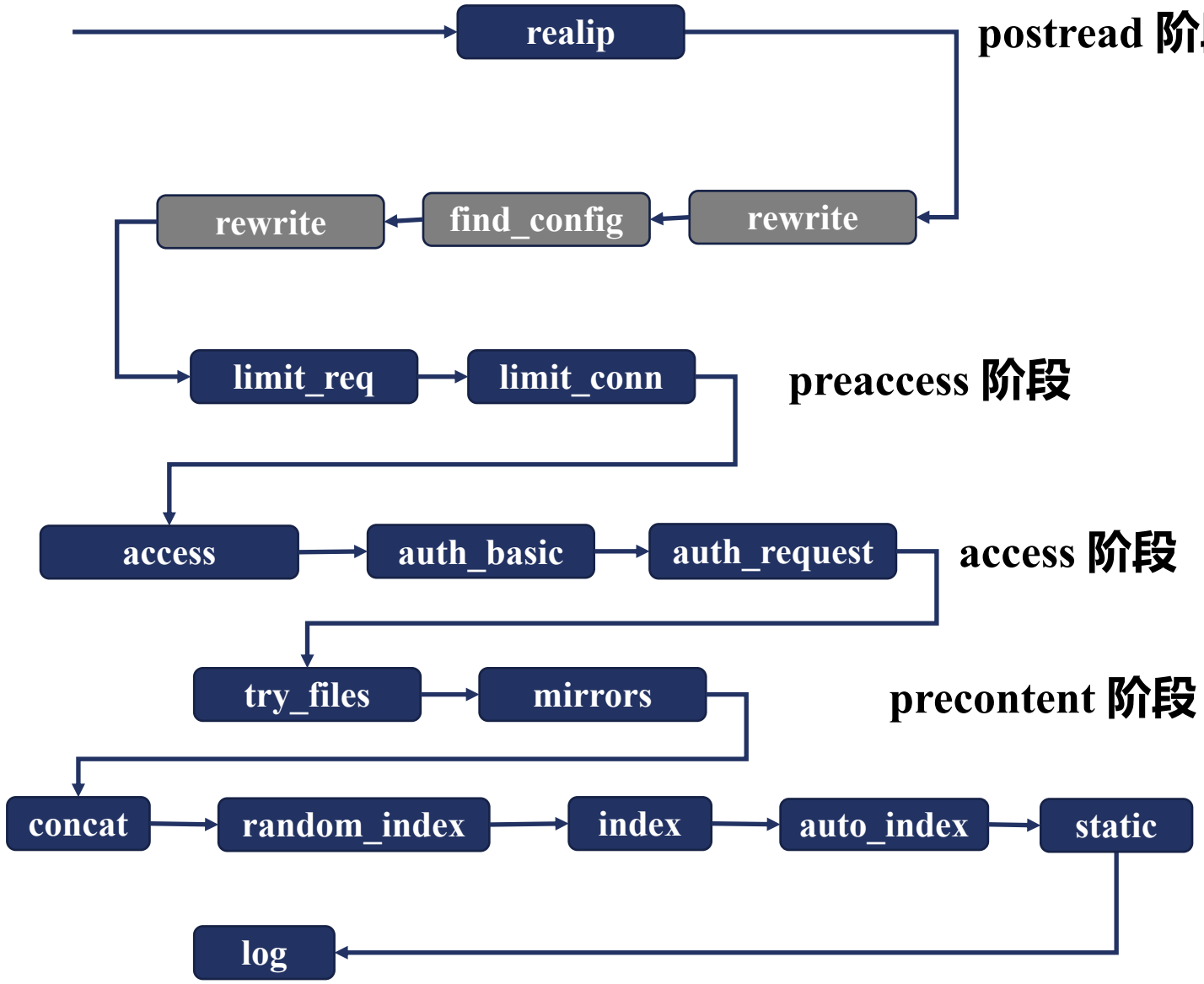
/test1/Test2

```
location ~ /Test1/$ {
        return 200 'first regular expressions match!';
}

location ~* /Test1/(\w+)$ {
        return 200 'longest regular expressions match!';
}

location ^~ /Test1/ {
        return 200 'stop regular expressions match!';
}

location /Test1/Test2 {
        return 200 'longest prefix string match!';
}

location /Test1 {
        return 200 'prefix string match!';
}


location = /Test1 {
        return 200 'exact match!';
}
```

location 匹配顺序



11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

问题：如何限制每个客户端的并发连接数？

ngx_http_limit_conn_module 模块

- **生效阶段：NGX_HTTP_PREACCESS_PHASE 阶段**
- **模块：http_limit_conn_module**
- **默认编译进nginx，通过--without-http_limit_conn_module禁用**
- **生效范围：**
  - 全部worker进程（基于共享内存）
  - 进入preaccess阶段前不生效
  - 限制的有效性取决于key的设计：依赖postread阶段的realip模块取到真实ip

limit_conn 指令

步骤

01

定义共享内存（包括大小），以及key关键字

Syntax: **limit_conn_zone** *key* zone=*name*:*size*;

Default: —

Context: http

02

限制并发连接数

Syntax: **limit_conn** *zone number*;

Default: —

Context: http, server, location

limit_conn 指令

限制发生时的日志级别

Syntax: **limit_conn_log_level** info | notice | warn | error;

Default: limit_conn_log_level error;

Context: http, server, location

限制发生时向客户端返回的错误码

Syntax: **limit_conn_status** *code*;

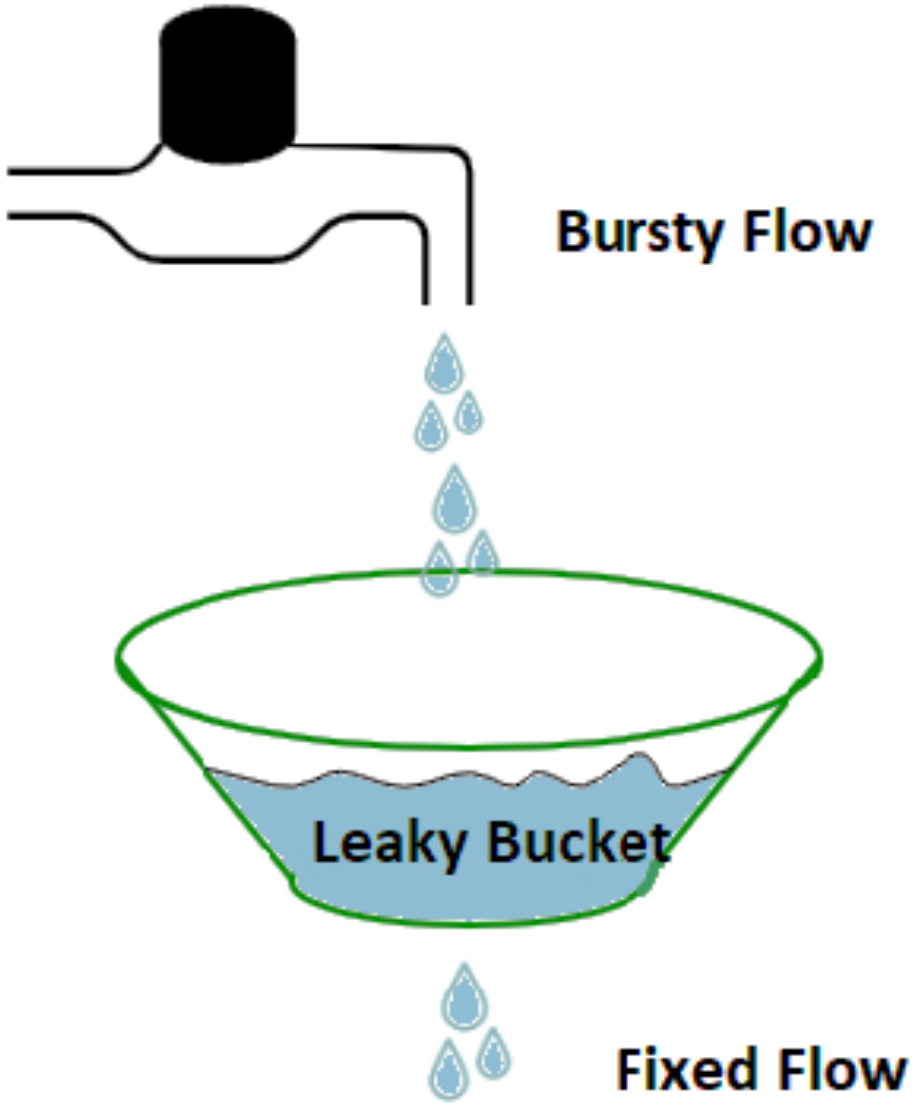
Default: limit_conn_status 503;

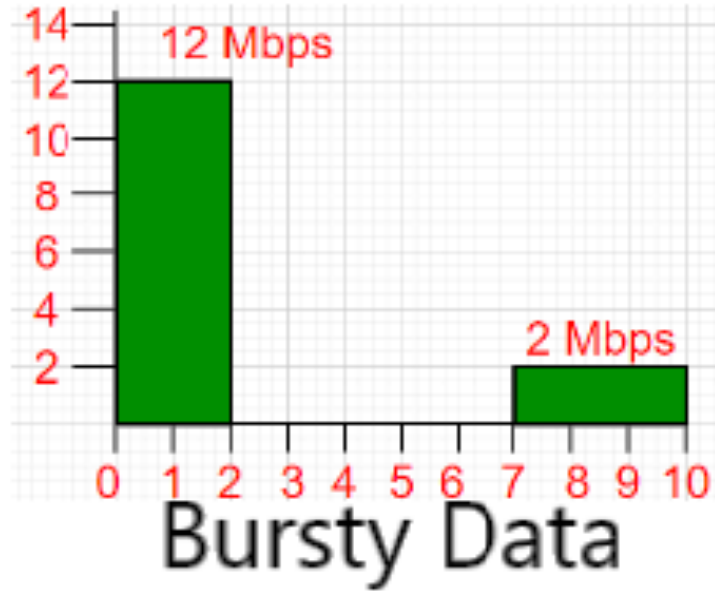
Context: http, server, location

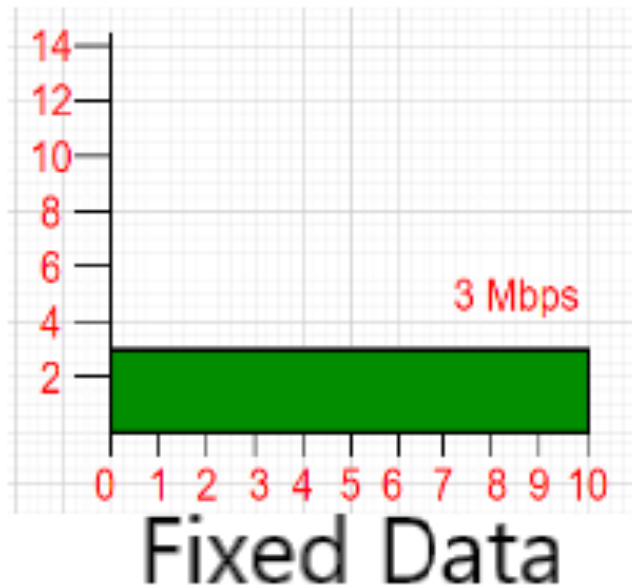
问题：如何限制每个客户端的每秒处理请求数？

ngx_http_limit_req_module 模块

- **生效阶段：NGX_HTTP_PREACCESS_PHASE 阶段**
- **模块：http_limit_req_module**
- **默认编译进nginx，通过--without-http_limit_req_module禁用功能**
- **生效算法：leaky bucket 算法**
- **生效范围：**
  - 全部worker进程（基于共享内存）
  - 进入preaccess阶段前不生效
  - 限制的有效性取决于key的设计：依赖postread阶段的realip模块取到真实ip


Bursty Flow
Leaky Bucket
Fixed Flow


14
12
10
8
6
4
2
12 Mbps
2 Mbps
0 1 2 3 4 5 6 7 8 9 10
Bursty Data


14
12
10
8
6
4
2
3 Mbps
0 1 2 3 4 5 6 7 8 9 10
Fixed Data

limit_req 指令

步骤

定义共享内存（包括大小），以及 key 关键字和限制速率

Syntax: **limit_req_zone** *key* zone=*name*:*size* rate=*rate* ;

Default: —

Context: http

- rate单位为r/s或者r/m

限制并发连接数

Syntax: **limit_req** zone=*name* [burst=*number*] [nodelay];

Default: —

Context: http, server, location

- burst默认为0
- nodelay，对burst中的请求不再采用延时处理的做法，而是立刻处理

limit_req 指令

限制发生时的日志级别

Syntax: **limit_req_log_level** info | notice | warn | error;

Default: limit_req_log_level error;

Context: http, server, location

限制发生时向客户端返回的错误码

Syntax: **limit_red_status** *code*;

Default: limit_red_status 503;

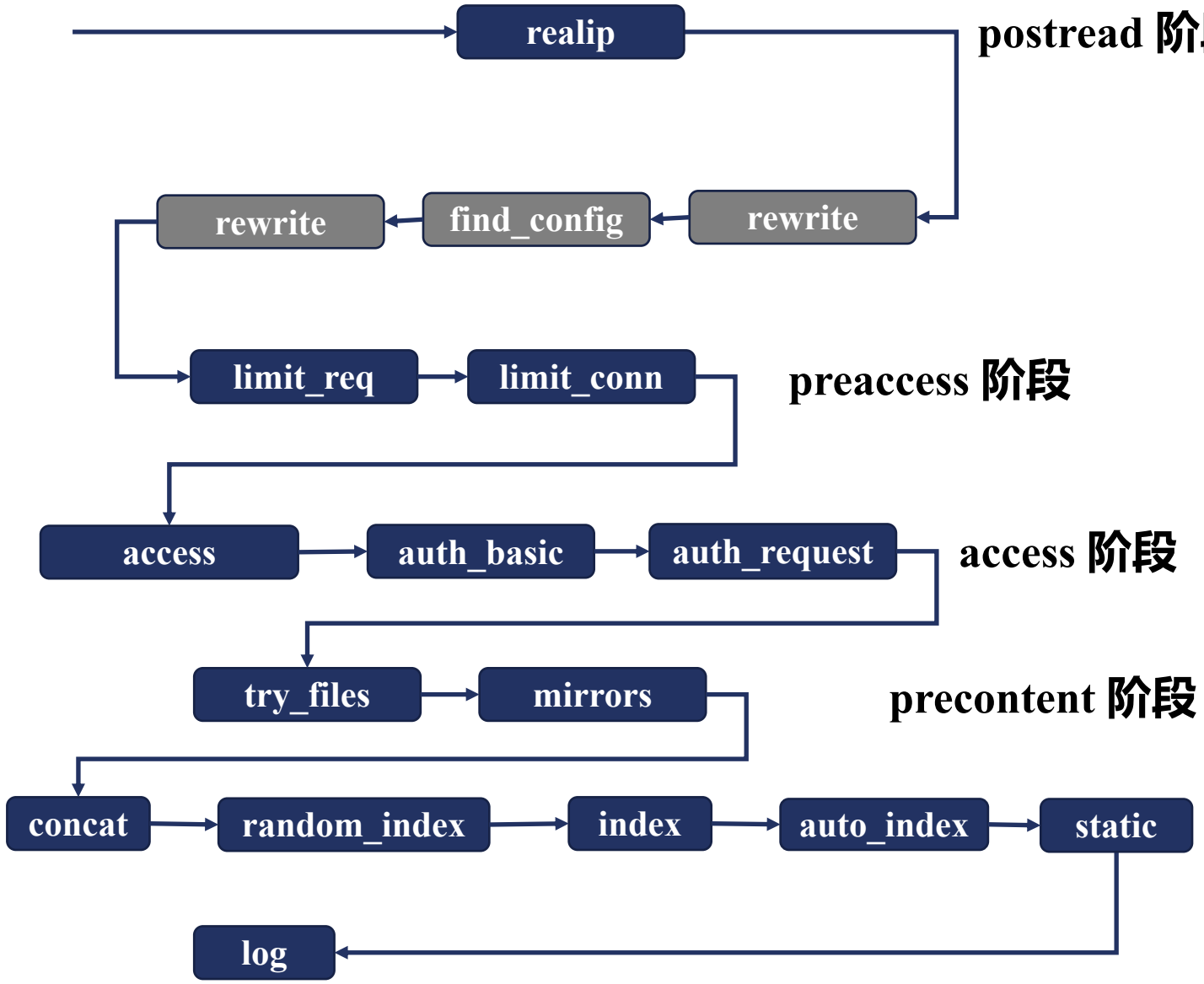
Context: http, server, location

问题：

limit_req 与 limit_conn 配置同时生效时，哪个有效？

nodelay 添加与否，有什么不同？

11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

如何限制某些 IP 地址的访问权限？

Syntax: **allow** *address* | *CIDR* | unix: | all;

Default: —

Context: http, server, location, limit_except

Syntax: **deny** *address* | *CIDR* | unix: | all;

Default: —

Context: http, server, location, limit_except

示例：

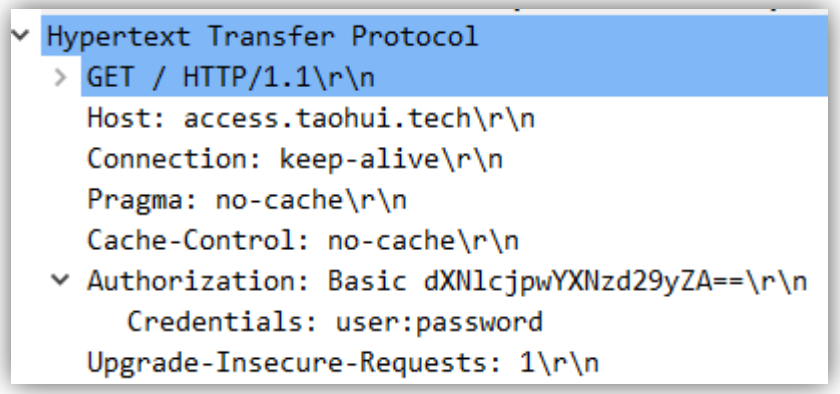
```
location / {
        deny 192.168.1.1;
        allow 192.168.1.0/24;
        allow 10.1.1.0/16;
        allow 2001:0db8::/32;
        deny all;
}
```

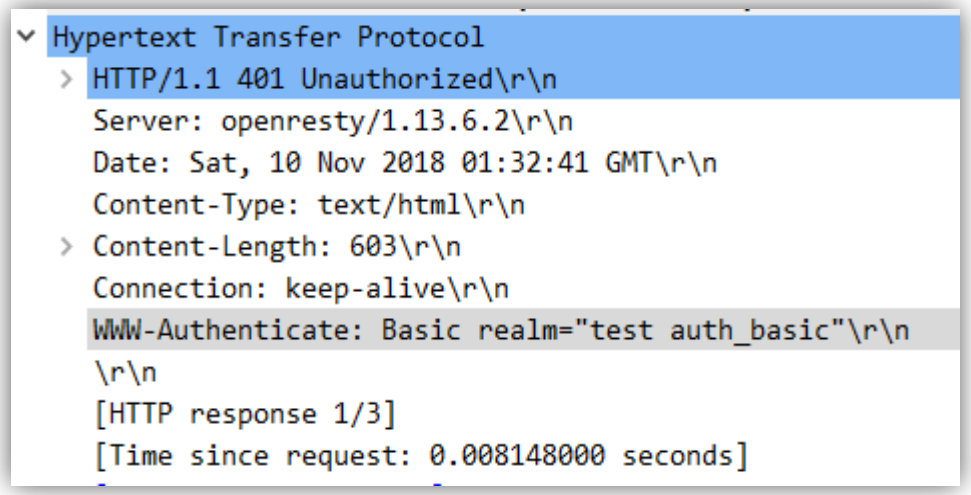
RFC2617：HTTP Basic Authentication

客户端

Nginx

GET / HTTP/1.1





auth_basic 模块的指令

极客时间

功能

基于 HTTP Basic Authutication 协议进行用户名密码的认证

默认编译进 Nginx:

--without-http_auth_basic_module disable ngx_http_auth_basic_module

指令

Syntax: **auth_basic** *string* | off;

Default: auth_basic off;

Context: http, server, location, limit_except

Syntax: **auth_basic_user_file** *file*;

Default: —

Context: http, server, location, limit_except

Nginx 进程管理：信号

comment

name1:password1

name2:password2:comment

name3:password3

依赖安装包：httpd-tools

htpasswd –c file –b user pass

统一的用户权限验证系统

auth_request 模块

功能

向上游的服务转发请求，若上游服务返回的响应码是2xx，则继续执行，若上游服务返回的是401或者403，则将响应返回给客户端

原理

收到请求后，生成子请求，通过反向代理技术把请求传递给上游服务

默认未编译进 Nginx

--with-http_auth_request_module

指令

Syntax: **auth_request** *uri* | off;

Default: auth_request off;

Context: http, server, location

Syntax: **auth_request_set** *$variable value*;

Default: —

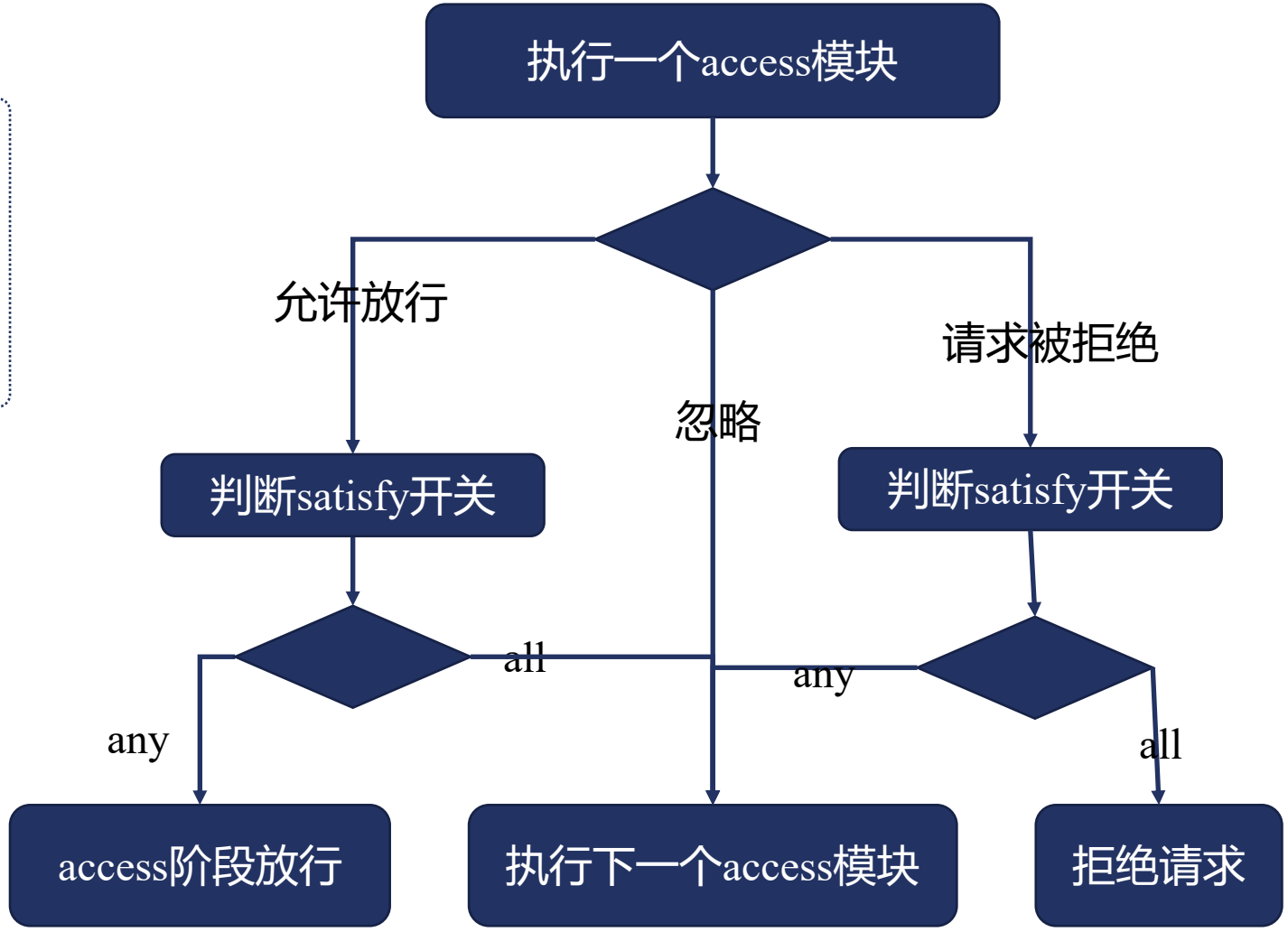
Context: http, server, location

限制所有 access 阶段模块的 satisfy 指令

Syntax: **satisfy** all | any;
Default: satisfy all;
Context: http, server, location

access阶段的模块：

- access模块
- auth_basic模块
- auth_request模块
- 其他模块



问题

1. 如果有return指令，access阶段会生效吗？

2. 多个access模块的顺序有影响吗？

   查看ngx_modules.c

   - &ngx_http_auth_request_module,
   - &ngx_http_auth_basic_module,
   - &ngx_http_access_module,

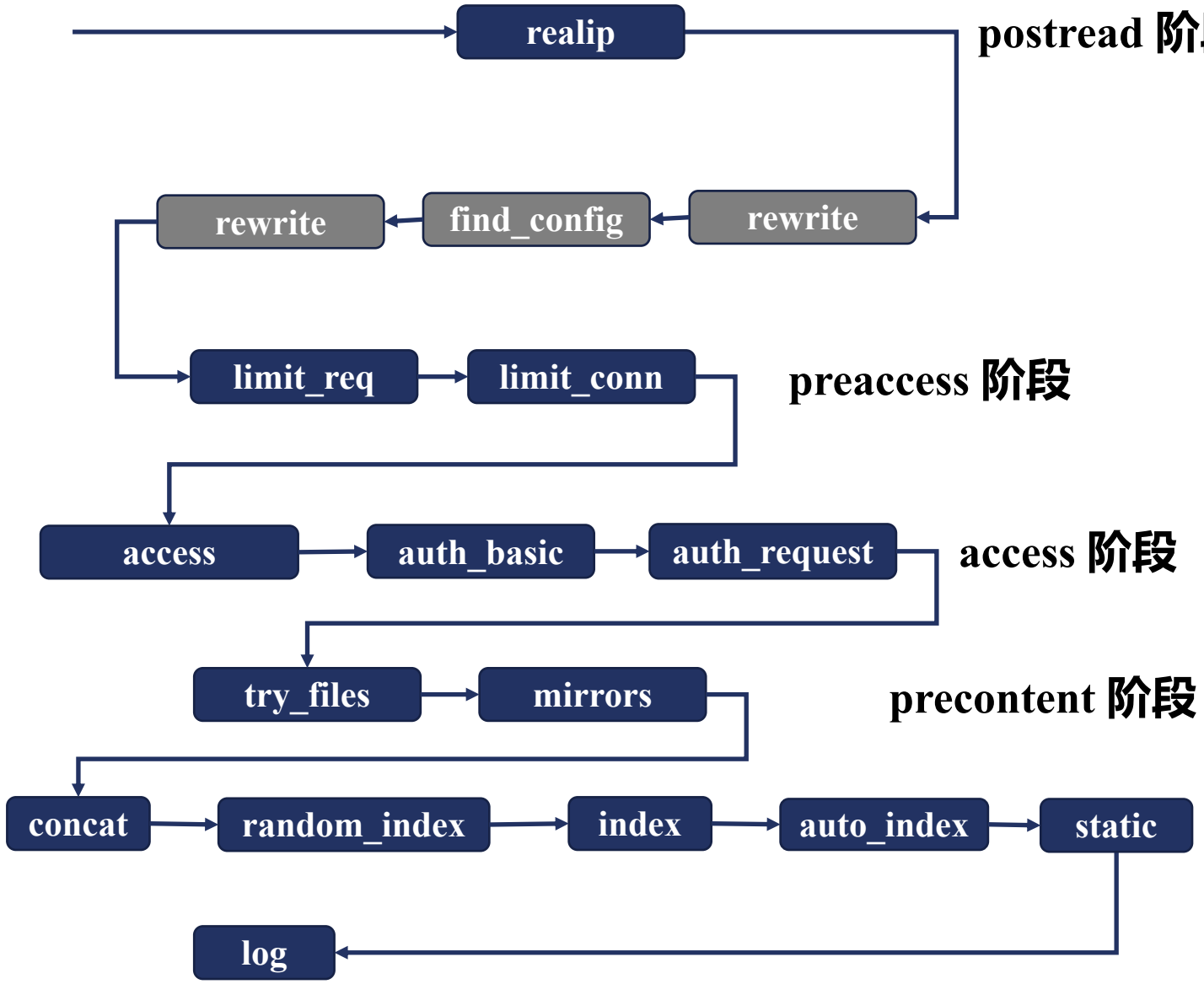
3. 输对密码，下面可以访问到文件吗？

```
location /{
        satisfy any;
        auth_basic "test auth_basic";
        auth_basic_user_file examples/auth.pass;
        deny all;
    }
```

4. 如果把deny all提到auth_basic之前呢？

5. 如果改为allow all，有机会输入密码吗？

11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

precontent 阶段的 try_files 指令

Syntax: **try_files** *file ... uri*;
try_files *file ... =code*;

Default: —

Context: server, location

模块 ngx_http_try_files_module模块

功能

依次试图访问多个url对应的文件（由root或者alias指令指定），当文件存在时直接返回文件内容，如果所有文件都不存在，则按最后一个 URL 结果或者code返回

实时拷贝流量：precontent 阶段的 mirror 模块

模块

ngx_http_mirror_module模块，
默认编译进 Nginx，
通过--without-http_mirror_module移除模块

功能

处理请求时，生成子请求访问其他服务，对子请求的返回值不做处理

Syntax:	**mirror** *uri* | off;
Default:	mirror off;
Context:	http, server, location

Syntax:	**mirror_request_body** on | off;
Default:	mirror_request_body on;
Context:	http, server, location

content 阶段：root 和 alias 指令

Syntax:	**alias** *path*;
Default:	—
Context:	location

Syntax:	**root** *path*;
Default:	root html;
Context:	http, server, location, if in location

将url映射为文件路径，以返回静态文件内容

功能

差别

root会将完整url映射进文件路径中

alias只会将location后的URL映射到文件路径

问题

文件路径

```
html/first/
└── 1.txt
```

访问以下URL会得到什么响应

/root

/alias

/root/1.txt

/alias/1.txt

```
location /root {
        root html;
}

location /alias {
        alias html;
}

location ~ /root/(\w+\.txt) {
        root html/first/$1;
}

location ~ /alias/(\w+\.txt) {
        alias html/first/$1;
}
```

生成待访问文件的三个相关变量

问题

访问/RealPath/1.txt时，
这三个变量的值各为多少？

```
location  /RealPath/ {
        alias html/realpath/;
}
```

realpath_root

将document_root中的软链接等换成真实路径

document_root

由URI和root/alias规则生成的文件夹路径

request_filename

待访问文件的完整路径

静态文件返回时的 content-type

Syntax: **types** { ... }
Default: types { text/html html; image/gif gif; image/jpeg jpg; }
Context: http, server, location

Syntax: **default_type** *mime-type*;
Default: default_type text/plain;
Context: http, server, location

Syntax: **types_hash_bucket_size** *size*;
Default: types_hash_bucket_size 64;
Context: http, server, location

Syntax: **types_hash_max_size** *size*;
Default: types_hash_max_size 1024;
Context: http, server, location

未找到文件时的错误日志

Syntax: **log_not_found** on | off;

Default: log_not_found on;

Context: http, server, location

[error] 10156#0: *10723 open() "/html/first/2.txt/root/2.txt" failed (2: No such file or directory)

问题：

访问目录时 URL 最后没有带/？

static模块实现了root/alias功能时，发现访问目标是目录，但 URL 末尾未加 / 时，会返回301重定向

重定向跳转的域名

Syntax: **server_name_in_redirect** on | off;
Default: server_name_in_redirect off;
Context: http, server, location

Syntax: **port_in_redirect** on | off;
Default: port_in_redirect on;
Context: http, server, location

Syntax: **absolute_redirect** on | off;
Default: absolute_redirect on;
Context: http, server, location

对访问/时的处理：content 阶段的 index 模块

功能 指定/访问时返回index文件内容

ngx_http_index_module

模块

Syntax: **index** *file* ...;

Default: index index.html;

Context: http, server, location

随机 index.html 文件：content 阶段的 autoindex 模块

功能

随机选择index指令指定的一系列index文件中的一个，作为/路径的返回文件内容

模块

ngx_http_index_module

默认不编译进Nginx：

--with-http_random_index_module

Syntax: **random_index** on | off;

Default: random_index off;

Context: location

显示目录内容：content 阶段的 autoindex 模块

当URL以/结尾时，尝试以html/xml/json/jsonp等格式返回root/alias中指向目录的目录结构

ngx_http_index_module

默认编译进Nginx：

--without-http_autoindex_module取消

autoindex 模块的指令

极客时间

Syntax: **autoindex** on | off;
Default: autoindex off;
Context: http, server, location

Syntax: **autoindex_exact_size** on | off;
Default: autoindex_exact_size on;
Context: http, server, location

Syntax: **autoindex_format** html | xml | json | jsonp;
Default: autoindex_format html;
Context: http, server, location

Syntax: **autoindex_localtime** on | off;
Default: autoindex_localtime off;
Context: http, server, location

提升性能：content 阶段的 concat 模块

功能

当页面需要访问多个小文件时，把它们的内容合并到一次http响应中返回，提升性能

模块

ngx_http_concat_module

模块开发者Tengine(https://github.com/alibaba/nginx-http-concat)：

--add-module=../nginx-http-concat/

使用

在uri后加上??，后通过多个,逗号分隔文件。如果还有参数，则在最后通过?添加参数

https://g.alicdn.com/??kissy/k/6.2.4/seed-min.js,kg/global-util/1.0.7/index-min.js,tb/tracker/4.3.5/index.js,kg/tb-nav/2.5.3/index-min.js,secdev/sufei_data/3.3.5/index.js

concat 模块的指令

concat ：on | off
default：concat off
Context：http, server, location

concat_delimiter：string
Default：NONE
Context： http, server, locatione

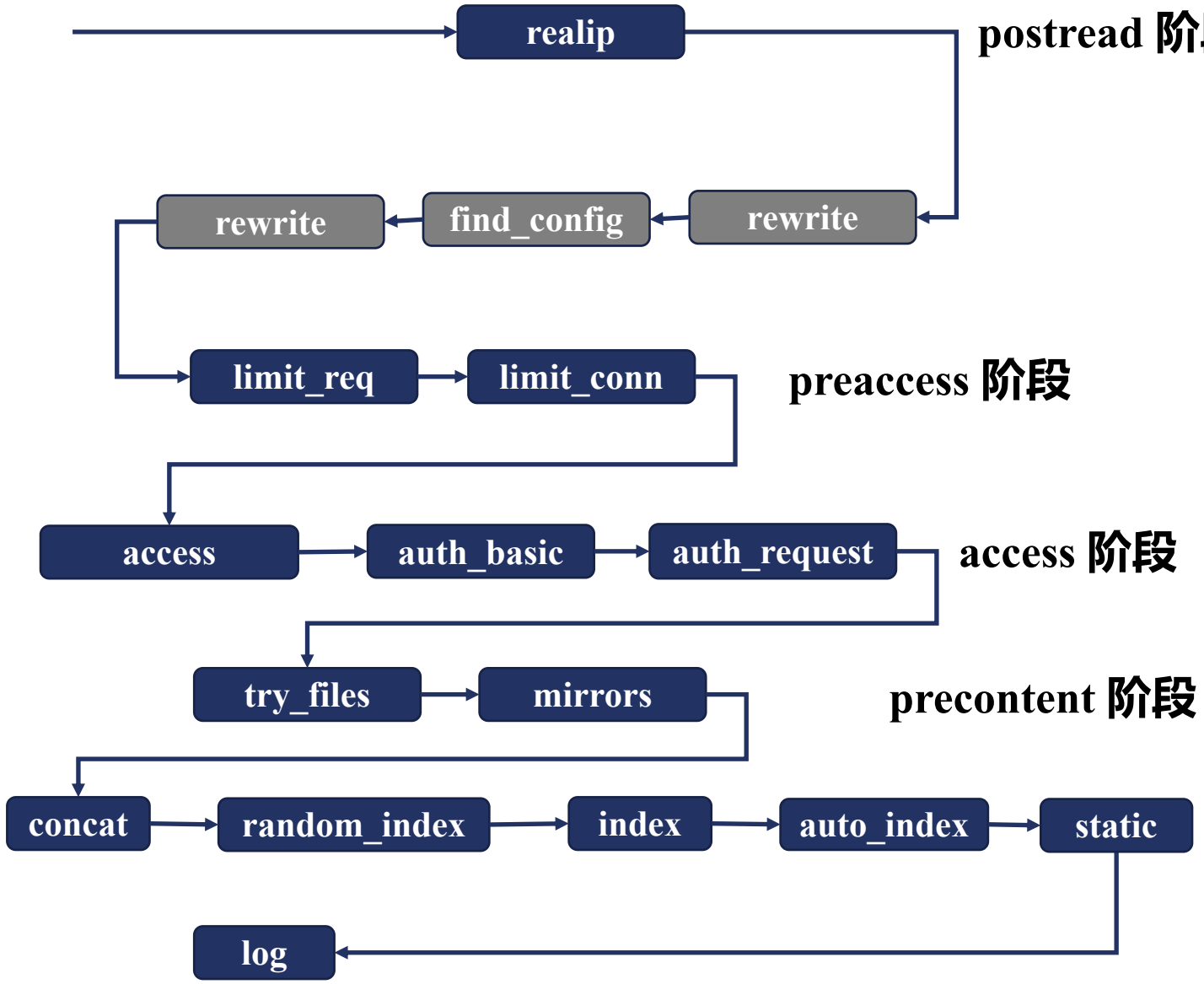
concat_types：MIME types
Default：concat_types: text/css application/x-javascript
Context：http, server, location

concat_unique：on | off
Default： concat_unique on
Context： http, server, location

concat_ignore_file_error：on | off
Default：off
Context：http, server, location

concat_max_files：numberp
Default：concat_max_files 10
Context： http, server, location

11 个阶段的顺序处理



postread 阶段

preaccess 阶段

access 阶段

precontent 阶段

content 阶段

log 阶段

```
char *ngx_module_names[] = {
    … …
    "ngx_http_static_module",
    "ngx_http_autoindex_module",
    "ngx_http_index_module",
    "ngx_http_random_index_module",
    "ngx_http_mirror_module",
    "ngx_http_try_files_module",
    "ngx_http_auth_request_module",
    "ngx_http_auth_basic_module",
    "ngx_http_access_module",
    "ngx_http_limit_conn_module",
    "ngx_http_limit_req_module",
    "ngx_http_realip_module",
    "ngx_http_referer_module",
    "ngx_http_rewrite_module",
    "ngx_http_concat_module",
    … …
}
```

log 阶段：记录请求访问日志的 log 模块

将 HTTP 请求相关
信息记录到日志

ngx_http_log_module，
无法禁用

access 日志格式

Syntax: **log_format** *name* [escape=default|json|none] *string* ...;

Default: log_format combined "...";

Context: http

默认的 combined 日志格式

```
log_format combined '$remote_addr - $remote_user [$time_local] '
'"$request" $status $body_bytes_sent ' '"$http_referer"
"$http_user_agent"';
```

配置日志文件路径

Syntax:	**access_log** *path* [*format* [buffer=*size*] [gzip[=*level*]] [flush=*time*] [if=*condition*]]; **access_log** off;
Default:	access_log logs/access.log combined;
Context:	http, server, location, if in location, limit_except

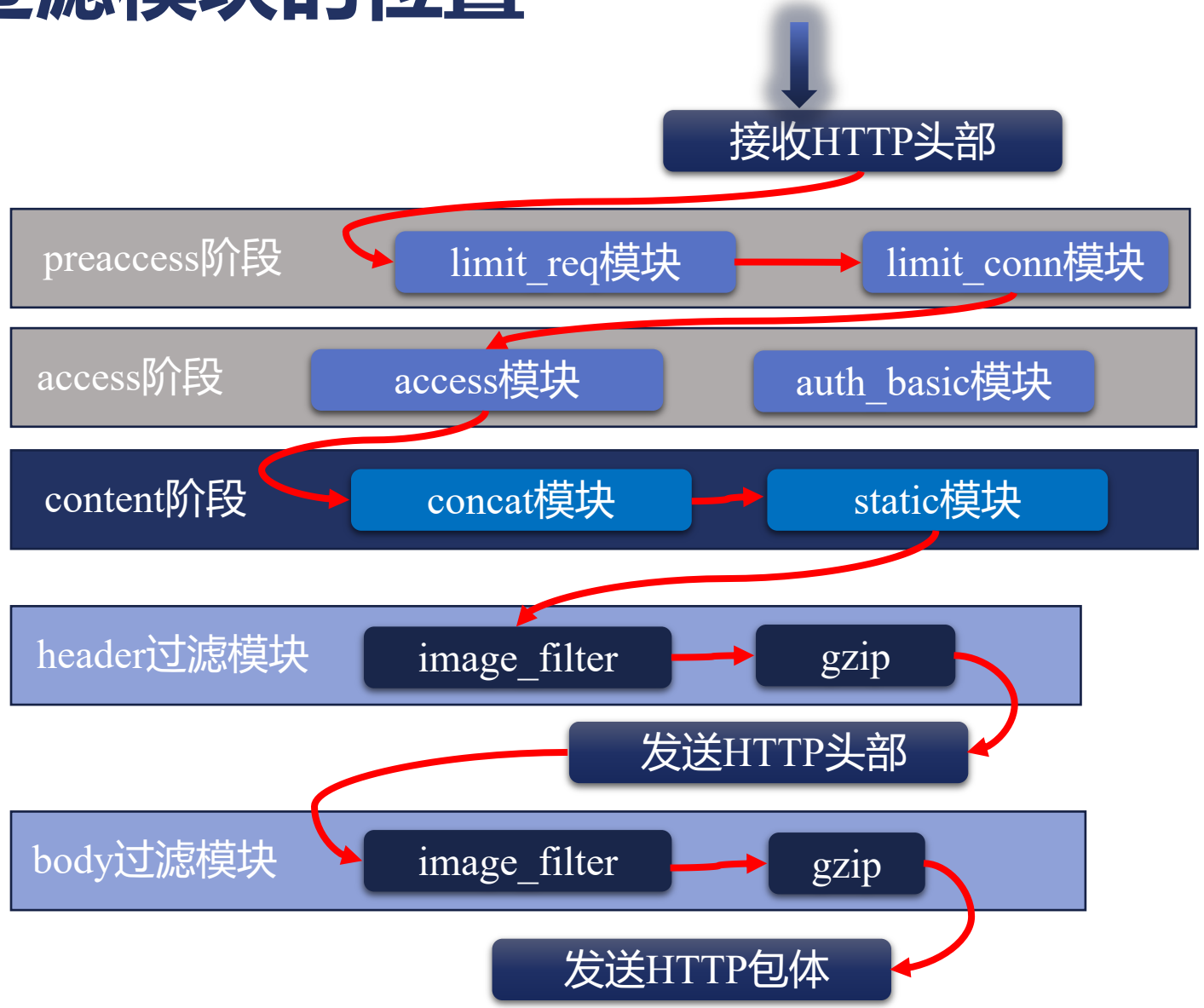
- **path 路径可以包含变量：不打开 cache 时每记录一条日志都需要打开、关闭日志文件**
- **if 通过变量值控制请求日志是否记录**
- **日志缓存**
  - 功能：批量将内存中的日志写入磁盘
  - 写入磁盘的条件
    - 所有待写入磁盘的日志大小超出缓存大小
    - 达到 flush 指定的过期时间
    - worker 进程执行 reopen 命令，或者正在关闭
- **日志压缩**
  - 功能：批量压缩内存中的日志，再写入磁盘
  - buffer 大小默认为 64KB
  - 压缩级别默认为 1（1 最快压缩率最低，9 最慢压缩率最高）

对日志文件名包含变量时的优化

Syntax:	**open_log_file_cache** max=*N* [inactive=*time*] [min_uses=*N*] [valid=*time*]; **open_log_file_cache** off;
Default:	open_log_file_cache off;
Context:	http, server, location

max	缓存内的最大文件句柄数，超出后用 LRU 算法淘汰
inactive	文件访问完后在这段时间内不会被关闭。默认 10 秒
min_uses	在 inactive 时间内使用次数超过 min_uses 才会继续存在内存中。默认 1
valid	超出 valid 时间后，将对缓存的日志文件检查是否存在。默认 60 秒
off	关闭缓存功能

过滤模块的位置




```
limit_req   zone=req_one
burst=120;
limit_conn c_zone 1;


satisfy any;
allow 192.168.1.0/32;
auth_basic_user_file
access.pass;


gzip on;
image_filter resize 80 80;
```

返回响应-加工响应内容

HTTP 过滤模块

copy_filter: 复制包体内容

HTTP 过滤模块

postpone_filter: 处理子请求

HTTP 过滤模块

header_filter: 构造响应头部

write_filter: 发送响应

```
&ngx_http_write_filter_module,
&ngx_http_header_filter_module,
&ngx_http_chunked_filter_module,
&ngx_http_v2_filter_module,
&ngx_http_range_header_filter_module,
&ngx_http_gzip_filter_module,
&ngx_http_postpone_filter_module,
&ngx_http_ssi_filter_module,
&ngx_http_charset_filter_module,
&ngx_http_sub_filter_module,
&ngx_http_addition_filter_module,
&ngx_http_userid_filter_module,
&ngx_http_headers_filter_module,
&ngx_http_echo_module,
&ngx_http_xss_filter_module,
&ngx_http_srcache_filter_module,
&ngx_http_lua_module,
&ngx_http_headers_more_filter_module,
&ngx_http_rds_json_filter_module,
&ngx_http_rds_csv_filter_module,
&ngx_http_copy_filter_module,
&ngx_http_range_body_filter_module,
&ngx_http_not_modified_filter_module,
```

替换响应中的字符串：sub 模块

将响应中指定的字符串，
替换成新的字符串

ngx_http_sub_filter_module 模块，
默认未编译进 Nginx，
通过--with-http_sub_module启用

sub 模块的指令

Syntax: **sub_filter** *string replacement*;
Default: —
Context: http, server, location

Syntax: **sub_filter_last_modified** on | off;
Default: sub_filter_last_modified off;
Context: http, server, location

Syntax: **sub_filter_once** on | off;
Default: sub_filter_once on;
Context: http, server, location

Syntax: **sub_filter_types** *mime-type* ...;
Default: sub_filter_types text/html;
Context: http, server, location

在响应的前后添加内容：addition 模块

在响应前或者响应后增加内容，而增加内容的方式是通过新url响应完成

ngx_http_addition_filter_module模块，默认未编译进Nginx，通过--with-http_addition_module启用

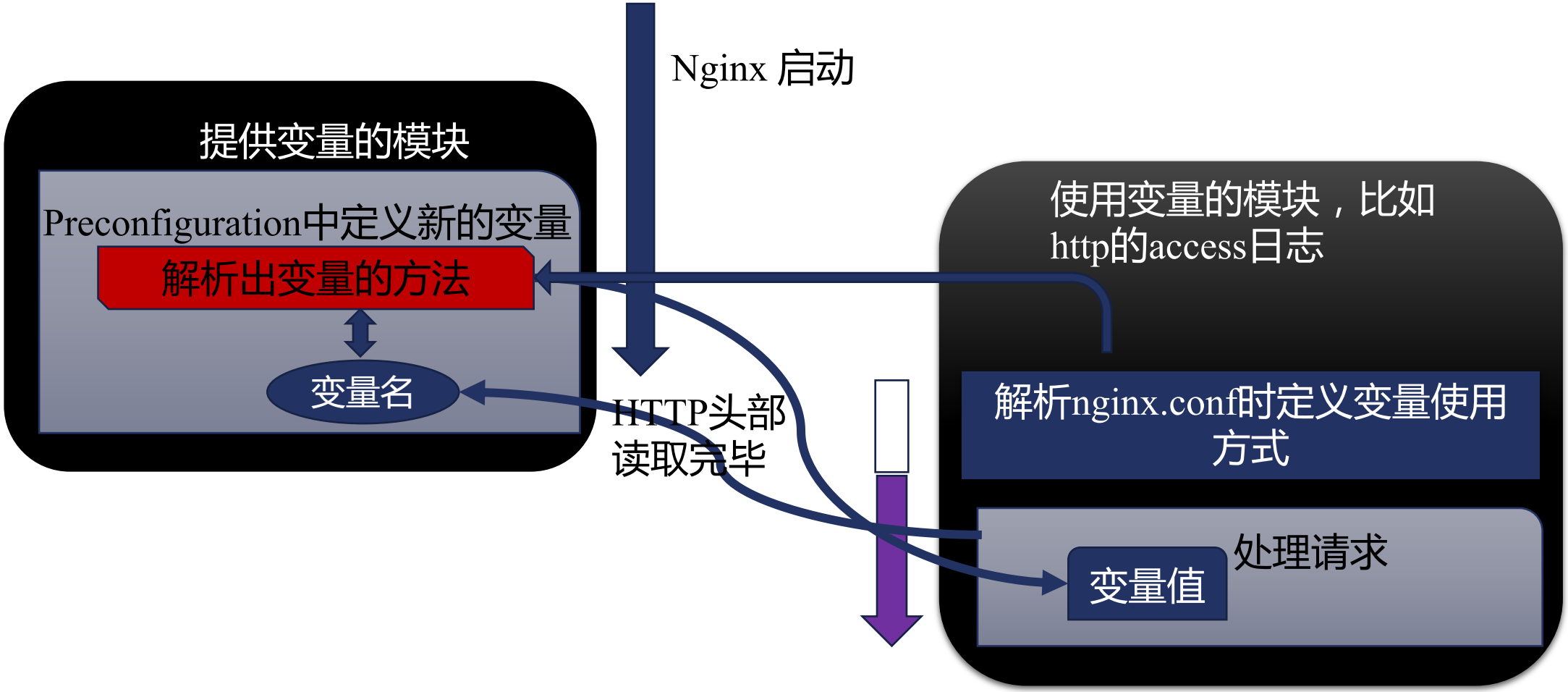
addition 模块的指令

Syntax: **add_before_body** *uri*;
Default: —
Context: http, server, location

Syntax: **add_after_body** *uri*;
Default: —
Context: http, server, location

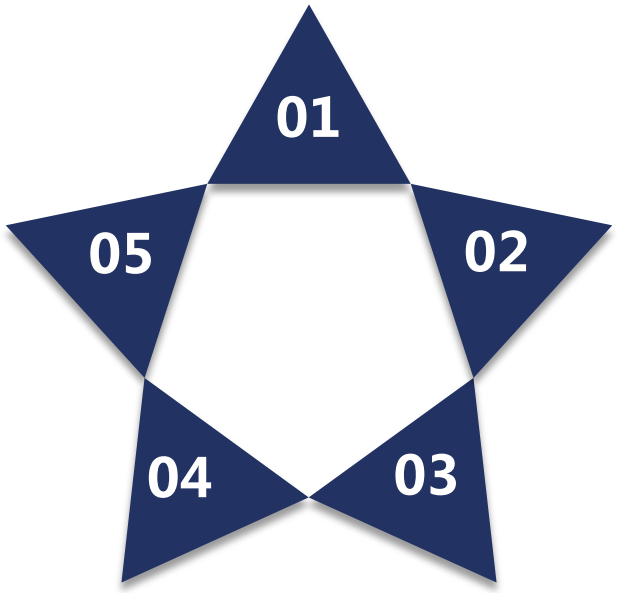
Syntax: **addition_types** *mime-type* ...;
Default: addition_types text/html;
Context: http, server, location

变量的惰性求值


Nginx 启动
提供变量的模块
Preconfiguration中定义新的变量
解析出变量的方法
变量名
HTTP头部
读取完毕
使用变量的模块，比如
http的access日志
解析nginx.conf时定义变量使用
方式
处理请求
变量值

变量的特性

HTTP 框架提供的变量



01 HTTP 请求相关的变量

02 TCP 连接相关的变量

03 Nginx 处理请求过程中产生的变量

04 发送 HTTP 响应时相关的变量

05 Nginx 系统变量

HTTP请求相关的变量（一）

arg_参数名	URL 中某个具体参数的值
query_string	与 args 变量完全相同
args	全部 URL 参数
is_args	如果请求 URL 中有参数则返回?否则返回空
content_length	HTTP 请求中标识包体长度的 Content-Length 头部的值
content_type	标识请求包体类型的 Content-Type 头部的值

HTTP 请求相关的变量（二）

uri	请求的 URI（不同于URL，不包括?后的参数）
document_uri	与uri完全相同
request_uri	请求的 URL（包括 URI 以及完整的参数）
scheme	协议名，例如 HTTP 或者 HTTPS
request_method	请求方法，例如 GET 或者 POST
request_length	所有请求内容的大小，包括请求行、头部、包体等
remote_user	由 HTTP Basic Authentication 协议传入的用户名

HTTP 请求相关的变量（三）

request_body_file

- **临时存放请求包体的文件**
  - 如果包体非常小则不会存文件
  - client_body_in_file_only 强制所有包体存入文件，且可决定是否删除

request_body

请求中的包体，这个变量当且仅当使用反向代理，且设定用内存暂存包体时才有效

request

原始的url请求，含有方法与协议版本，例如GET /?a=1&b=22 HTTP/1.1

HTTP 请求相关的变量（四）

host

- 先从请求行中获取
- 如果含有 Host 头部，则用其值替换掉请求行中的主机名
- 如果前两者都取不到，则使用匹配上的 server_name

HTTP 请求相关的变量（五）

http_头部名字

返回一个具体请求头部的值

特殊

- http_host
- http_user_agent
- http_referer
- http_via
- http_x_forwarded_for
- http_cookie

通用

TCP 连接相关的变量

binary_remote_addr	客户端地址的整型格式，对于IPv4是4字节，对于 IPv6是16字节
connection	递增的连接序号
connection_requests	当前连接上执行过的请求数，对keepalive连接有意义
remote_addr	客户端端口
remote_port	客户端端口
proxy_protocol_addr	若使用了proxy_protocol协议则返回协议中的地址，否则返回空

TCP 连接相关的变量

proxy_protocol_port	若使用了proxy_protocol协议则返回协议中的端口，否则返回空
server_addr	服务器端地址
server_port	服务器端端口
TCP_INFO	tcp内核层参数，包括$tcpinfo_rtt, $tcpinfo_rttvar, $tcpinfo_snd_cwnd, $tcpinfo_rcv_space
server_protocol	服务器端协议，例如HTTP/1.1

Nginx 处理请求过程中产生的变量

request_time
请求处理到现在的耗时，单位为秒，精确到毫秒

server_name
匹配上请求的server_name值

https
如果开启了TLS/SSL，则返回on，否则返回空

request_completion
若请求处理完则返回OK，否则返回空

request_id
以16进制输出的请求标识id，该id共含有16个字节，是随机生成的

Nginx 处理请求过程中产生的变量

request_filename

待访问文件的完整路径

document_root

由URI和root/alias规则生成的文件夹路径

realpath_root

将document_root中的软链接等换成真实路径

limit_rate

返回客户端响应时的速度上限，单位为每秒字节数。可以通过 set 指令修改对请求产生效果

发送 HTTP 响应时相关的变量

body_bytes_sent

- 响应中body包体的长度

bytes_sent

- 全部http响应的长度

status

- http响应中的返回码

sent_trailer_名字

- 把响应结尾内容里值返回

sent_http_头部名字：响应中某个具体头部的值

- **特殊处理**
  - sent_http_content_type
  - sent_http_content_length
  - sent_http_location
  - sent_http_last_modified
  - sent_http_connection
  - sent_http_keep_alive
  - sent_http_transfer_encoding
  - sent_http_cache_control
  - sent_http_link
- **通用**

Nginx 系统变量

time_local	以本地时间标准输出的当前时间，例如14/Nov/2018:15:55:37 +0800
time_iso8601	使用 ISO 8601 标准输出的当前时间，例如2018-11-14T15:55:37+08:00
nginx_version	Nginx 版本号
pid	所属 worker 进程的进程 id
pipe	使用了管道则返回 p，否则返回 .
hostname	所在服务器的主机名，与 hostname 命令输出一致
hostname	1970年1月1日到现在的时间，单位为秒，小数点后精确到毫秒

简单有效的防盗链手段：referer 模块

场景

- 某网站通过url引用了你的页面，当用户在浏览器上点击url时，http请求的头部中会通过referer头部，将该网站当前页面的url带上，告诉服务器本次请求是由这个页面发起的

目的

- 拒绝非正常的网站访问我们站点的资源

思路

- 通过referer模块，用invalid_referer变量根据配置判断referer头部是否合法

referer 模块

- 默认编译进Nginx，通过--without-http_referer_module禁用

referer 模块的指令

Syntax: **valid_referers** none | blocked | server_names | *string* ...;
Default: —
Context: server, location

Syntax: **referer_hash_bucket_size** *size*;
Default: referer_hash_bucket_size 64;
Context: server, location

Syntax: **referer_hash_max_size** *size*;
Default: referer_hash_max_size 2048;
Context: server, location

valid_referers 指令

可同时携带多个参数，表示多个 referer 头部都生效

参数值

none

- 允许缺失referer头部的请求访问

block

- 允许referer头部没有对应的值的请求访问

server_names

- 若referer中站点域名与server_name中本机域名某个匹配，则允许该请求访问

表示域名及URL的字符串，对域名可在前缀或者后缀中含有*通配符

- 若referer头部的值匹配字符串后，则允许访问

正则表达式

- 若referer头部的值匹配正则表达式后，则允许访问

invalid_referer 变量

- 允许访问时变量值为空
- 不允许访问时变量值为1

问题

```
server_name referer.taohui.tech;

location /{
        valid_referers none blocked server_names
            *.taohui.pub www.taohui.org.cn/nginx/
            ~\.google\.;

        if ($invalid_referer) {
            return 403;
        }
}
```

以下请求哪些会被拒绝？

curl -H 'referer: http://www.taohui.org.cn/ttt' referer.taohui.tech/

curl -H 'referer: http://www.taohui.pub/ttt' referer.taohui.tech/

curl -H 'referer: ' referer.taohui.tech/

curl referer.taohui.tech/

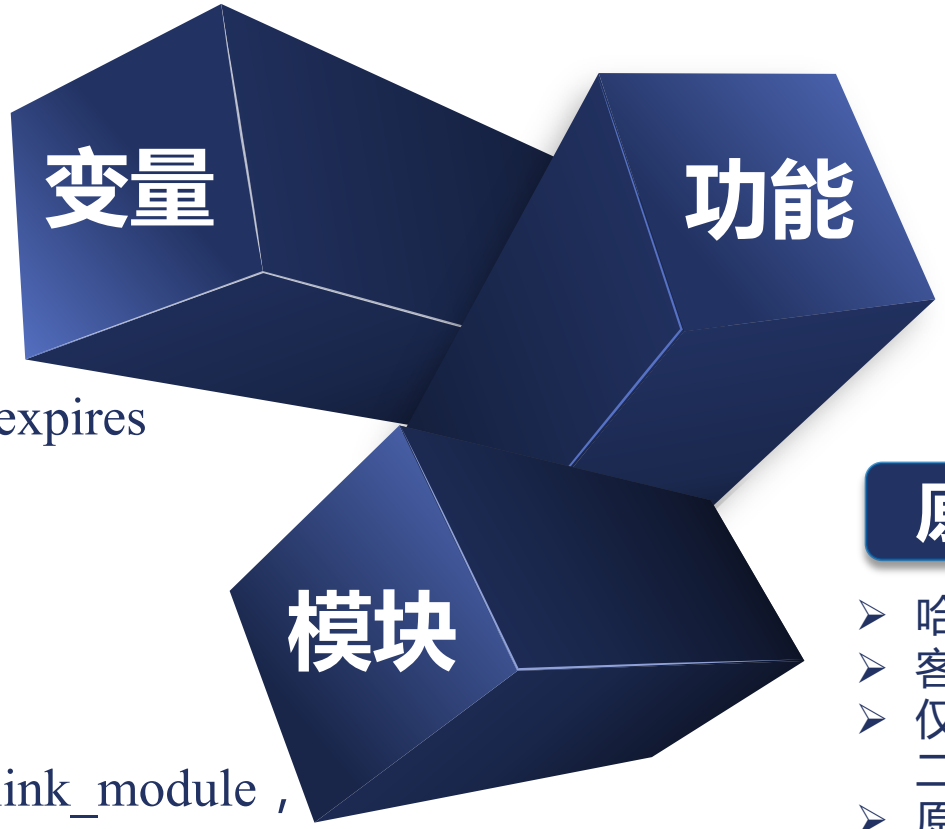
curl -H 'referer: http://www.taohui.tech' referer.taohui.tech/

curl -H 'referer: http://referer.taohui.tech' referer.taohui.tech/

curl -H 'referer: http://image.baidu.com/search/detail' referer.taohui.tech/

curl -H 'referer: http://image.google.com/search/detail' referer.taohui.tech/

防盗链的一种解决方案：secure_link 模块



- secure_link
- secure_link_expires

ngx_http_secure_link_module，

默认未编译进nginx，

需要通过--with-http_secure_link_module添加

通过验证URL中哈希值的方式防盗链

过程

- 由某服务器（也可以是nginx）生成加密后的安全链接url，返回给客户端
- 客户端使用安全url访问nginx，由nginx的secure_link变量判断是否验证通过

原理

- 哈希算法是不可逆的
- 客户端只能拿到执行过哈希算法的URL
- 仅生成URL的服务器、验证URL是否安全的nginx这二者，才保存执行哈希算法前的原始字符串
- 原始字符串通常由以下部分有序组成：
  - 资源位置，例如HTTP中指定资源的URI，防止攻击者拿到一个安全URL后可以访问任意资源
  - 用户信息，例如用户IP地址，限制其他用户盗用安全URL
  - 时间戳，使安全URL及时过期。
  - 密钥，仅服务器端拥有，增加攻击者猜测出原始字符串的难度

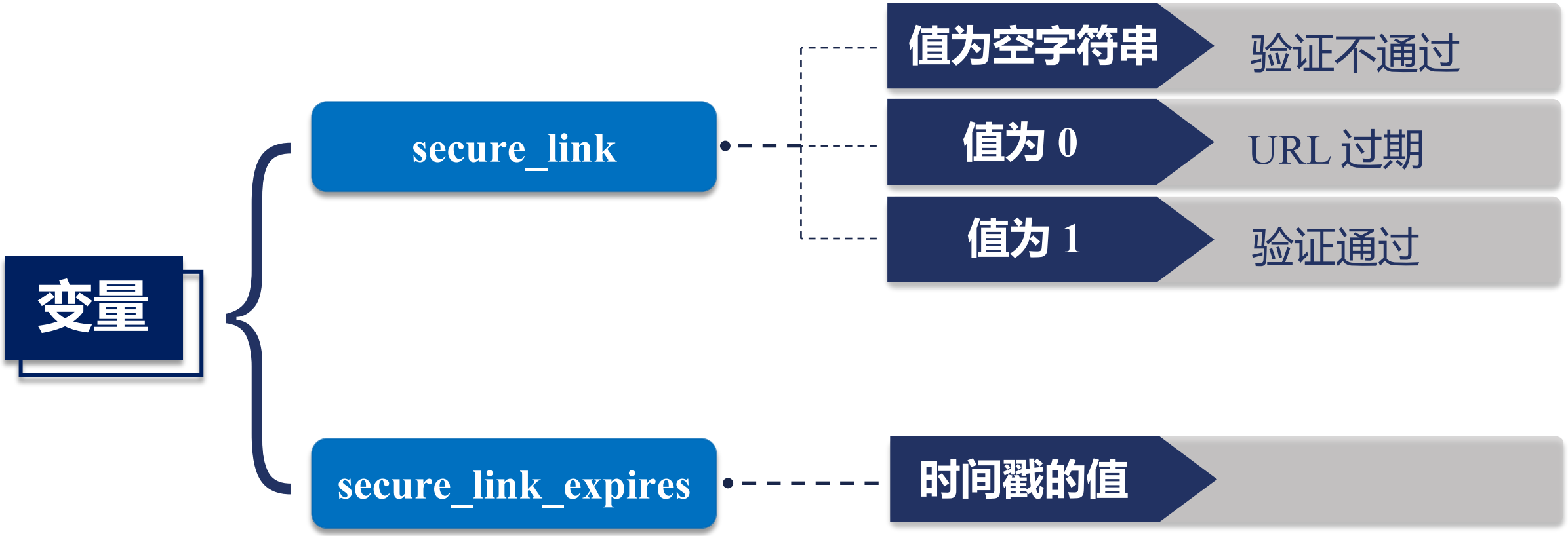
secure_link 模块指令

Syntax: **secure_link** *expression*;
Default: —
Context: http, server, location

Syntax: **secure_link_md5** *expression*;
Default: —
Context: http, server, location

Syntax: **secure_link_secret** *word*;
Default: —
Context: location

变量值及带过期时间的配置示例



变量值及带过期时间的配置示例

示例

命令行生成安全链接

- **原请求：**
  - /test1.txt?md5=md5生成值&expires=时间戳（如2147483647）
- **生成md5**
  - echo -n '时间戳URL客户端IP 密钥' | openssl md5 -binary | openssl base64 | tr +/ - | tr -d =

Nginx 配置

- **secure_link $arg_md5,$arg_expires;**
- **secure_link_md5 "$secure_link_expires$uri$remote_addr secret";**

仅对 URI 进行哈希的简单办法

01 将请求 URL 分为三个部分：
/prefix/hash/link

02 Hash 生成方式

- 对“link密钥”做md5哈希求值

03 用 secure_link_secret secret;
配置密钥

仅对 URI 进行哈希的简单办法

示例

命令行生成安全链接

- **原请求：**
  - link
- **生成的安全请求**
  - /prefix/md5/link
- **生成md5**
  - echo -n 'linksecret' | openssl md5 –hex

Nginx配置

- **secure_link_secret secret;**

map 模块的指令

Syntax: **map** *string $variable* { ... }
Default: —
Context: http

Syntax: **map_hash_bucket_size** *size*;
Default: map_hash_bucket_size 32|64|128;
Context: http

Syntax: **map_hash_max_size** *size*;
Default: map_hash_max_size 2048;
Context: http

通过映射新变量提供更多的可能性：map 模块

基于已有变量，使用类似switch {case: … default: …}的语法创建新变量，为其他基于变量值实现功能的模块提供更多的可能性

ngx_http_map_module，
默认编译进Nginx，
通过--without-http_map_module禁用

通过映射新变量提供更多的可能性：map 模块

已有变量
- 字符串
- 一个或者多个变量
- 变量与字符串的组合

case 规则
- 字符串严格匹配
- 使用hostnames指令，可以对域名使用前缀*泛域名匹配
- 使用hostnames指令，可以对域名使用后缀*泛域名匹配
- ~和~*正则表达式匹配，后者忽略大小写

default 规则
- 没有匹配到任何规则时，使用default
- 缺失default时，返回空字符串给新变量

其他
- 使用include语法提升可读性
- 使用volatile禁止变量值缓存

问题

对右边的map配置，当以下请求发生时，name变量值是？

- 'Host: map.taohui.org.cn’
- 'Host: map.tao123.org.cn’
- 'Host: map.taohui.pub’
- 'Host: map.taohui.tech’

```
map $http_host $name {
    hostnames;

    default       0;

    ~map\.tao\w+\.org.cn 1;
    *.taohui.org.cn   2;
    map.taohui.tech   3;
    map.taohui.*    4;
}

map $http_user_agent $mobile {
    default       0;
    "~Opera Mini" 1;
}
```

实现 AB 测试：split_clients 模块

ngx_http_split_clients_module，默认编译进Nginx，通过—without-http_split_clients_module禁用

基于已有变量创建新变量，为其他AB测试提供更多的可能性

- 对已有变量的值执行MurmurHash2算法得到32位整型哈希数字，记为hash
- 32位无符号整型的最大数字2^32-1，记为max
- 哈希数字与最大数字相除hash/max，可以得到百分比percent
- 配置指令中指示了各个百分比构成的范围，如0-1%,1%-5%等，及范围对应的值
- 当percent落在哪个范围里，新变量的值就对应着其后的参数

已有变量

- 字符串
- 一个或者多个变量
- 变量与字符串的组合

case规则

- xx.xx%，支持小数点后2位，所有项的百分比相加不能超过100%
- *，由它匹配剩余的百分比（100%减去以上所有项相加的百分比）

split_clients 模块的指令

Syntax: **split_clients** *string $variable* { ... }

Default: —

Context: http

下面这行配置有问题吗？

```
split_clients "${http_testcli}" $variant {
        0.51%       .one;
        20.0%       .two;
        50.5%       .three;
        40%         .four;
        *           "";
}
```

根据客户端地址创建新变量：geo 模块

Syntax: **geo** [*$address*] *$variable* { ... }

Default: —

Context: http

功能

根据IP地址创建新变量

模块

ngx_http_geo_module，

默认编译进nginx，

通过--without-http_geo_module禁用

根据客户端地址创建新变量：geo 模块

规则

- **如果geo指令后不输入$address，那么默认使用$remote_addr变量作为IP地址**
- **{ } 内的指令匹配：优先最长匹配**
  - 通过IP地址及子网掩码的方式，定义IP范围，当IP地址在范围内时新变量使用其后的参数值
  - default指定了当以上范围都未匹配上时，新变量的默认值
  - 通过proxy指令指定可信地址（参考realip模块），此时remote_addr的值为X-Forwarded-For头部值中最后一个IP地址
  - proxy_recursive允许循环地址搜索
  - include，优化可读性
  - delete删除指定网络

geo 模块示例

```
geo $country {
    default        ZZ;
    #include        conf/geo.conf;
    proxy          116.62.160.193;

    127.0.0.0/24   US;
    127.0.0.1/32   RU;
    10.1.0.0/16    RU;
    192.168.1.0/24 UK;
}
```

问题：以下命令执行时，变量country的值各为多少？（proxy为客户端地址）

- **curl -H 'X-Forwarded-For: 10.1.0.0,127.0.0.2' geo.taohui.tech**
- **curl -H 'X-Forwarded-For: 10.1.0.0,127.0.0.1' geo.taohui.tech**
- **curl -H 'X-Forwarded-For: 10.1.0.0,127.0.0.1,1.2.3.4' geo.taohui.tech**

基于MaxMind数据库从客户端地址获取变量：geoip模块

根据IP地址创建新变量

ngx_http_geoip_module，
默认未编译进nginx，
通过--with-http_geoip_module禁用

- 安装MaxMind里geoip的C开发库（ https://dev.maxmind.com/geoip/legacy/downloadable/ ）
- 编译nginx时带上--with-http_geoip_module参数
- 下载MaxMind中的二进制地址库
- 使用geoip_country或者geoip_city指令配置好nginx.conf
- 运行（或者升级nginx）

geoip_country 指令提供的变量

Syntax: **geoip_country** *file*;
Default: —
Context: http

Syntax: **geoip_proxy** *address* | *CIDR*;
Default: —
Context: http

变量

- **$geoip_country_code**
  - 两个字母的国家代码，比如CN或者US
- **$geoip_country_code3**
  - 三个字母的国家代码，比如CHN或者USA
- **$geoip_country_name**
  - 国家名称，例如 “China”, “United States”.

geoip_city 指令提供的变量

Syntax: **geoip_city** *file*;

Default: —

Context: http

geoip_city 指令提供的变量

- $geoip_latitude：纬度
- $geoip_longitude：经度
- $geoip_city_continent_code：属于全球哪个洲，例如EU或者AS
- 与geoip_country指令生成的变量重叠
  - $geoip_city_country_code：两个字母的国家代码，比如CN或者US
  - $geoip_city_country_code3：三个字母的国家代码，比如CHN或者USA
  - $geoip_city_country_name：国家名称，例如“China”，“United States”
- $geoip_region：洲或者省的编码，例如02
- $geoip_region_name：洲或者省的名称，例如Zhejiang或者Saint Petersburg
- $geoip_city：城市名
- $geoip_postal_code：邮编号
- $geoip_area_code：仅美国使用的电话区号，例如408
- $geoip_dma_code：仅美国使用的DMA编号，例如807

对客户端 keepalive 行为控制的指令

功能

多个HTTP请求通过复用TCP连接，实现以下功能

- 减少握手次数
- 通过减少并发连接数减少了服务器资源的消耗
- 降低TCP拥塞控制的影响

协议

- Connection头部：取值为close或者keepalive，前者表示请求处理完即关闭连接，后者表示复用连接处理下一条请求
- Keep-Alive头部：其值为timeout=n，后面的数字n单位是秒，告诉客户端连接至少保留n秒

对客户端 keepalive 行为控制的指令

Syntax: **keepalive_disable** none | *browser* ...;
Default: keepalive_disable msie6;
Context: http, server, location

Syntax: **keepalive_requests** *number*;
Default: keepalive_requests 100;
Context: http, server, location

Syntax: **keepalive_timeout** *timeout* [*header_timeout*];
Default: keepalive_timeout 75s;
Context: http, server, location

扫码试看/订阅

《Nginx 核心知识100讲》