浅析Nginx架构和设计

owenzhang

自我介绍

- 目前：Web服务与风控组
- 经历：多年QQ后台服务器端开发
- 对高并发、分布式的海量互联网服务架构有一定的理解

课程简介

- 目标：
  - 了解Nginx的设计理念，为深入研究Nginx web服务器、设计高并发网络服务器打基础。
- 内容
  - 介绍Nginx的架构和设计思想；
  - 详细解读Nginx的内存池、hash结构和匹配算法。

目录

Nginx简介

- Nginx描述
- Nginx主要功能
- 在活跃网站中占有率
- Nginx与Apache对比

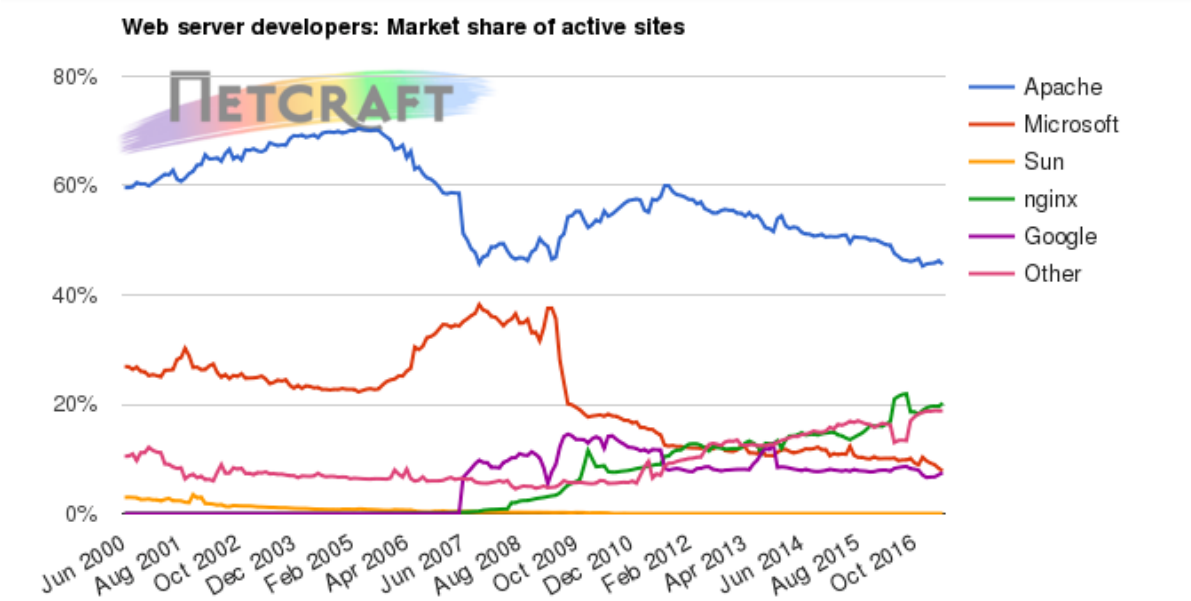
Nginx描述

- 高性能的HTTP和反向代理服务器
- 是一个IMAP/POP3/SMTP服务器
- 由Igor Sysoev为俄罗斯访问量第二的Rambler.ru站点开发
- 第一个公开版本0.1.0发布于2004年10月4日
- 源代码以类BSD许可证的形式发布

Nginx主要功能

- http服务器
- 静态资源服务器
- 反向代理
  - 负载均衡

在活跃网站中占有率



Developer	April 2017	Percent	May 2017	Percent	Change
Apache	78,489,472	46.28%	77,080,638	45.61%	-0.67
nginx	33,176,490	19.56%	34,172,855	20.22%	0.66
Microsoft	14,033,779	8.28%	13,225,171	7.83%	-0.45
Google	12,048,089	7.10%	12,698,425	7.51%	0.41

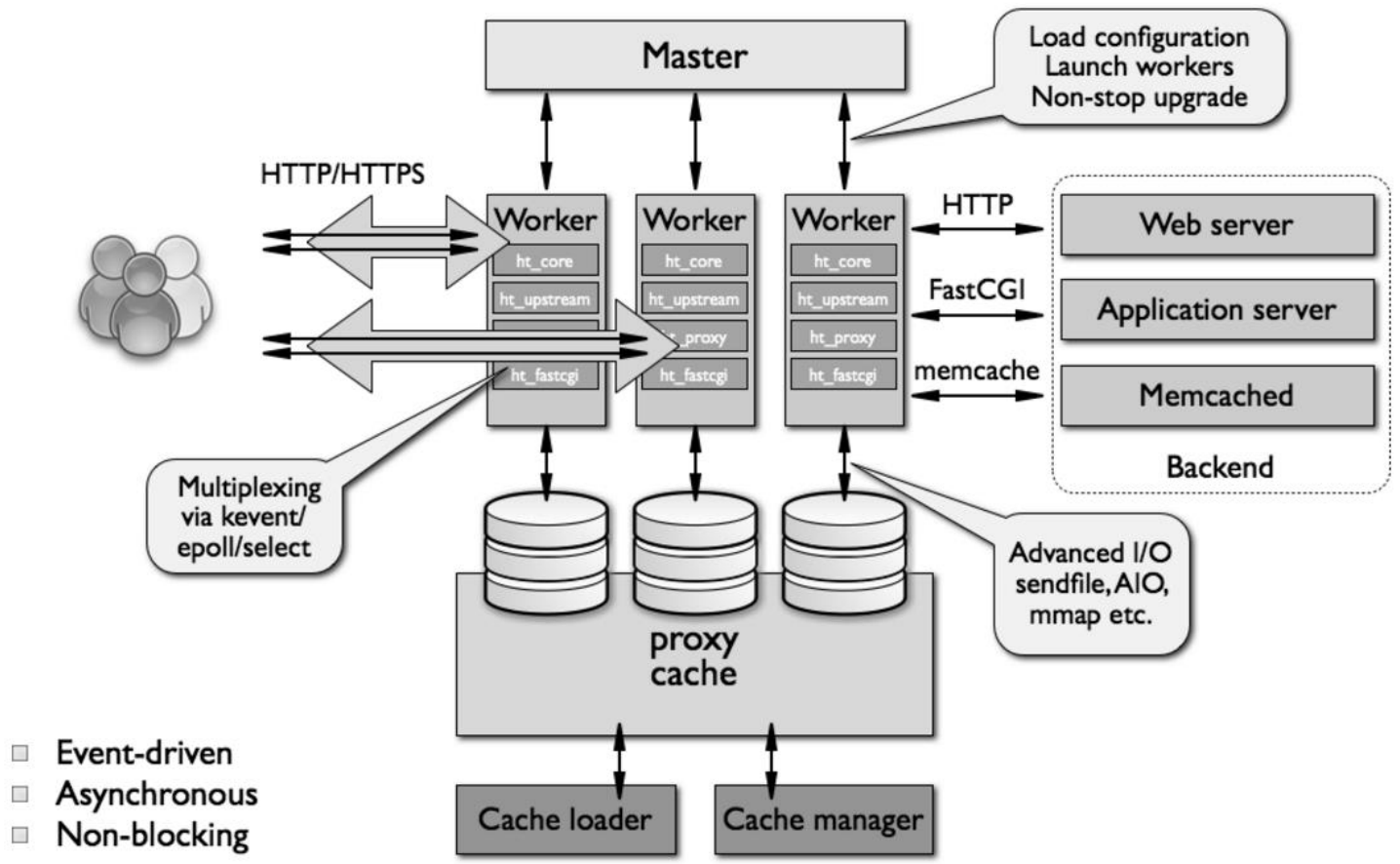
Nginx与Apache对比

	Nginx	Apache
设计模型	异步； 多个连接对应一个进程	同步； 一个连接对应一个进程
占用资源	少	多
功能模块	少	多

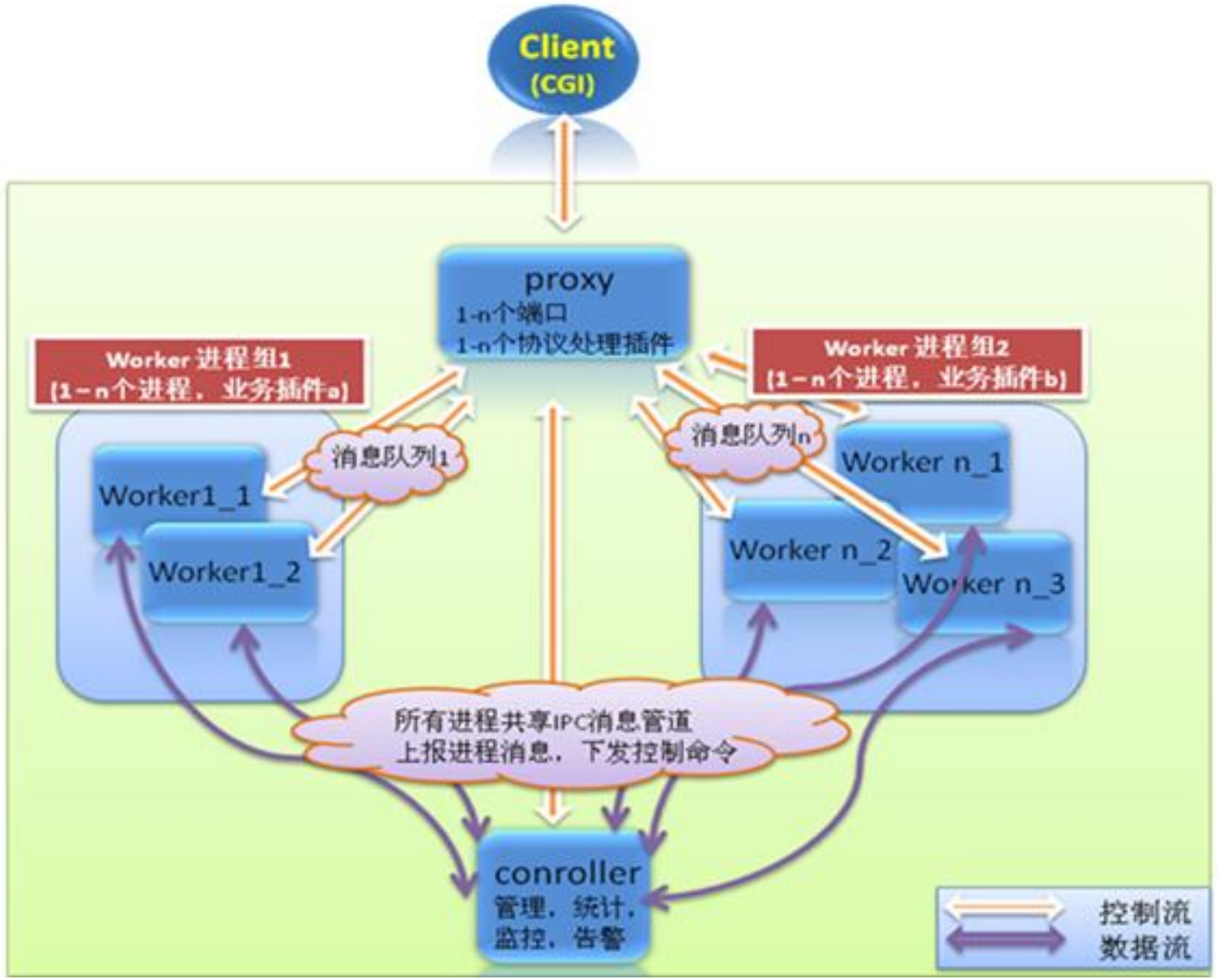
Nginx软件架构

- 整体架构
- 与spp架构对比
- 思考

整体架构


Master
Load configuration
Launch workers
Non-stop upgrade
HTTP/HTTPS
Worker
Worker
Worker
ht_core
ht_upstream
ht_proxy
ht_fastcgi
HTTP
FastCGI
memcache
Web server
Application server
Memcached
Backend
Multiplexing via kevent/ epoll/select
Advanced I/O sendfile, AIO, mmap etc.
proxy cache
Cache loader
Cache manager
Event-driven
Asynchronous
Non-blocking

与spp架构对比


Client
(CGI)
proxy
1-n个端口
1-n个协议处理插件
Worker 进程组1
(1－n个进程，业务插件a)
Worker 进程组2
(1－n个进程，业务插件b)
消息队列1
消息队列n
Worker1_1
Worker1_2
Worker n_1
Worker n_2
Worker n_3
所有进程共享IPC消息管道
上报进程消息，下发控制命令
conroller
管理，统计，
监控，告警
控制流
数据流

思考

- Nginx的worker相当于spp的哪部分？
- Spp和Nginx哪种设计好，为什么？
- 为什么Spp没有使用Nginx的设计方法？

主要模块分析

- 惊群效应的处理
- 内存池
- 域名匹配hash结构和算法

惊群效应的处理

- 惊群定义
- 避免惊群方法
- 避免惊群的配置
- 惊群影响

惊群定义

- 惊群现象（thundering herd）就是当多个进程和线程在同时阻塞等待同一个事件时，如果这个事件发生，会唤醒所有的进程，但最终只可能有一个进程/线程对该事件进行处理，其他进程/线程会在失败后重新休眠，这种性能浪费就是惊群。

避免惊群方法

- 进程先获取到accept锁
- 然后把监听fd加入到监听队列
- 消费掉accept队列中的事件
- 释放锁

避免惊群的配置

Syntax: **accept_mutex** on | off;
Default: accept_mutex off;
Context: events

If accept_mutex is enabled, worker processes will accept new connections by turn. Otherwise, all worker processes will be notified about new connections, and if volume of new connections is low, some of the worker processes may just waste system resources.

> There is no need to enable accept_mutex on systems that support the EPOLLEXCLUSIVE flag (1.11.3) or when using reuseport.

reuseport

this parameter (1.9.1) instructs to create an individual listening socket for each worker process (using the SO_REUSEPORT socket option), allowing a kernel to distribute incoming connections between worker processes. This currently works only on Linux 3.9+ and DragonFly BSD.

> Inappropriate use of this option may have its security implications.

惊群影响

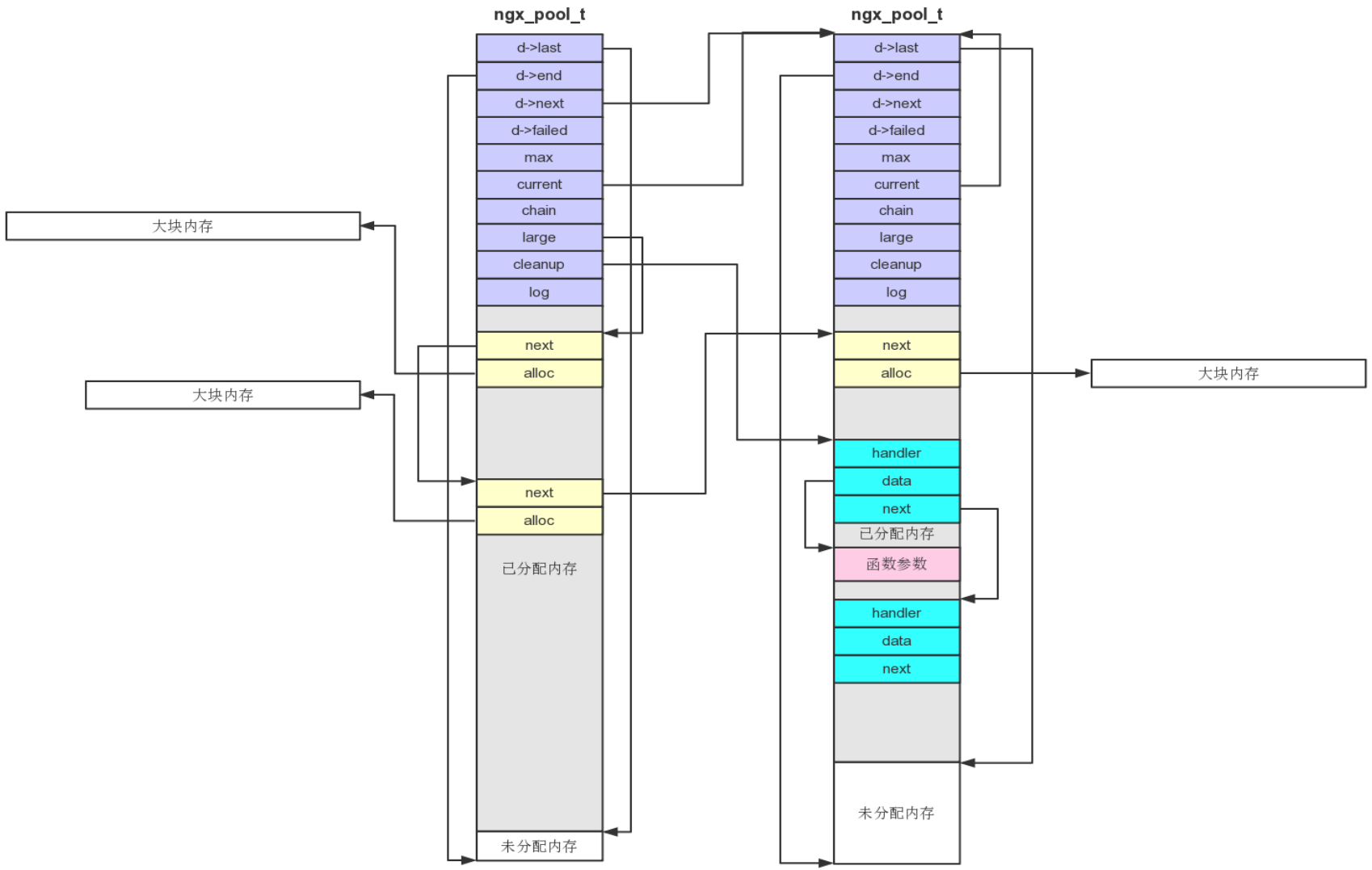
If Listen is single Apache workers just call blocking accept().
If Listen are several worker can not just call blocking accept() on one listening socket. It calls select() for all listening sockets instead, and then calls accept() for returned socket.

nginx workers always use select/kqueue/epoll/etc. before accept().

OS may wake all processes waiting on accept() and select(), this is called http://en.wikipedia.org/wiki/Thundering_herd_problem
This is a problem if you have a lot of workers as in Apache (hundreds and more), but this insensible if you have just several workers as nginx usually has. Therefore turning accept_mutex off is as scheduling incoming connection by OS via select/kqueue/epoll/etc (but not accept().

--
Igor Sysoev
http://sysoev.ru/en/

内存池



- 只分配不释放
- 一次分配4k
- 字节对齐
- 空间换时间

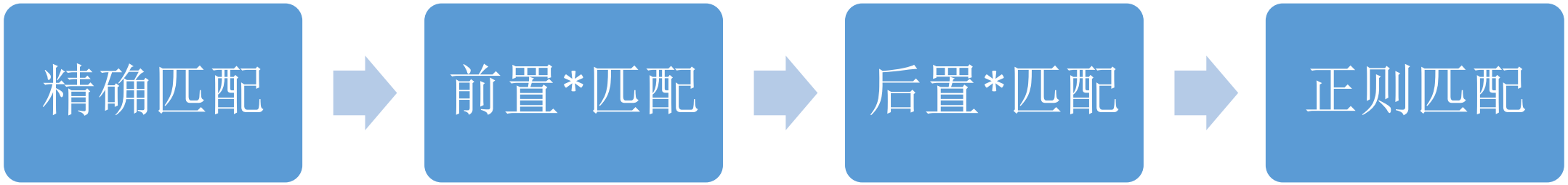
域名匹配hash结构和算法

- 匹配顺序
- 精确匹配
- 前后置通配符匹配

匹配顺序

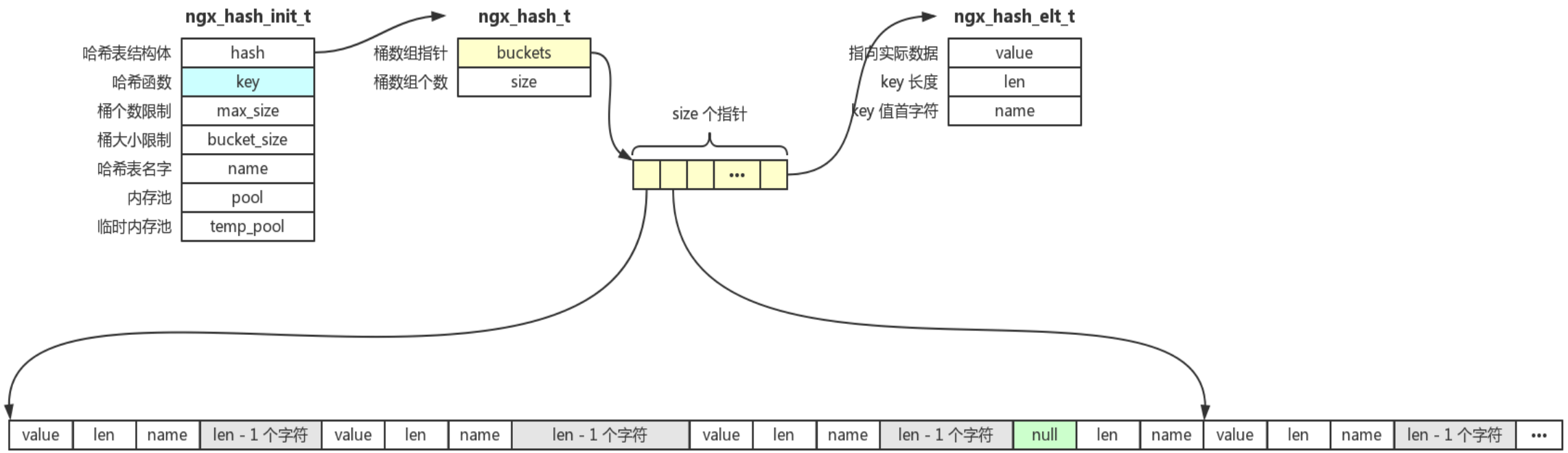
- 输入www.test.com返回什么？
- 输入test.com返回什么？
- 输入www.test.im返回什么？
- 输入www.testtest.com返回什么？
- 调整顺序后再重试上面操作返回什么？

匹配顺序



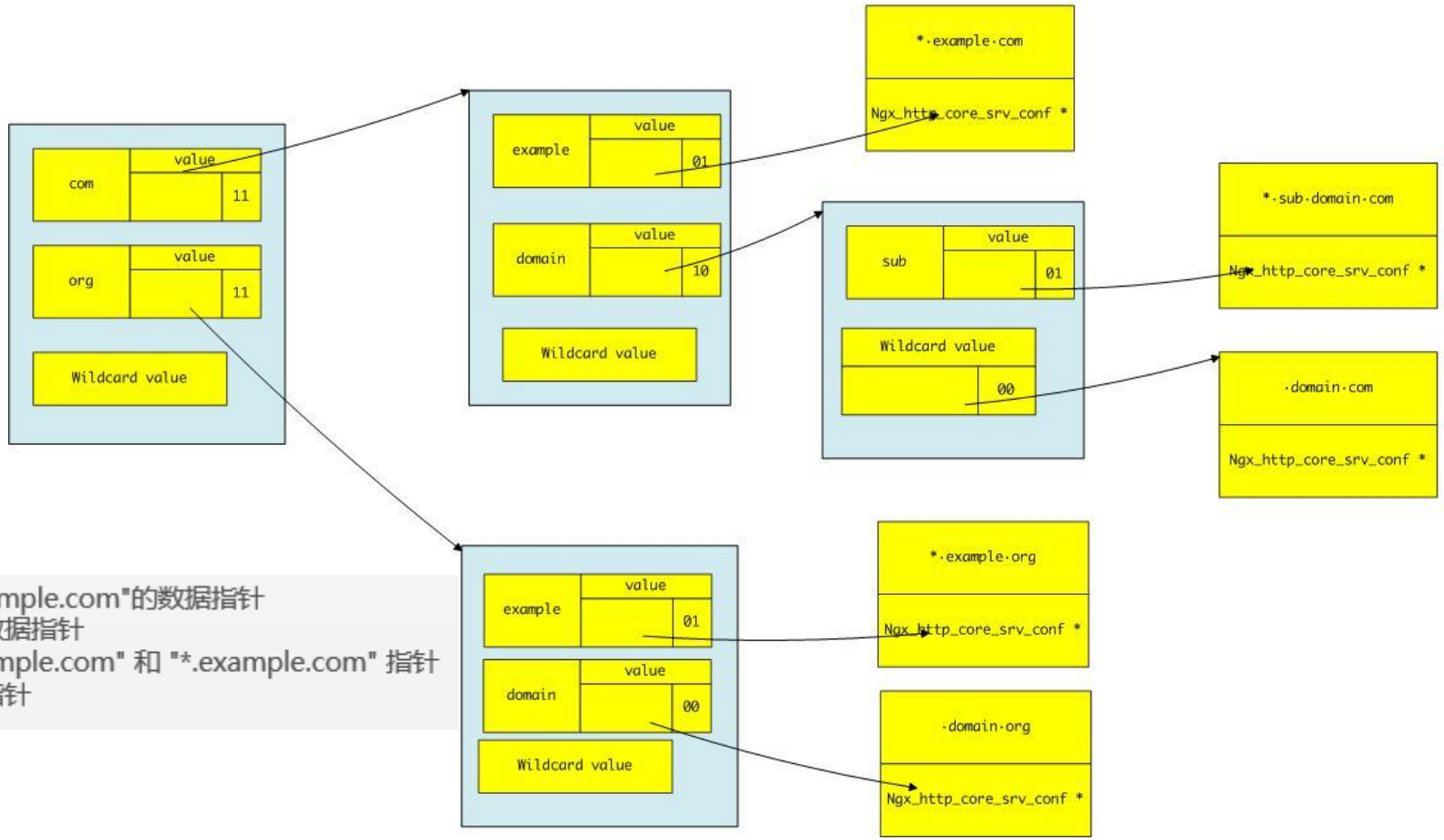
精确匹配

- #define ngx_hash(key, c) ((ngx_uint_t) key * 31 + c)
- server_names_hash_bucket_size和server_names_hash_max_size参数



前后置通配符匹配

如果wildcard hash中有“.domain.com”，“*.sub.domain.com”，“*.example.com”，“*.example.org”，“.domain.org”五个元素，处理后为“com.domain\0”，“com.domain.sub.\0”，“com.example.\0”，“org.example.\0”“org.domain\0”，初始化后的哈希表结构如下：



00 - value 是 "example.com" 和 "*.example.com"的数据指针
01 - value 仅仅是 "*.example.com"的数据指针
10 - value 是 支持通配符哈希表是 "example.com" 和 "*.example.com" 指针
11 - value 仅仅是 "*.example.com"的指针

常用tips

- location匹配顺序？
- 如何强制跳转https？
- 如何优化大消息体传输速度？
- 如何优化静态文件传输速度？
- 如何规范日志格式，方便数据分析？
- nginx -t检查会有写操作吗？

Nginx配置结构

```
main                                    # 全局配置
events {                                # nginx工作模式配置
}
http {                                    # http设置
    ....
    server {                            # 服务器主机配置
        ....
        location {                        # 路由配置
            ....
        }
        location path {
            ....
        }
        location otherpath {
            ....
        }
    }
    server {
        ....

        location {
            ....
        }
    }
    upstream name {                        # 负载均衡配置
        ....
    }
}
```

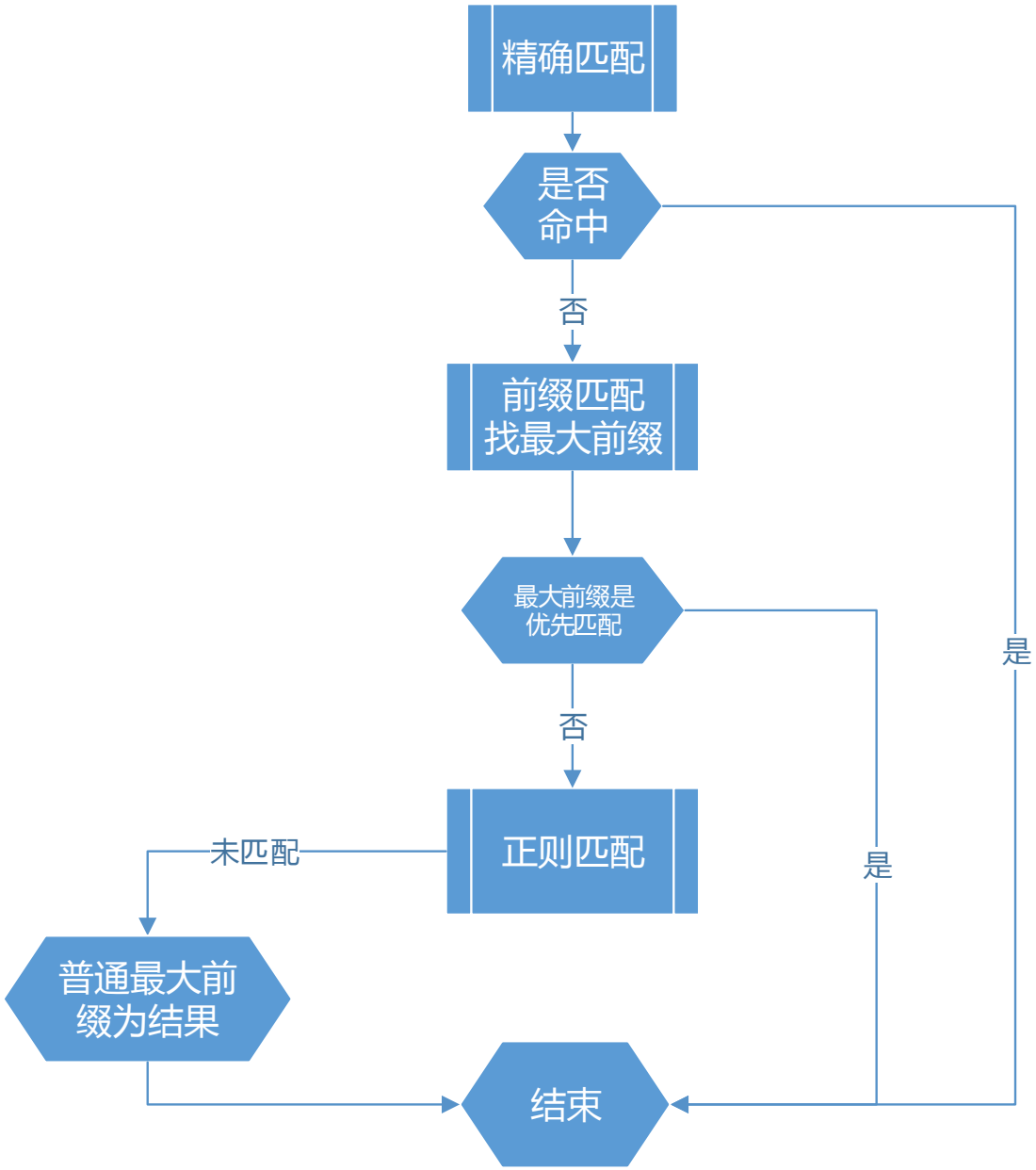
Location匹配顺序

pattern	含义
location = /uri	精确匹配
location ^~ /uri	优先前缀匹配
location ~ pattern	正则匹配-区分大小写的
location ~* pattern	正则匹配-不区分大小写的
location /uri	普通前缀匹配

Location匹配顺序

- 访问location.test.com
- 访问location.test.com/
- 访问location.test.com/justtest
- 访问location.test.com/test/just
- 访问location.test.com/just

Location匹配顺序



强制跳转https

- rewrite ^(.*)$ https://$host$1 permanent;
- Syntax: rewrite regex replacement [flag];
- flag
  - last
  - break
  - permanent
  - redirect

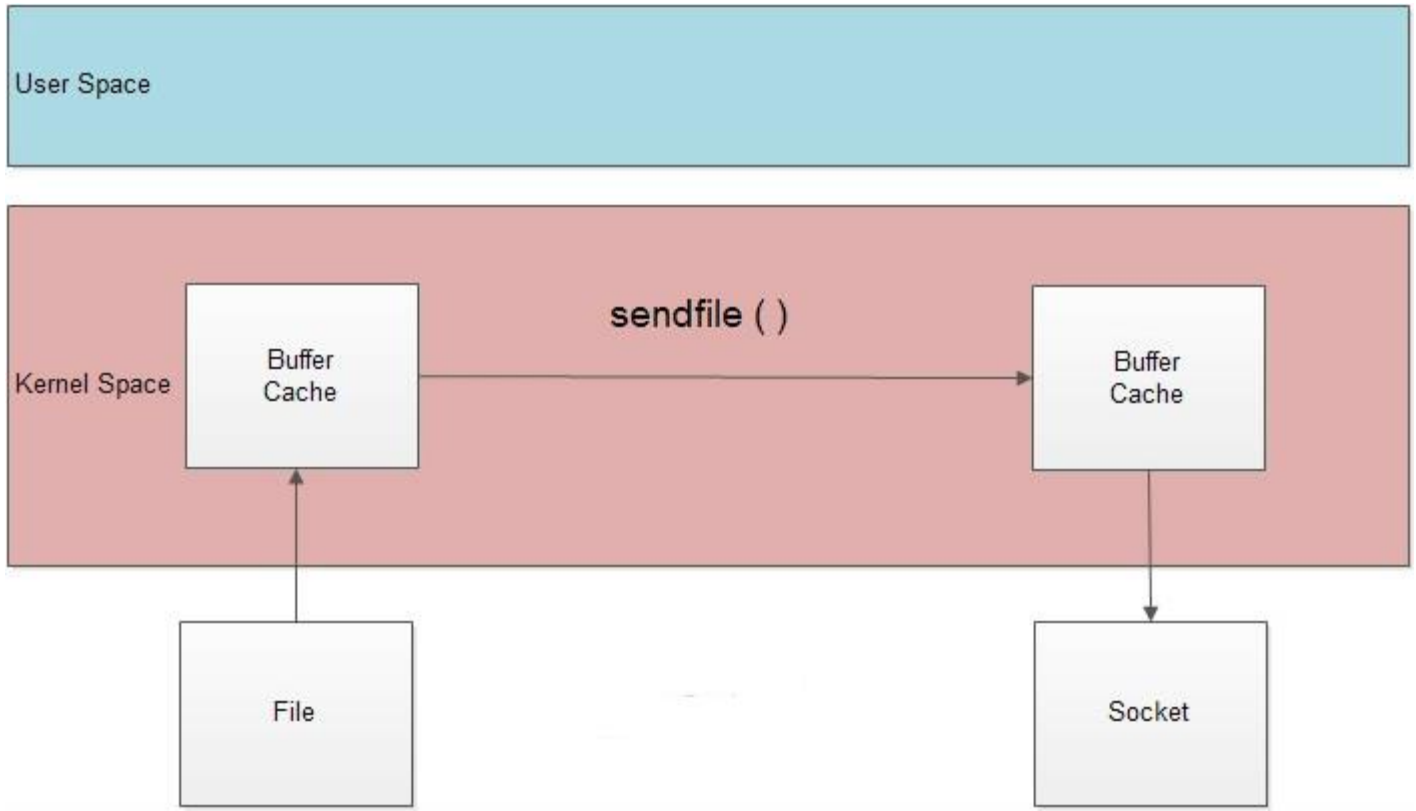
优化消息体传输

- gzip

```
gzip on;
gzip_min_length  1k;
gzip_buffers     4 16k;
gzip_http_version 1.0;
gzip_comp_level 2;
gzip_types       text/plain application/x-javascript text/css application/xml application/javascript image/svg+xml;
gzip_vary on;
```

优化静态文件传输

- sendfile



规范日志格式

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent $request_time $upstream_response_time "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /data/log/nginx/access.log  main;

    charset  utf-8;
```

```
http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    log_format access_log_json
'{"user_ip":"$http_x_real_ip","lan_ip":"$remote_addr","log_time":"$time_iso8601","user_req":"$request","http_code":"$status","body_by
tes_sents":"$body_bytes_sent","req_time":"$request_time","user_ua":"$http_user_agent"}';    # 这行是新添加的，指定为json格式，键值对的格式

    access_log  /var/log/nginx/access.log access_log_json;   # 使用刚才定义的日志格式
```

Nginx -t与用户名

- Syntax: user user [group];
- Default: user nobody nobody;

- 注意：Nginx -t检查配置会更改目录权限！

The End !

Thanks for listening !