PHP程序猿应该知道的Nginx(中)

Panda

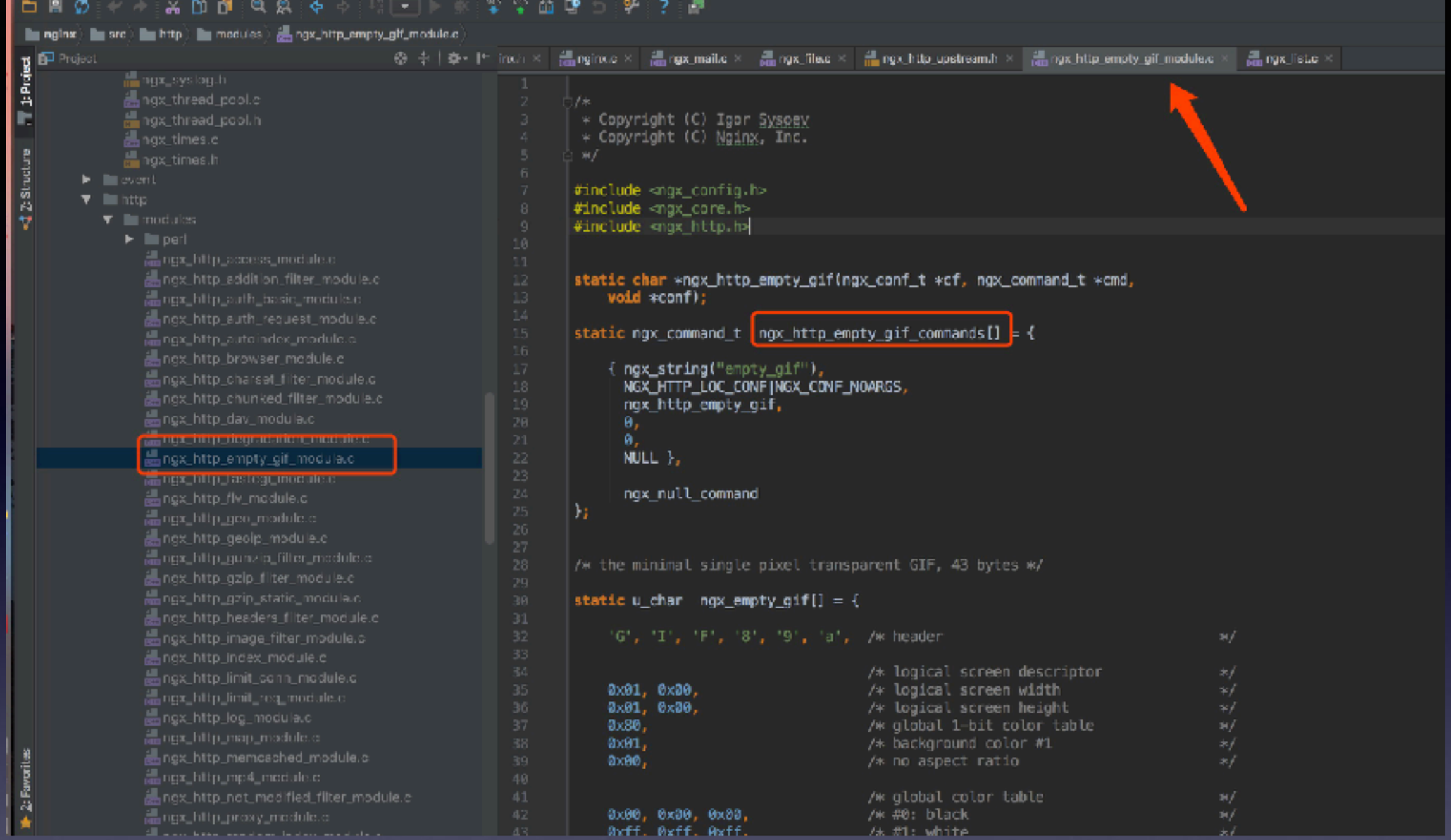
INDEX

- Nginx有趣的功能
- Nginx架构
- Nginx工作原理
- Nginx Upstream
- Nginx负载均衡
- 剖析Nginx高性能的秘密

empty_gif模块

- 1.用过百度统计的小伙伴可能发现了,百度使用1x1的空白图片传递统计参数
- 2. 为什么不是用自己的图片传参数,而是用静态图片呢?

- Nginx里面的空白图片是放在内存中的,速度绝对比硬盘快N倍
- Nginx默认内置ngx_http_empty_gif_module模块
- 下面一起看一下ngx_http_empty_gif_module模块的代码



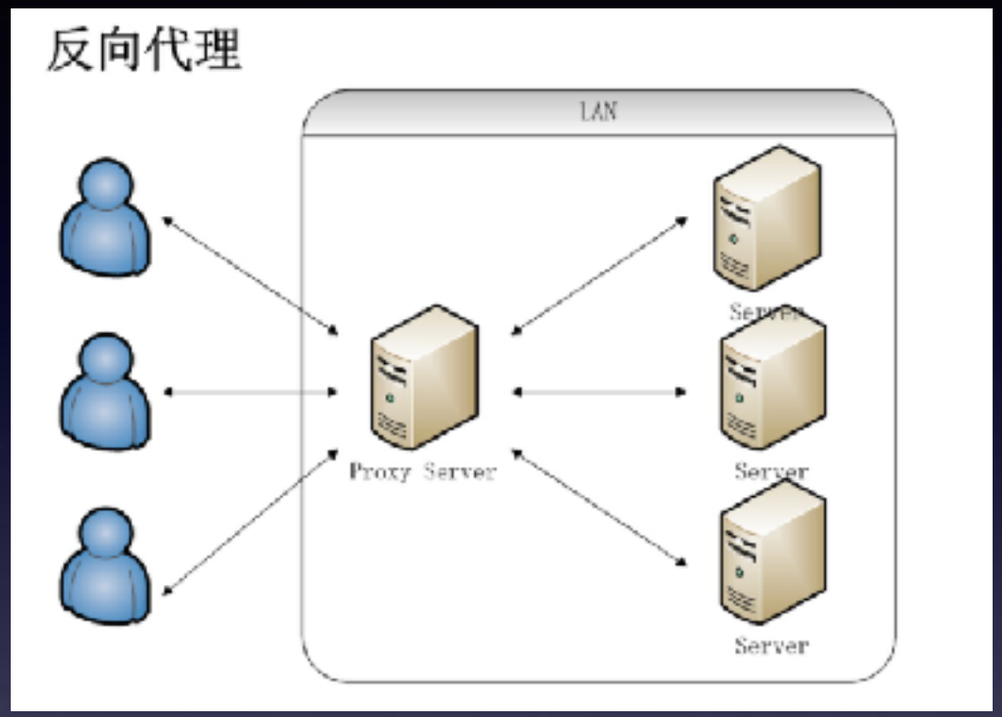
ngx_http_empty_gif_module.c

https://github.com/nginx/nginx/blob/master/src/http/modules/ngx_http_empty_gif_module.c

- 语法:　empty_gif;
- 配置: —
- 配置段: location
- 结果:　开启后响应1*1空白图片
- 具体操作步骤: 传送门

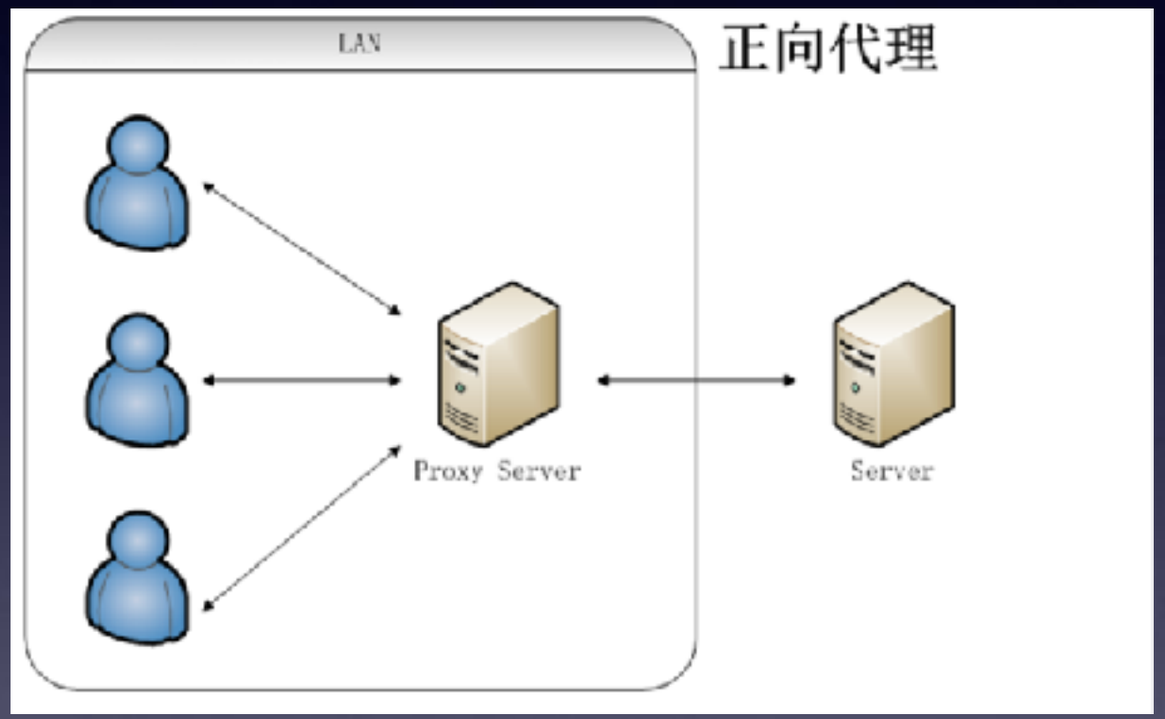
Nginx之反向代理

1. 支持多种形式的代理方式:
Location URL host 等
2.可以方便的修改request和response
3.支持多种形式的负载均衡:
ip_hash、round_robin、轮询、权重、url hash、fair等
4. 故障处理 一台后端服务器请求失败,会自动请求下一台
5. 可以灵活的设置 404页面
等等



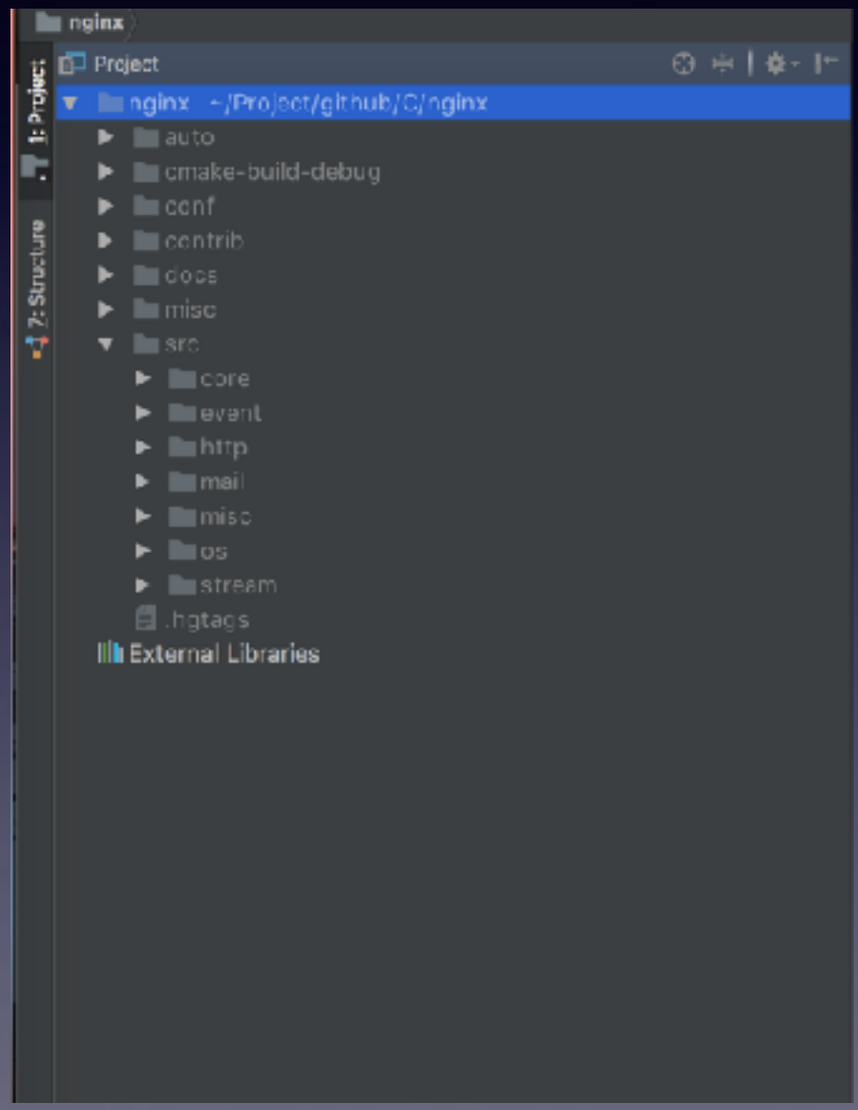
Nginx之正向代理

- 1.一般用于让内网的用户访问外网



Nginx架构

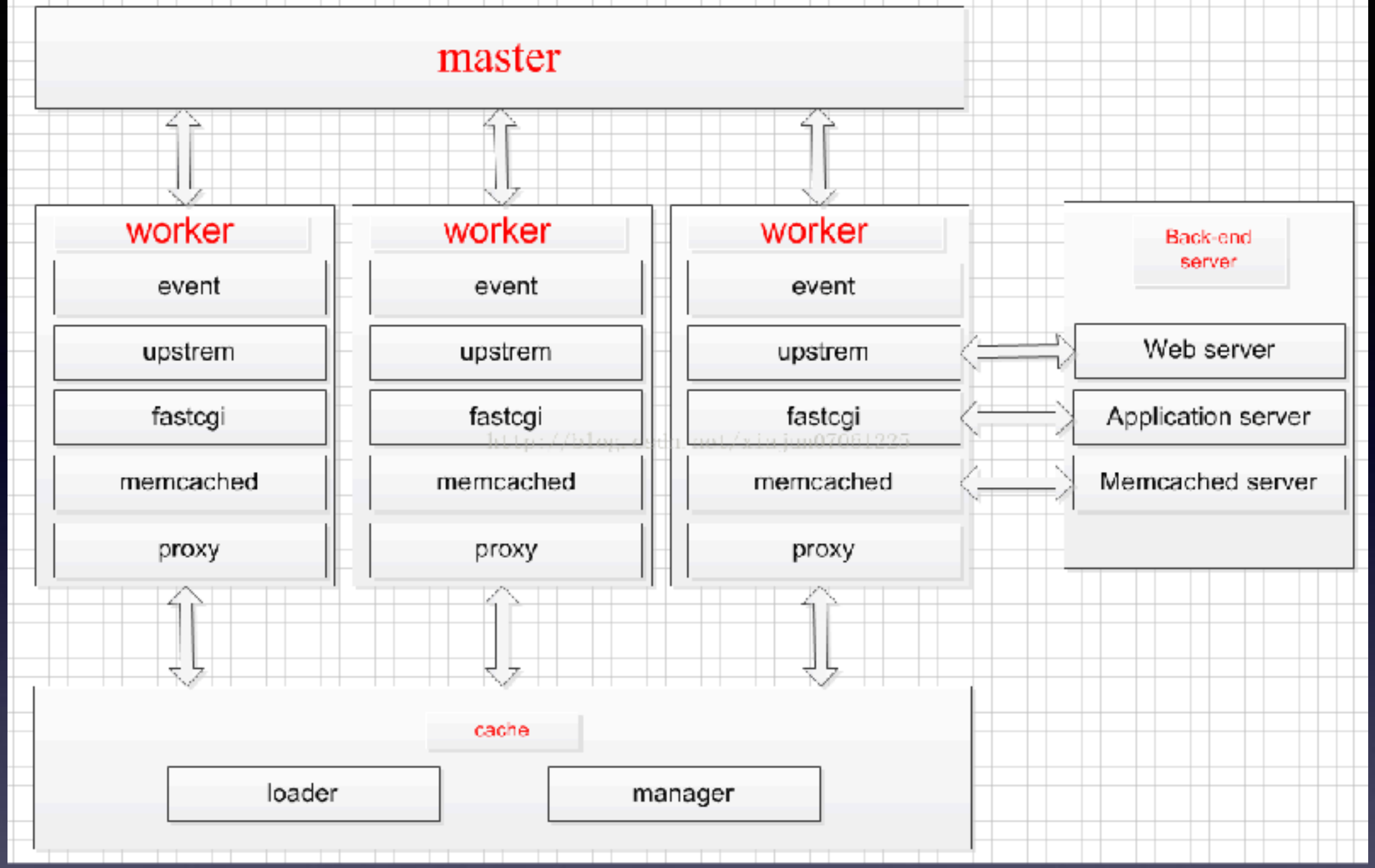
- auto 目录: 安装中使用
- src　Nginx核心代码
- src/core： 包含了Nginx的最基础的库和框架。包括了内存池、链表、hashmap、String等常用的**数据结构**
- src/event：事件模块。Nginx自己实现了事件模型。而我们所熟悉的Memcached是使用了Libevent的事件库。自己实现event会性能和效率方便更加高效。
- src/http: 实现HTTP的模块。实现了HTTP的具体协议的各种模块，该部分内容量比较大。



- Nginx是一款多进程的软件。Nginx启动后，会产生一个master进程和N个工作进程。其中nginx.conf中可以配置工作进程的个数：worker_processes 1

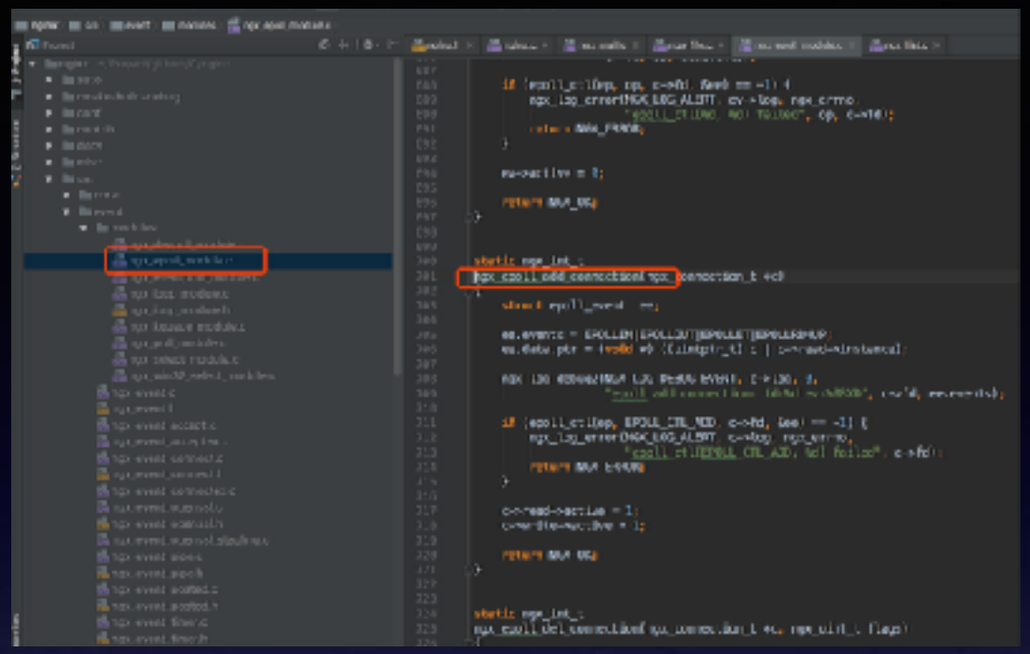
- 多进程的好处: 不需要太多考虑并发锁的问题

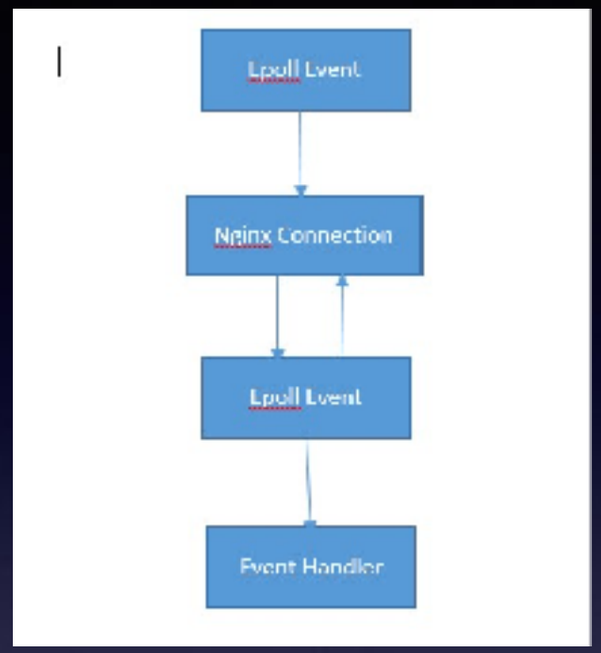
- 我们常用的Memcached和Nginx相反 它是典型的多线程模型软件


master
worker
event
upstrem
fastcgi
memcached
proxy
worker
event
upstrem
fastcgi
memcached
proxy
worker
event
upstrem
fastcgi
memcached
proxy
Back-end server
Web server
Application server
Memcached server
cache
loader
manager

Nginx进程模型

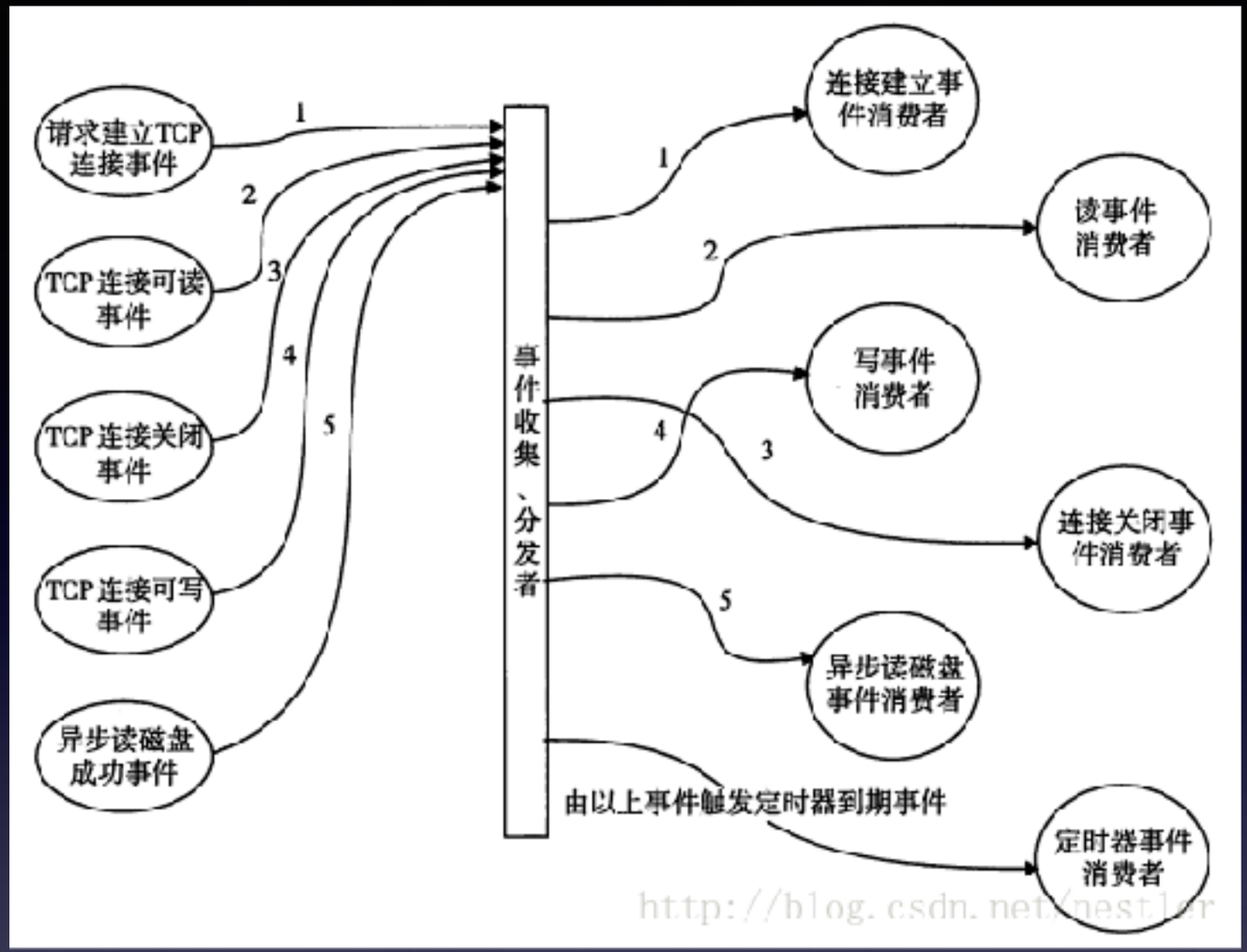
Nginx事件驱动





1.传统的WebServer下:
一个请求是由一个进程消费,请求在建立连接后始终占用着资源,直到关闭连接才会释放资源
这样做的缺点:
进程数添加会有进程切换,影响总体性能
当某个进程要等待事件而处于拥堵状态,该进程依然占用资源,直到连接结束,造成资源极大浪费
(一句话总结: 粒度大 不好控制)

2.Nginx的处理方式:
接受到一个请求时,不会产生一个单独的进程来处理该请求,而是由事件收集,分发器(进程),调用某个模块,由模块处理请求.处理完之后再返回到事件收集器,分发器



如图所示:

左侧按序号收集事件

右侧按序号调用消费者模块 (各司其职)

一个HTTP请求包括多个阶段:

每一个阶段的发生都会出发事件驱动框架,然后交由事件消费者处理

Nginx采用这种设计:

减少了进程休眠几率 提高网络性能 减少请求延迟

可以轻松应对高并发

Nginx基础库

- 1.内存池
- 2. ngx_str_t
- 3. ngx_array_t
- 4.ngx_hash_t
- 5.ngx_list_t
- 6.ngx_queue_t
- 红黑树
- Spinlock自旋锁
- ngx_shmtx_lock 多进程加锁

```
} ngx_pool_data_t;


struct ngx_pool_s {
    ngx_pool_data_t       d;
    size_t                max;
    ngx_pool_t           *current;
    ngx_chain_t          *chain;
    ngx_pool_large_t     *large;
    ngx_pool_cleanup_t   *cleanup;
    ngx_log_t            *log;
};
```

```
typedef struct {
    void        *elts;
    ngx_uint_t   nelts;
    size_t       size;
    ngx_uint_t   nalloc;
    ngx_pool_t  *pool;
} ngx_array_t;
```

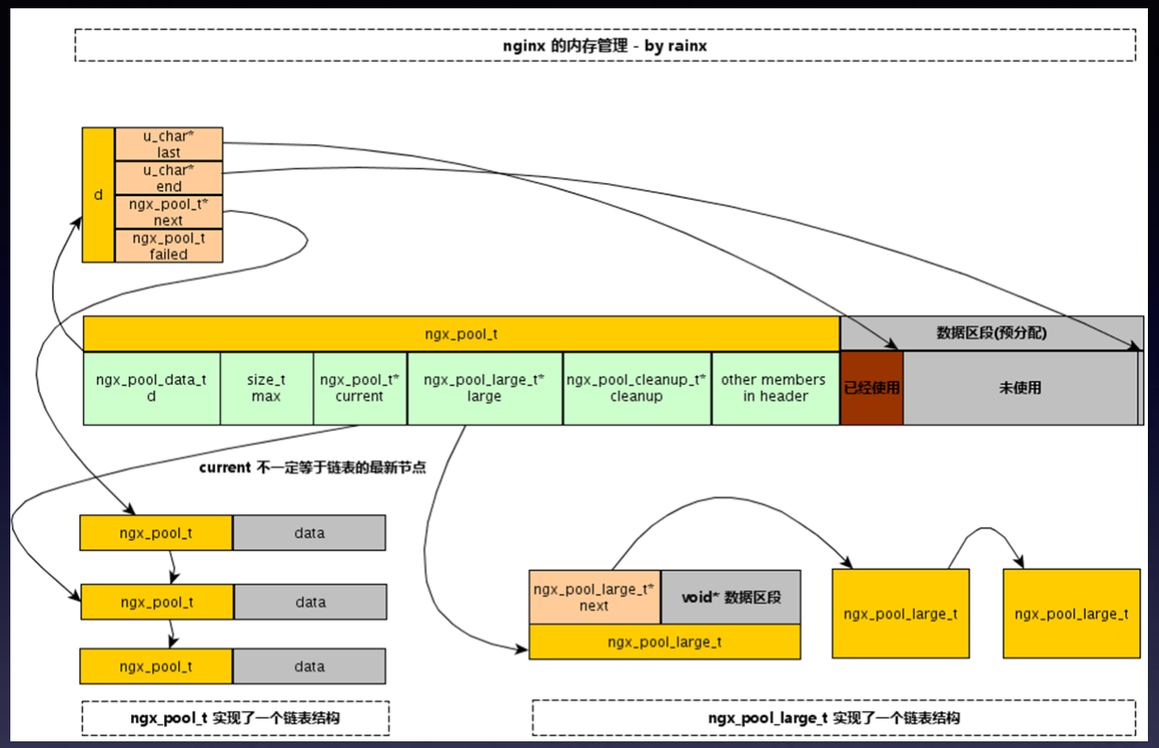
```
typedef struct {
    size_t      len;
    u_char     *data;
} ngx_str_t;
```

内存池:

一般我们使用malloc/alloc/free等函数来分配和释放内存。但是直接使用这些函数会有一些弊端:

1.频繁使用这些函数分配和释放内存，会导致内存碎片，不容易让系统直接回收内存。典型的例子就是大并发频繁分配和回收内存，会导致进程的内存产生碎片，并且不会立马被系统回收。

2.容易产生内存泄漏



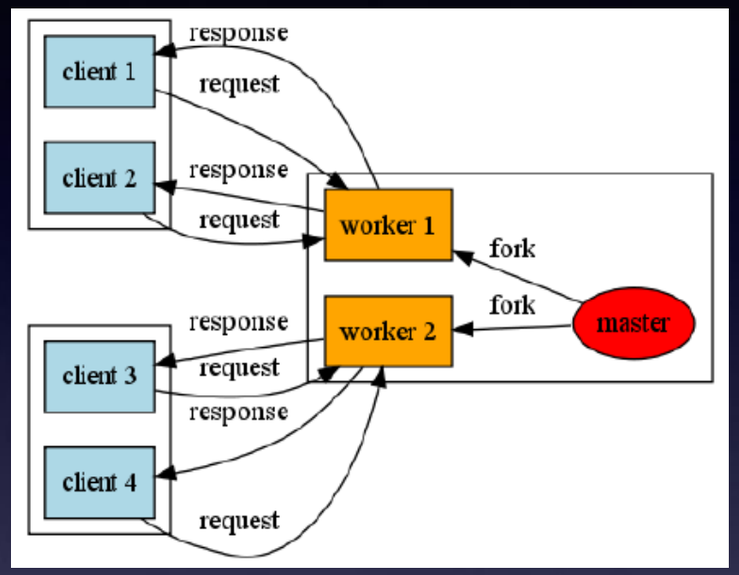
Nginx使用内存池的好处

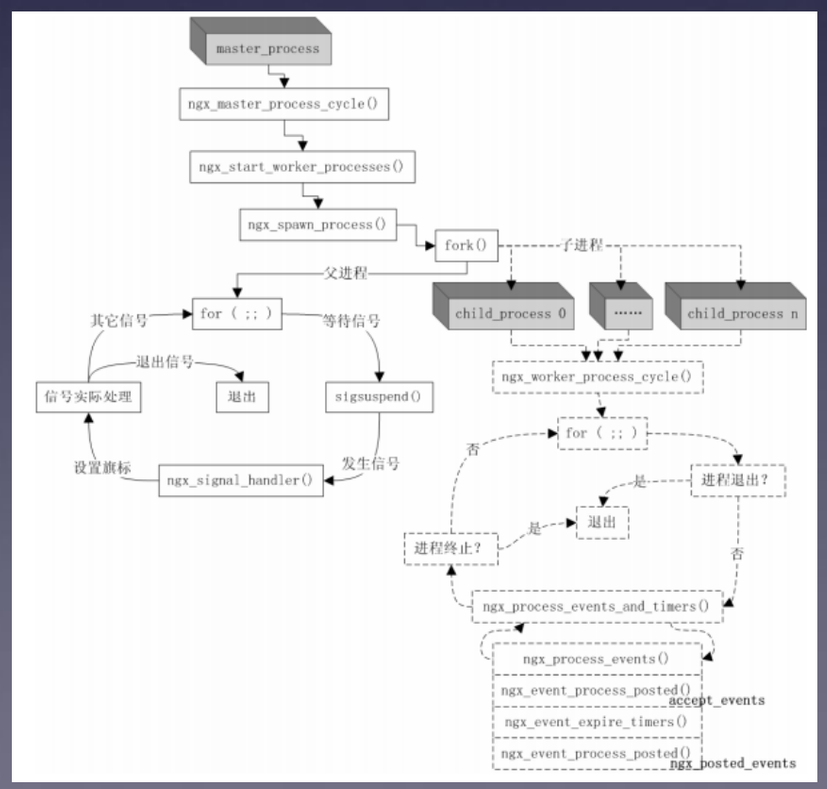
1. 提示内存分配效率,不用每次分配内存都执行malloc/alloc
2. 简化内存管理,先申请一块大的内存,回收的时间直接回收大块内存就能回收所有内存,

防止了内存管理混乱和内存泄漏问题

```
57
58  /**
59   * Nginx 内存池数据结构
60   */
61  struct ngx_pool_s {
62      ngx_pool_data_t       d;       /* 内存池的数据区域*/
63      size_t                max;     /* 最大每次可分配内存 */
64      ngx_pool_t           *current; /* 指向当前的内存池指针地址。ngx_pool_t链表上最后一个储存池结构*/
65      ngx_chain_t          *chain;   /* 缓冲区链表 */
66      ngx_pool_large_t     *large;   /* 存储大数据的链表 */
67      ngx_pool_cleanup_t   *cleanup; /* 可自定义回调函数，清除内存块分配的内存 */
68      ngx_log_t            *log;     /* 日志 */
69  };
70
71
```

Nginx总体架构

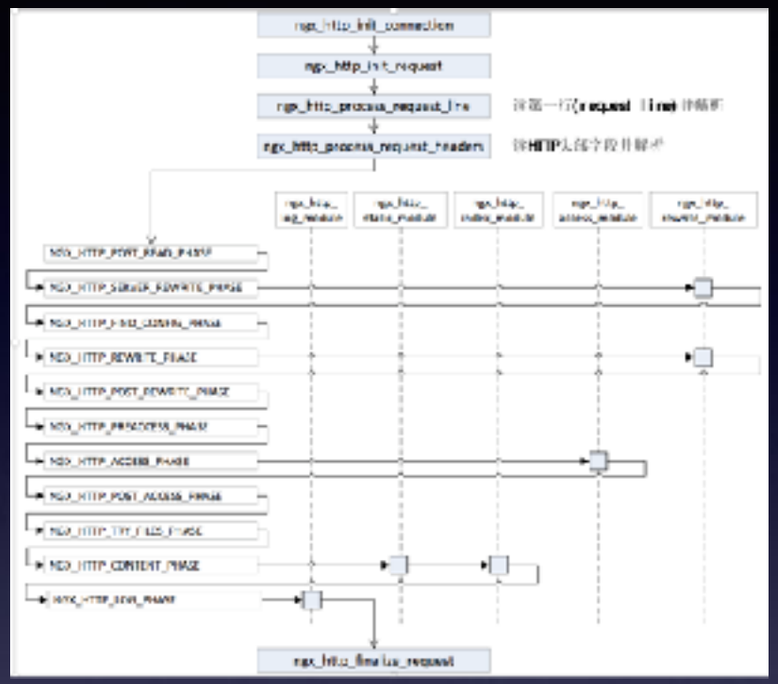


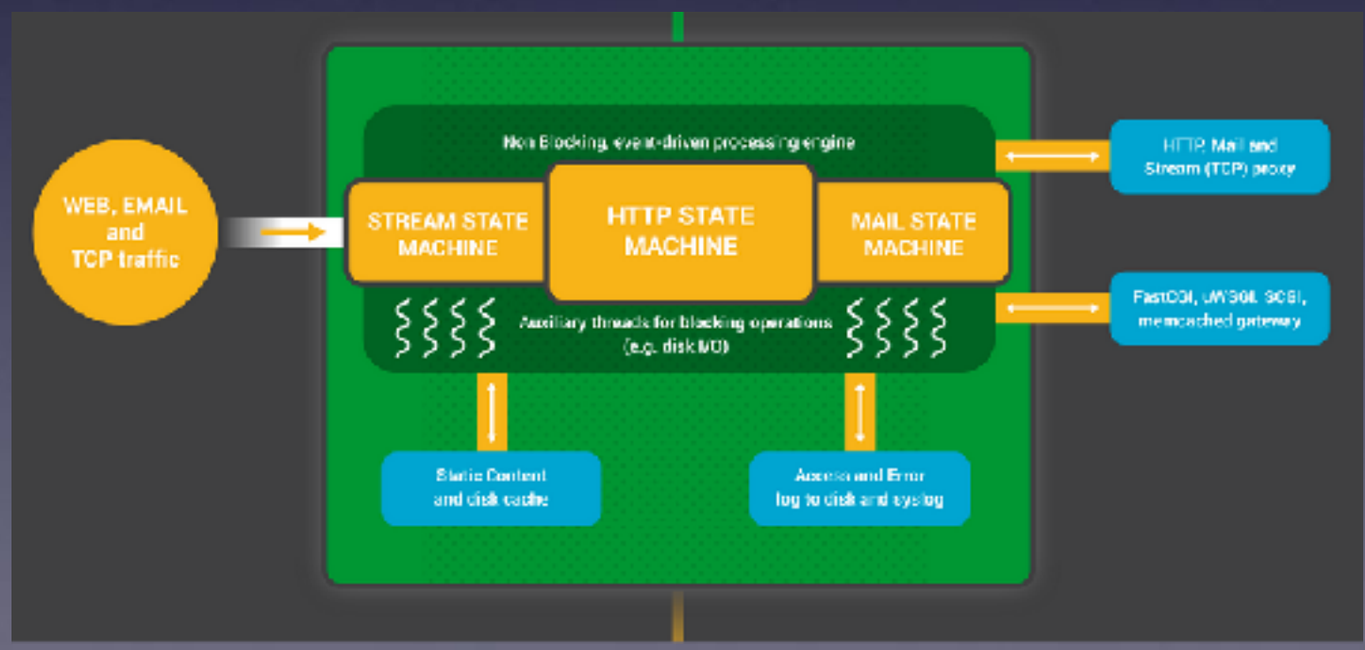


- 1. Master && worker模型
- 2. 事件驱动
- 3. 高度模块化
- 4.独立的Cache管理进程
- 平滑重启

Nginx状态机

- 用状态机来管理每个连接的状态
- 1.预处理
- 2.状态机
- 3.Header Filter链
- 4. Content Filter链





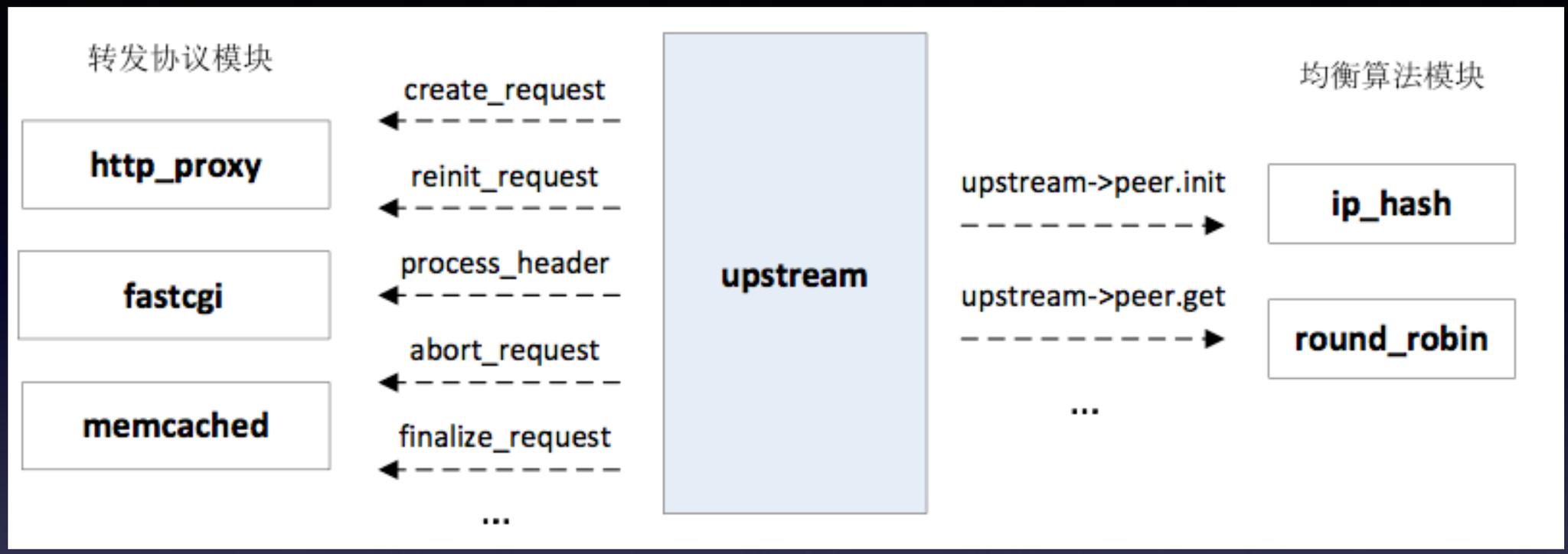
- NGX_HTTP_POST_READ_PHASE。realip模块。
- NGX_HTTP_SERVER_REWRITE_PHASE。URL的转换。Rewrite模块
- NGX_HTTP_FIND_CONFIG_PHASE。找到对应的location配置文件。 不能扩展。
- NGX_HTTP_REWRITE_PHASE。在location级别进行URL的转换。Rewite模块
- NGX_HTTP_POST_REWRITE_PHASE。在post-processing阶段进行请求URL的转换
- NGX_HTTP_PREACCESS_PHASE。在preprocessing阶段进行ACCESS检查。limit_req、limit_zone、degradation、realip模块
- NGX_HTTP_ACCESS_PHASE。Access检查。对应有ACCESS 、auth_basic模块。
- NGX_HTTP_POST_ACCESS_PHASE。在post-processing阶段进行Access检查。
- NGX_HTTP_TRY_FILES_PHASE。Try_files指令处理阶段。
- NGX_HTTP_CONTENT_PHASE。产生回复内容的阶段。Autoindex、static、random_index、gzip_static等。
- NGX_HTTP_LOG_PHASE。打日志的阶段。

Nginx Handler模块

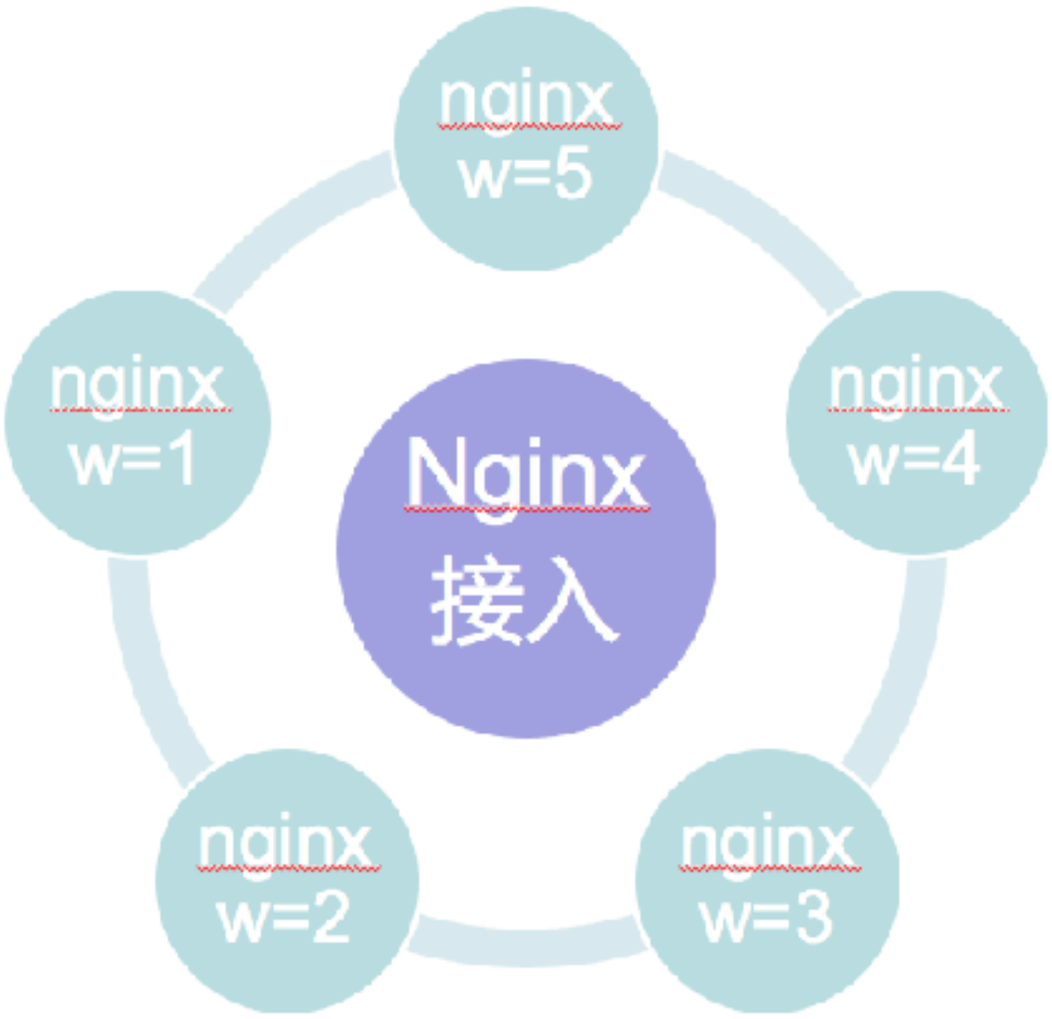
- Header Filter
- Content Filter
- 单链表
- 可扩展可插入
- 优化点：Buff实现流式Filter。
- 比如：gzip、 more_set_header

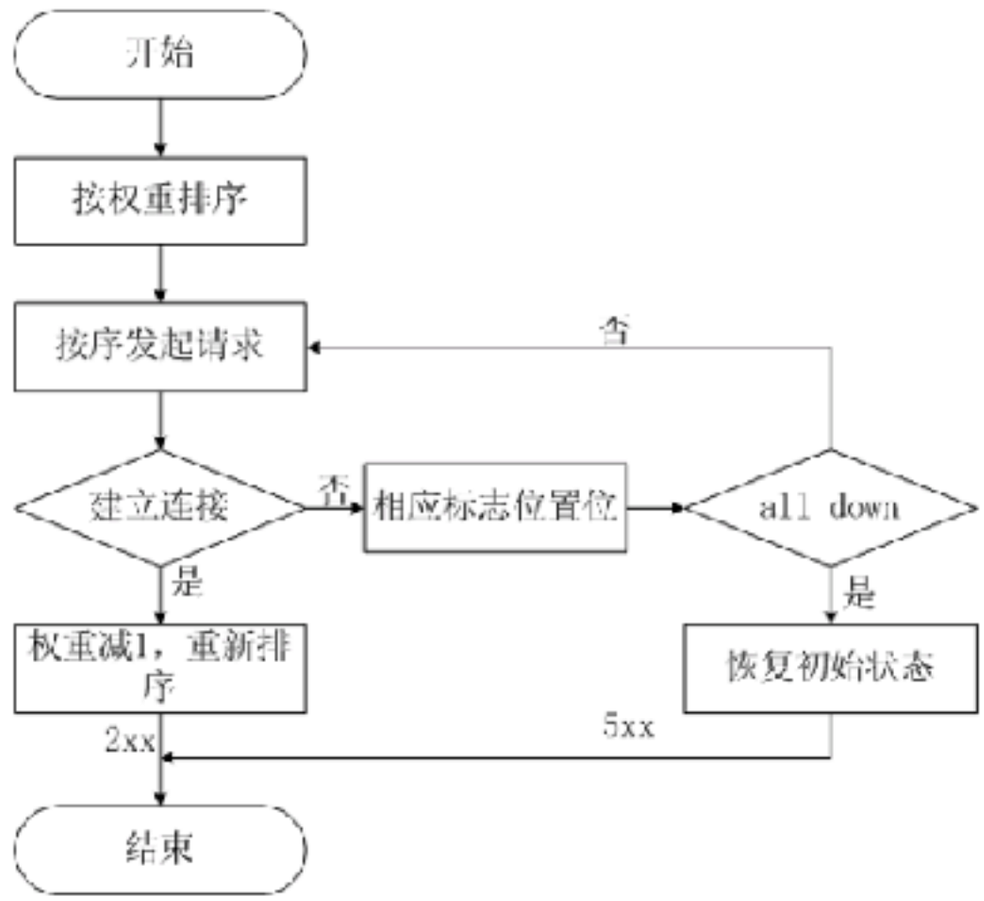
Nginx Filter模块

Nginx Upstream



- 多样化的后端模块(交互协议)
  HTTP Fastcig Memcache SCGI (灵活)
- 不同的负载均衡算法 hash/random/轮询/加权轮询(丰富)
- 可扩展 可插拔(健壮性)





Upstream负载均衡:加权轮询策略

Upstream负载均衡: ip_hash策略

- 1. 每个请求按照访问ip的hash结果分配,这样可以每个访客固定到一个后端服务器,可以解决SEESION问题

- 2. upstream bakend {
  ip_hash;
  server 192.168.0.14:88;
  server 192.168.0.15:80;
  }

其他负载均衡配置方式

- fair（公平的 第三方）
  fair策略是扩展策略，默认不被编译进nginx内核。其原理是根据后端服务器的响应时间判断负载情况，从中选出负载最轻的机器进行分流。这种策略具有很强的自适应性，但是实际的网络环境往往不是那么简单，因此要慎用。

- 通用Hash　一致性Hash
  内置的变量为key进行hash，一致性hash采用了nginx内置的一致性hash环，可以支持memcache。

负载均衡方式总结

	均衡性	一致性	容灾性	适用场景
轮询	★★★	★	★★★	通用性强，无特殊需求的场景下均可
fair	★★	★	★★★	自适应性强，在网络环境复杂的情况下表现较好
ip hash	★★	★★★	★★★	要求 ip 请求一致性的场景
其他 hash	★★★	★★★	★★	请求一致性，如缓存服务器等

- 对上面的集中负载均衡算法进行测试（测试工具polygraph），
  考察下面三个关键的测试指标：
  均衡性：是否能够将请求均匀的发送给后端
  一致性：同一个key的请求，是否能落到同一台机器

无论哪种策略都不是万金油,要因地制宜

Nginx高性能的秘密

- 1. 为高性能而生 (自己实现的event 内存池 红黑树)
- 2. 事件驱动 异步 非阻塞　高度模块化 低耦合
- 3. 使用Linux的epoll kqueue

参考文档

- 1. http://nginx.org/en/docs/
- 2. http://blog.csdn.net/initphp
- 3. http://www.ttlsa.com/