[第一章 .Nginx 安装 2](#_Toc70546669)

[第二章 .Nginx简介 3](#_Toc70546670)

[第三章 .Nginx源码分析 4](#_Toc70546671)

[3.1 Nginx启动过程 4](#_Toc70546672)

[3.2 Nginx字符串 6](#_Toc70546673)

[3.3 Nginx数组 6](#_Toc70546674)

[3.4 Nginx单向链表 6](#_Toc70546675)

[3.5 Nginx双向链表 6](#_Toc70546676)

[3.6 Nginx内存池 6](#_Toc70546677)

[3.7 Nginx模块初始化 6](#_Toc70546678)

[3.8 Nginx解析配置文件 6](#_Toc70546679)

[3.9 Nginx多进程 6](#_Toc70546680)

[3.10 Nginx Event模块的初始化 6](#_Toc70546681)

[3.11 Nginx Epoll事件模块 6](#_Toc70546682)

[3.12 Nginx Http模块初始化 6](#_Toc70546683)

[3.13 Nginx Tcp建立过程 6](#_Toc70546684)

[3.14 Nginx事件处理 6](#_Toc70546685)

[第四章 http处理请求 7](#_Toc70546686)

[4.1.1 1-2.7 Nginx反向代理负载均衡配置详解，Nginx子域名映射 8](#_Toc70546687)

[4.1.2 1-2.8 Restful Http第三方接口实现，异步Http请求 12](#_Toc70546688)

[4.1.3 Nginx内存池 12](#_Toc70546689)

[1. Nginx 安装 16](#_Toc70546690)

**源码分析专题**

# Nginx 安装

使用下面的安装方法可以部署gdb调试用版本

准备第三方支持库源码：

λ nginx-1.13.7.tar.gz

λ openssl-1.1.0g.tar.gz

λ pcre-8.41.tar.gz

λ zlib-1.2.11.tar.gz

解压每个包 公网部署

wget <http://nginx.org/download/nginx-1.13.7.tar.gz>

wget <https://www.openssl.org/source/openssl-1.1.0g.tar.gz>

wget <http://ftp.pcre.org/pub/pcre/pcre-8.41.tar.gz>

wget <http://www.zlib.net/zlib-1.2.11.tar.gz>

$ tar xzvf nginx-1.13.7.tar.gz

$ tar xzvf openssl-1.1.0g.tar.gz

$ tar xzvf pcre-8.41.tar.gz

$ tar xzvf zlib-1.2.11.tar.gz

$ cd nginx-1.13.7

./configure --prefix=/usr/local/nginx/ --with-http\_realip\_module --with-http\_addition\_module --with-http\_gzip\_static\_module --with-http\_secure\_link\_module --with-http\_stub\_status\_module --with-stream --with-pcre=/home/alpha/share/nginx/pcre-8.41 --with-zlib=/home/alpha/share/nginx/zlib-1.2.11 --with-openssl=/home/alpha/share/nginx/openssl-1.1.0g --with-cc-opt='-g -o0'

$ make CFLAGS="-g -oO"

$ sudo make install

在/usr/local/目录下面， 产生了 nginx 的目录

$ ./sbin/nginx –c ./conf/nginx.conf

使用上述方法编译部署的版本可以gdb跟踪

#启动nginx：

sudo /usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf

注意：-c 指定配置文件的路径，不加的话，nginx会自动加载默认路径的配置文件，可以通过 -h查看帮助命令。

#查看nginx进程：

ps -ef|grep nginx

启动 Nginx

/usr/local/nginx/sbin/nginx

./sbin/nginx

停止 Nginx

./sbin/nginx -s stop

./sbin/nginx -s quit

-s都是采用向 Nginx 发送信号的方式。

Nginx重新加载配置

./sbin/nginx -s reload

指定配置文件

./sbin/nginx -c /usr/local/nginx/conf/nginx.conf

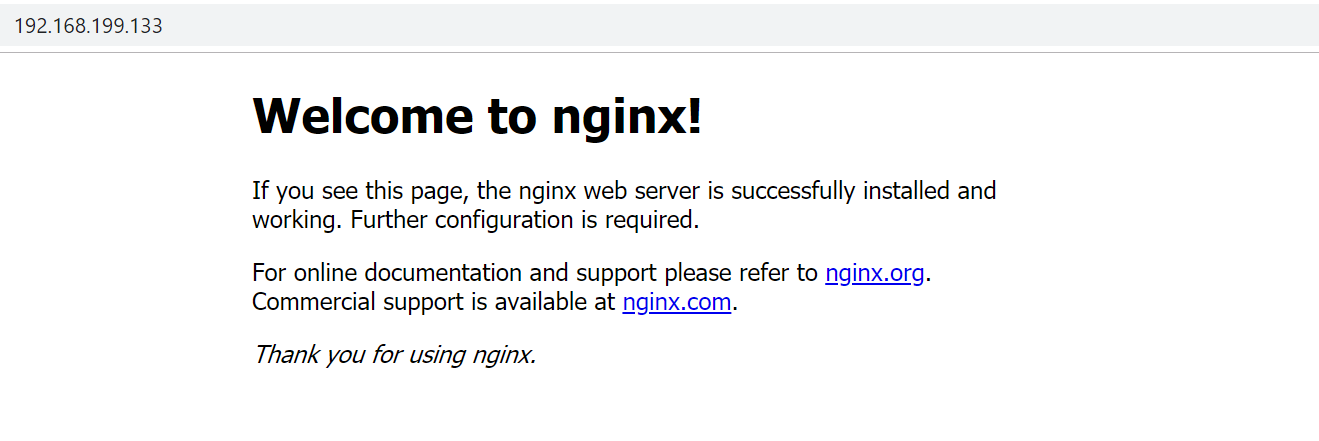
-c表示configuration，指定配置文件

查看 Nginx 版本

有两种可以查看 Nginx 的版本信息的参数。第一种如下：

./sbin/nginx –v

访问效果如下：



nginx源码编译，依赖于pcre和zlib

下载源码后，通过./configure --help可以查看编译选项，为了增加gdb调试信息，使用./configure --with-cc-opt='-g -o0'和make CFLAGS="-g -o0"

为了方便调试可以启动一个worker进程

nginx对gdb也有辅助支持，用debug\_points配置项：stop或者abort，表示在遇到严重错误的时候，

ngingx的进程可以stop住。如果是abort，可以设置生成core文件。

关于宏，宏默认是直接替换的，如果在编译的时候，将-g 改为--ggdb3可以保留宏，但是增大了可执行二进制文件。info macro NGX\_OK/ macro expand NGX\_OK/p NGX\_OK。如果info 没有找到，可以list文件加载一下后，再去执行info命令。

编译的时候可以增加--with-debug，以便生成调试信息。

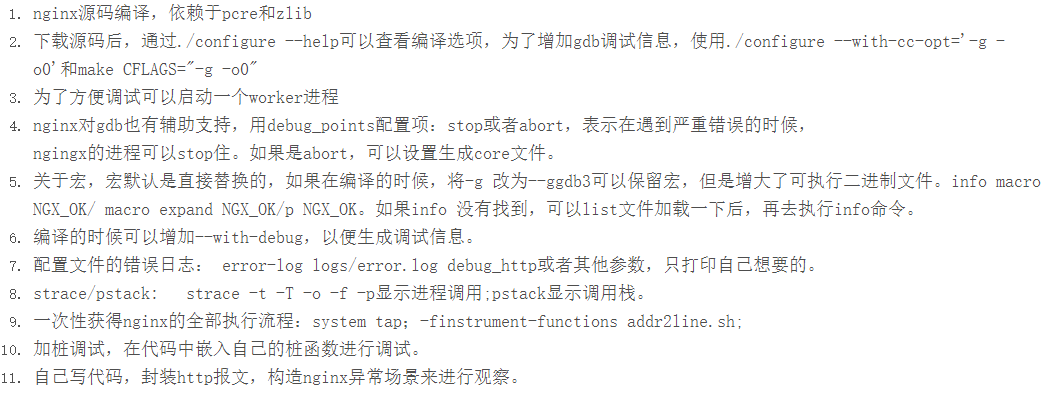
配置文件的错误日志： error-log logs/error.log debug\_http或者其他参数，只打印自己想要的。

strace/pstack:  strace -t -T -o -f -p显示进程调用;pstack显示调用栈。

一次性获得nginx的全部执行流程：system tap；-finstrument-functions addr2line.sh;

加桩调试，在代码中嵌入自己的桩函数进行调试。

自己写代码，封装http报文，构造nginx异常场景来进行观察。



# Nginx简介

Nginx是 HTTP 和反向代理服务器，邮件代理服务器，以及 Igor Sysoev 最初编写的通用TCP/UDP 代理服务器。在很长一段时间以来，它一直在许多负载重的俄罗斯网站上运行，包括： Yandex，Mail.Ru，VK 和 Rambler。根据Netcraft的说法，Nginx在 2017 年3 月份服务或代理了28.50％的最繁忙的网站。这里有一些成功案例：Netflix，Wordpress.com，FastMail.FM。

Nginx特性

Nginx有什么不同？Nginx使用可扩展的事件驱动架构，而不是更传统的过程驱动架构。这需要更低的内存占用，并且当并发连接扩大时，使内存使用更可预测。在传统的 Web 服务器体系结构中，每个客户端连接作为一个单独的进程或线程处理，随着网站的流行度增加，并发连接数量的增加，Web服务器减慢，延迟了对用户的响应。从技术的角度来看，产生一个单独的进程/线程需要将CPU切换到新的任务并创建一个新的运行时上下文，消耗额外的内存和CPU时间，从而对性能产生负面影响。Nginx开发的目标是实现10倍以上的性能，优化服务器资源的使用，同时也能够扩展和支持网站的动态增长。因此，Nginx成为最知名的模块化，事件驱动，异步，单线程 Web 服务器和 Web 代理之一。Nginx是一个高性能的 Web 和反向代理服务器, 它具有很多非常优越的特性:

作为 **Web** 服务器

相比 Apache， Nginx使用更少的资源，支持更多的并发连接，体现更高的效率，这点使 Nginx尤其受到虚拟主机提供商的欢迎。能够支持高达 50,000 个并发连接数的响应，感谢 Nginx为我们选择了 epoll and kqueue 作为开发模型

作为负载均衡服务器

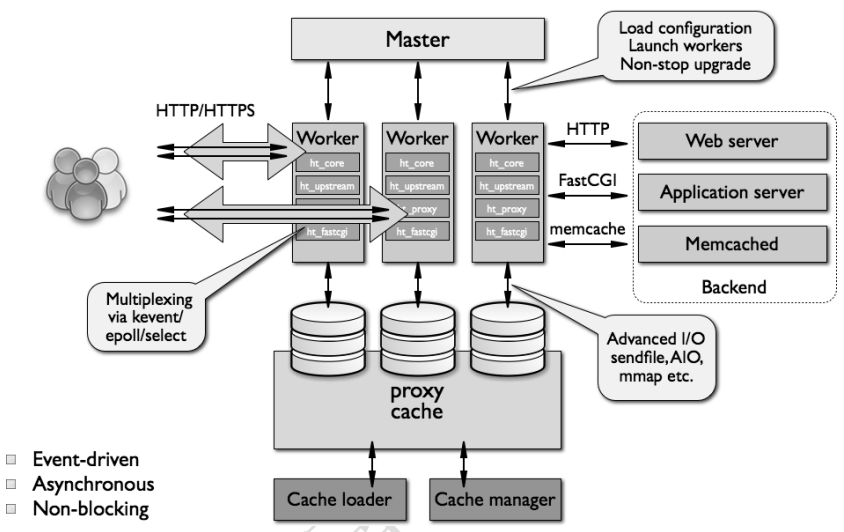
Nginx既可以在内部直接支持 Rails 和 PHP，也可以支持作为 HTTP 代理服务器 对外进行服务。Nginx用 C 编写, 不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好的多。

Nginx架构

处理并发连接的传统的基于进程或线程的模型涉及使用单独的进程或线程处理每个连接，并阻止网络或输入/输出操作。

根据应用，在内存和 CPU 消耗方面可能非常低效。产生一个单独的进程或线程需要准备一个新的运行时环境，包括分配堆和堆栈内存，以及创建新的执行上下文。额外的CPU时间也用于创建这些项目，这可能会导致由于线程在过多的上下文切换上的转机而导致性能下降。所有这些并发症都表现在较老的 Web 服务器架构(如Apache)中。 这是提供丰富的一般应用功能和优化的服务器资源使用之间的一个折衷。

从一开始 Nginx就是一个专门的工具，可以实现更高性能，更密集和经济地使用服务器资源，同时实现网站的动态发展，所以它采用了不同的模式。它实际上受到各种操作系统中高级事件机制的不断发展的启发。发展结果变成是一个模块化的，事件驱动的，异步的，单线程的非阻塞架构的 nginx 代码基础。nginx大量使用复用和事件通知，并专门用于分离进程的特定任务。连接在有限数量的单线程进程称为工作(worker)的高效运行循环中处理。 在每个工作(worker)中， nginx可以处理每秒数千个并发连接和请求



# Nginx源码分析

## Nginx启动过程

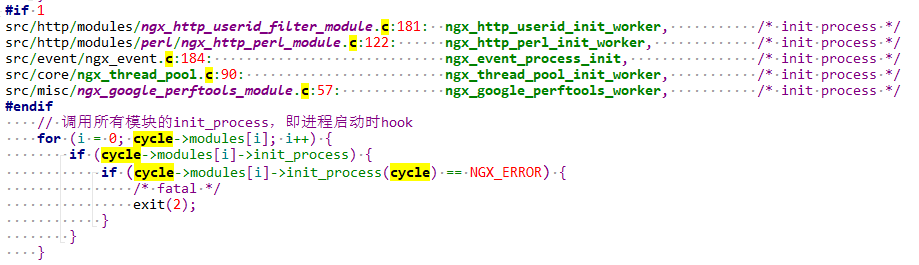
Nginx的主流程的实现函数在./src/core/nginx.c文件中。通过main()函数，我们可以窥探整个Nginx启动的流程

main()函数的启动过程如下：

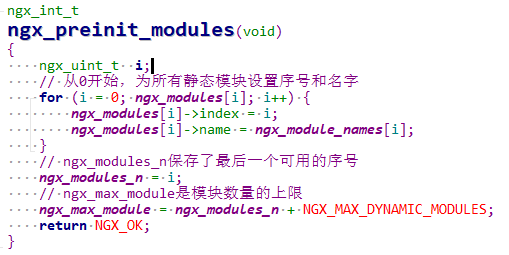
1. 调用ngx\_get\_options方法，主要用于解析命令行中的参数，例如：./nginx -s stop|start|restart
2. 调用ngx\_time\_init方法，初始化并更新时间，如全局变量ngx\_cached\_time
3. 调用ngx\_getpid方法，获取当前进程的pid。一般pid会放在/usr/local/nginx-1.4.7/nginx.pid的文件中，用于发送重启，关闭等信号命令。
4. 调用ngx\_log\_init方法，初始化日志，并得到日志的文件句柄ngx\_log\_file.fd
5. 初始化init\_cycle Nginx的全局变量。在内存池上创建一个默认大小1024的全局变量。这里只是最简单的初始化一个变量。
6. 调用ngx\_save\_argv方法，保存Nginx命令行中的参数和变量,放到全局变量ngx\_argv
7. 调用ngx\_process\_options方法，将ngx\_get\_options中获得这些参数取值赋值到ngx\_cycle中。prefix, conf\_prefix, conf\_file, conf\_param等字段。
8. 调用ngx\_os\_init()初始化系统相关变量，如内存页面大小ngx\_pagesize,ngx\_cacheline\_size,最大连接数ngx\_max\_sockets等
9. 调用ngx\_crc32\_table\_init方法，初始化一致性hash表，主要作用是加快查询
10. 调用ngx\_add\_inherited\_sockets方法，ngx\_add\_inherited\_sockets主要是继承了socket的套接字。主要作用是热启动的时候需要平滑过渡
11. 调用ngx\_preinit\_modules方法，主要是前置的初始化模块，对模块进行编号处理
12. 调用ngx\_init\_cycle方法，完成全局变量cycle的初始化
13. 调用ngx\_signal\_process方法，如果有信号，则进入ngx\_signal\_process方法。例如：例如./nginx -s stop,则处理Nginx的停止信号
14. 调用ngx\_get\_conf方法，得到核心模块ngx\_core\_conf\_t的配置文件指针
15. 调用ngx\_create\_pidfile方法，创建pid文件。例如：/usr/local/nginx-1.4.7/nginx.pid
16. 调用ngx\_master\_process\_cycle方法，这函数里面开始真正创建多个Nginx的子进程。**这个方法包括子进程创建、事件监听、各种模块运行等都会包含进去**



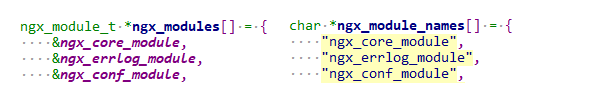
1. 模块init\_process中仅搜到如下：



1. ngx\_preinit\_modules 给所有支持的模块编号及名字

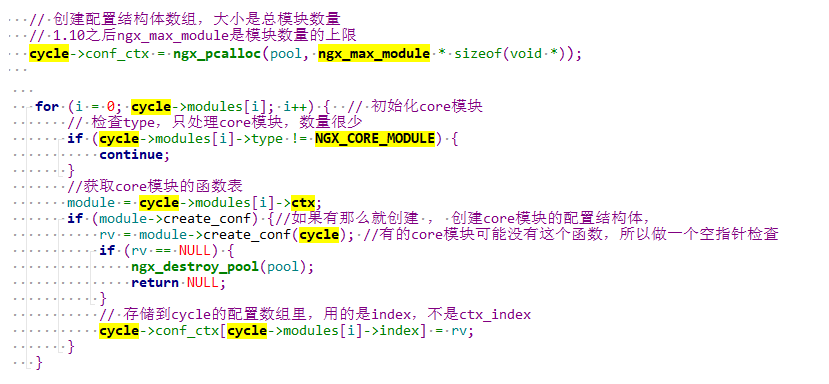


1. 在obj目录下Nginx-1.13.7\objs\ngx\_modules.c 会生成一个模块名字和模块表



1. 调用ngx\_get\_conf方法，得到核心模块ngx\_core\_conf\_t的配置文件指针

核心模块比较少，只有少部分有，会存到下面的配置数组中，在cycle全局变量中



## Nginx字符串

## Nginx数组

## Nginx单向链表

## Nginx双向链表

## Nginx内存池

## Nginx模块初始化

## Nginx解析配置文件

## Nginx多进程

## Nginx Event模块的初始化

## Nginx Epoll事件模块

## Nginx Http模块初始化

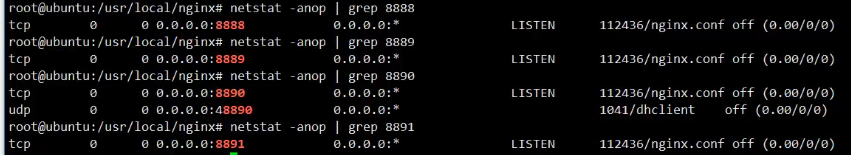
## Nginx Tcp建立过程

## Nginx事件处理

# http处理请求

先listen 在fork，那么子进程都会共享父进程的fd，此时子进程便都可以进行监听

每个进程都监听所有的端口，当连接出现时，服务器使用哪个进程来连接呢？ 惊群问题？

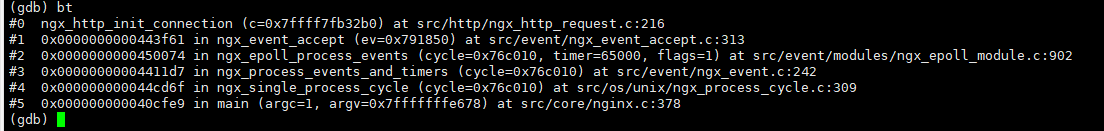


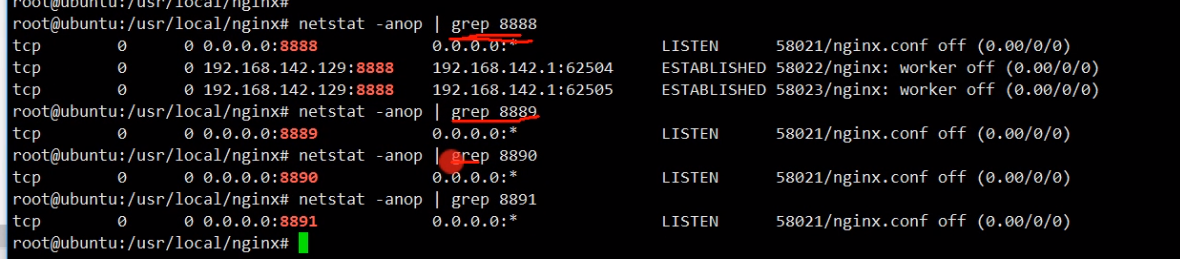
strace ./a.out

条件变量惊群 ； accept 惊群 已被解决； epoll\_wait惊群

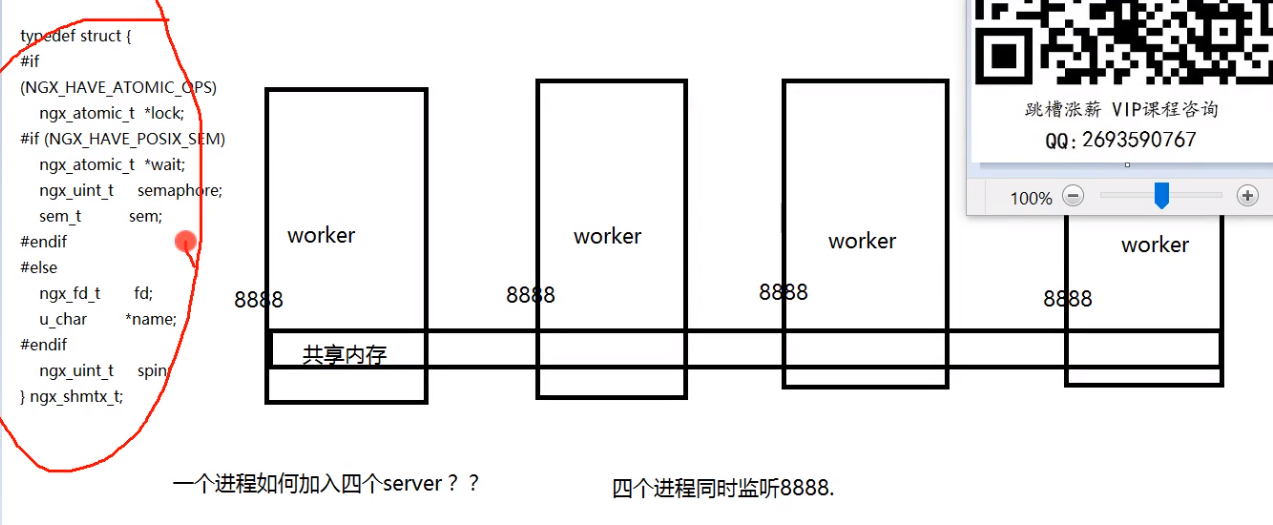
哪个进程获取到锁，哪个进程去执行epoll\_wait

lock process\_event unlock









## Nginx反向代理负载均衡配置详解，Nginx子域名映射

### Nginx 介绍

Nginx是 HTTP 和反向代理服务器，邮件代理服务器，以及 Igor Sysoev 最初编写的通用TCP/UDP 代理服务器。在很长一段时间以来，它一直在许多负载重的俄罗斯网站上运行，包括： Yandex，Mail.Ru，VK 和 Rambler。根据Netcraft的说法，Nginx在 2017 年3 月份服务或代理了28.50％的最繁忙的网站。这里有一些成功案例：Netflix，Wordpress.com，FastMail.FM。

Nginx特性

Nginx有什么不同？Nginx使用可扩展的事件驱动架构，而不是更传统的过程驱动架构。这需要更低的内存占用，并且当并发连接扩大时，使内存使用更可预测。在传统的 Web 服务器体系结构中，每个客户端连接作为一个单独的进程或线程处理，随着网站的流行度增加，并发连接数量的增加，Web服务器减慢，延迟了对用户的响应。从技术的角度来看，产生一个单独的进程/线程需要将CPU切换到新的任务并创建一个新的运行时上下文，消耗额外的内存和CPU时间，从而对性能产生负面影响。Nginx开发的目标是实现10倍以上的性能，优化服务器资源的使用，同时也能够扩展和支持网站的动态增长。因此，Nginx成为最知名的模块化，事件驱动，异步，单线程 Web 服务器和 Web 代理之一。Nginx是一个高性能的 Web 和反向代理服务器, 它具有很多非常优越的特性:

作为 **Web** 服务器

相比 Apache， Nginx使用更少的资源，支持更多的并发连接，体现更高的效率，这点使 Nginx尤其受到虚拟主机提供商的欢迎。能够支持高达 50,000 个并发连接数的响应，感谢 Nginx为我们选择了 epoll and kqueue 作为开发模型

作为负载均衡服务器

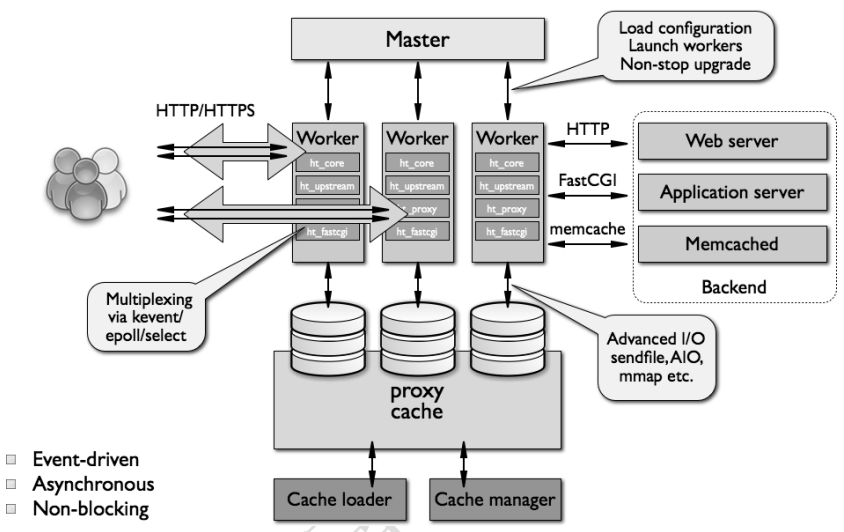
Nginx既可以在内部直接支持 Rails 和 PHP，也可以支持作为 HTTP 代理服务器 对外进行服务。Nginx用 C 编写, 不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好的多。

Nginx架构

处理并发连接的传统的基于进程或线程的模型涉及使用单独的进程或线程处理每个连接，并阻止网络或输入/输出操作。

根据应用，在内存和 CPU 消耗方面可能非常低效。产生一个单独的进程或线程需要准备一个新的运行时环境，包括分配堆和堆栈内存，以及创建新的执行上下文。额外的CPU时间也用于创建这些项目，这可能会导致由于线程在过多的上下文切换上的转机而导致性能下降。所有这些并发症都表现在较老的 Web 服务器架构(如Apache)中。 这是提供丰富的一般应用功能和优化的服务器资源使用之间的一个折衷。

从一开始 Nginx就是一个专门的工具，可以实现更高性能，更密集和经济地使用服务器资源，同时实现网站的动态发展，所以它采用了不同的模式。它实际上受到各种操作系统中高级事件机制的不断发展的启发。发展结果变成是一个模块化的，事件驱动的，异步的，单线程的非阻塞架构的 nginx 代码基础。nginx大量使用复用和事件通知，并专门用于分离进程的特定任务。连接在有限数量的单线程进程称为工作(worker)的高效运行循环中处理。 在每个工作(worker)中， nginx可以处理每秒数千个并发连接和请求



### Nginx 安装

准备第三方支持库源码：

nginx-1.13.7.tar.gz

openssl-1.1.0g.tar.gz

pcre-8.41.tar.gz

zlib-1.2.11.tar.gz

解压每个包

wget <http://nginx.org/download/nginx-1.13.7.tar.gz>

wget <https://www.openssl.org/source/openssl-1.1.0g.tar.gz>

wget <http://ftp.pcre.org/pub/pcre/pcre-8.41.tar.gz>

wget <http://www.zlib.net/zlib-1.2.11.tar.gz>

$ tar xzvf nginx-1.13.7.tar.gz

$ tar xzvf openssl-1.1.0g.tar.gz

$ tar xzvf pcre-8.41.tar.gz

$ tar xzvf zlib-1.2.11.tar.gz

$ cd nginx-1.13.7

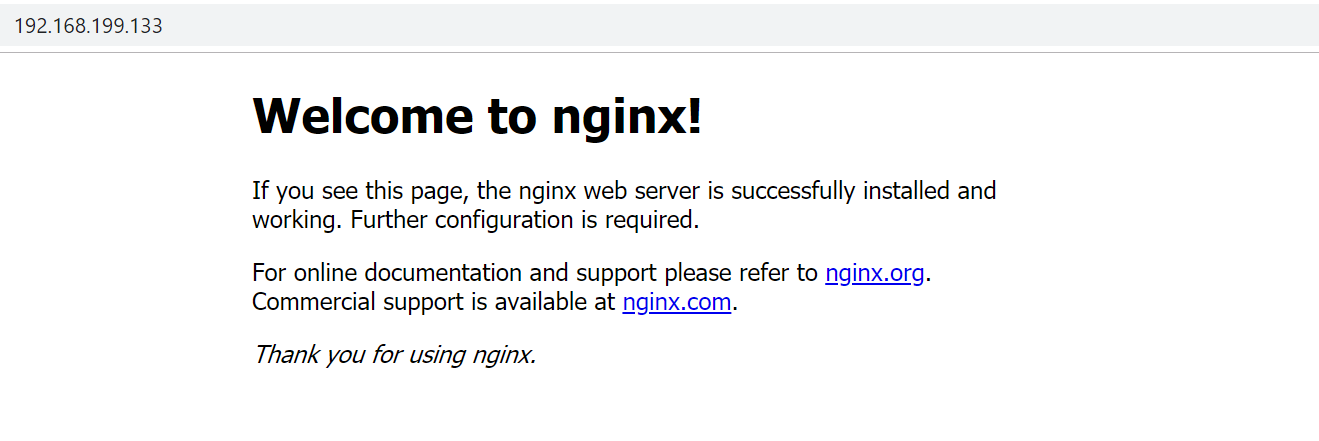
./configure --prefix=/usr/local/nginx/ --with-http\_realip\_module --with-http\_addition\_module --with-http\_gzip\_static\_module --with-http\_secure\_link\_module --with-http\_stub\_status\_module --with-stream --with-pcre=/home/alpha/share/nginx/pcre-8.41 --with-zlib=/home/alpha/share/nginx/zlib-1.2.11 --with-openssl=/home/alpha/share/nginx/openssl-1.1.0g

$ make

$ sudo make install

在/usr/local/目录下面， 产生了 nginx 的目录

$ ./sbin/nginx –c ./conf/nginx.conf



alpha@alpha:/usr/local/nginx$ ls

client\_body\_temp conf fastcgi\_temp html logs proxy\_temp sbin scgi\_temp uwsgi\_temp

alpha@alpha:/usr/local/nginx$ sudo ./sbin/nginx -c ./c

client\_body\_temp/ conf/

alpha@alpha:/usr/local/nginx$ sudo ./sbin/nginx -c ./conf/nginx.conf

[sudo] password for alpha:

alpha@alpha:/usr/local/nginx$ netstat -anop | grep 80

(No info could be read for "-p": geteuid()=1000 but you should be root.)

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN

demo版配置:

vi ./conf/demo.conf

events{

worker\_connections 1024;

}

http{

server{

listen 8080;

server\_name localhost;

location /{

root /usr/local/nginx/html/;

}

}

}

root@alpha:/usr/local/nginx# vi ./conf/demo.conf

root@alpha:/usr/local/nginx# ./sbin/nginx -c ./conf/demo.conf

提供多个服务给外界去用，分的越细越好

events{

worker\_connections 1024;

}

http{

server{

listen 8080; //通过8080端口去代理另一台服务器的服务

server\_name localhost;

location /{

# root /usr/local/nginx/html/;

proxy\_pass http://192.168.126.136;

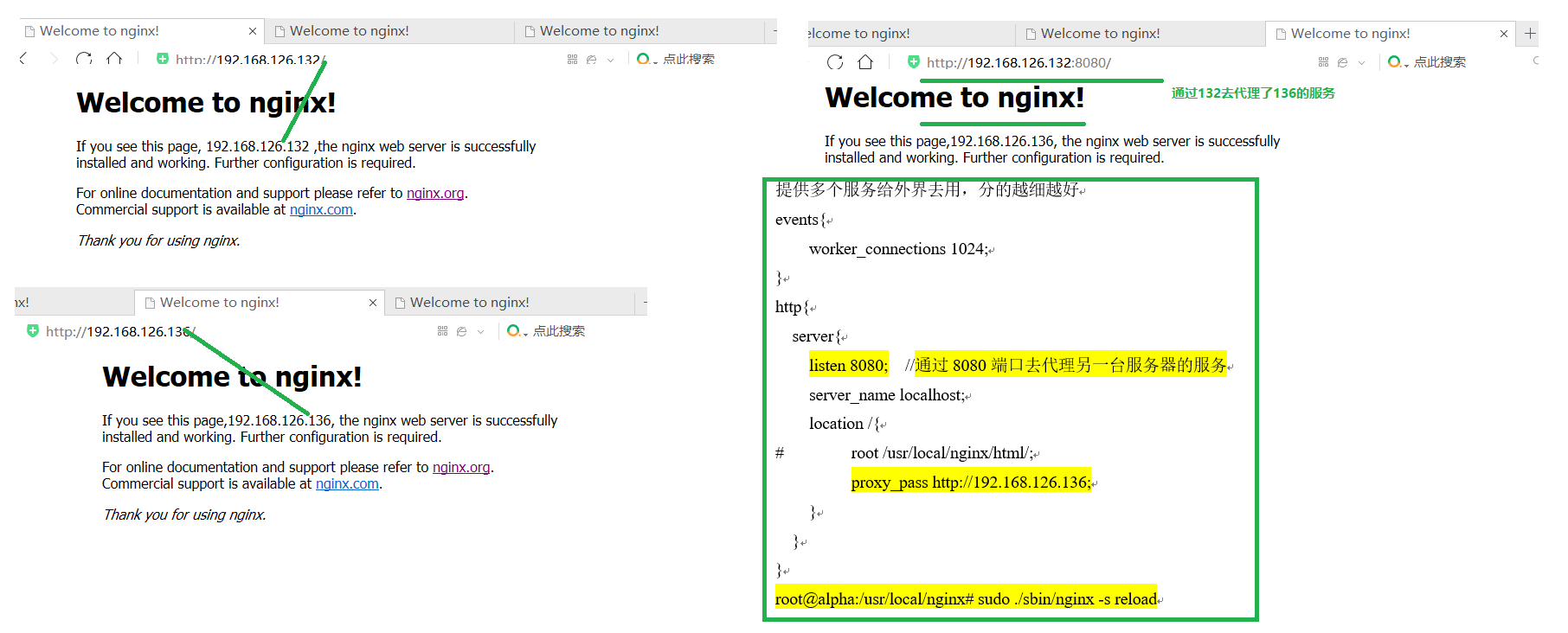
}

}

}

root@alpha:/usr/local/nginx# sudo ./sbin/nginx -s reload

重新加载配置文件



代理：使用本地IP，别人的网页

重定向，使用的是别人的IP和网页

图片：

events{

worker\_connections 1024;

}

http{

server{

listen 8080;

server\_name localhost;

location /{

# root /usr/local/nginx/html/;

proxy\_pass http://192.168.126.136;

location /images/{

root /usr/local/nginx/html/;

}

}

}

}

## 负载均衡：

使用一台机器代理另外两台机器

root@alpha:/usr/local/nginx/conf# vi proxy.conf

events{

worker\_connections 1024;

}

http{

upstream backend{

server 192.168.126.136;

server 192.168.126.137;

}

server{

listen 8888;

server\_name localhost;

location /{

# root /usr/local/nginx/html/;

# proxy\_pass http://192.168.126.136;

proxy\_pass http://backend;

location /images/{

root /usr/local/nginx/html/;

}

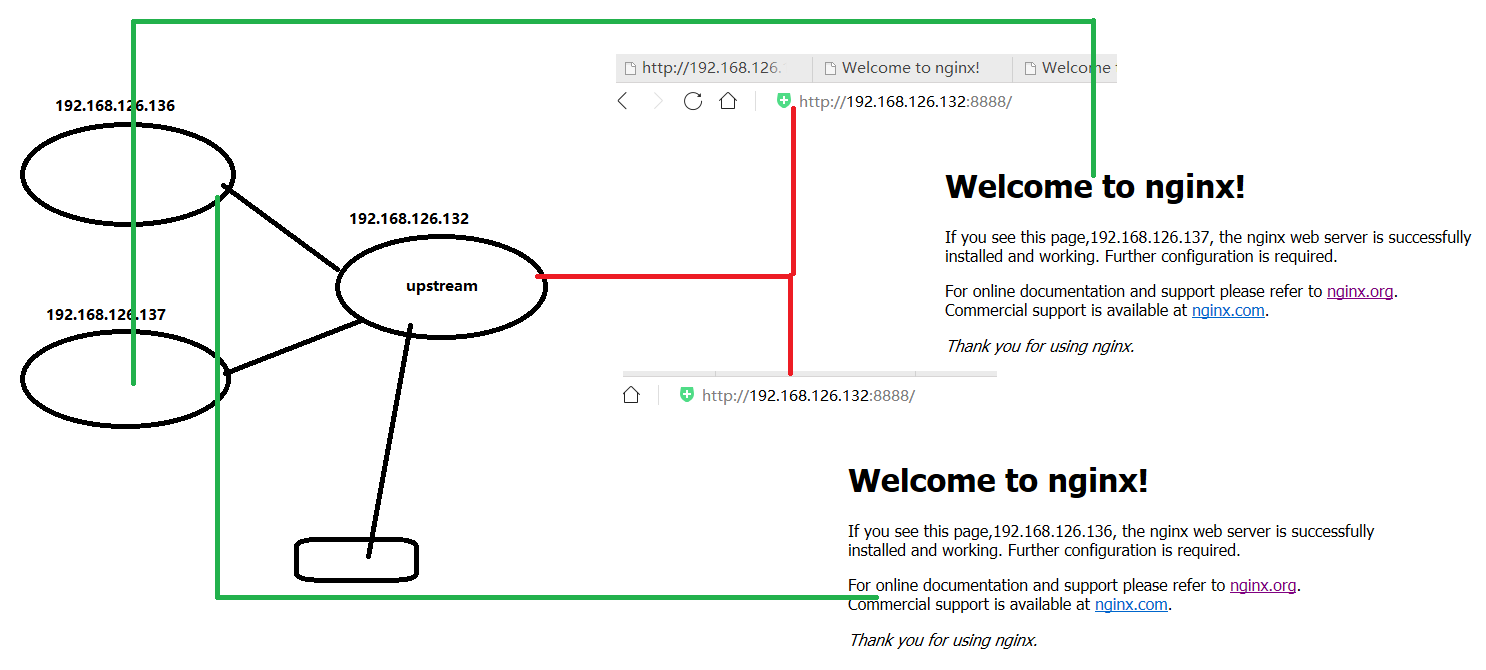
}

}

}

server 192.168.126.136 weight=2; 配置权重

server 192.168.126.137 weight=1;



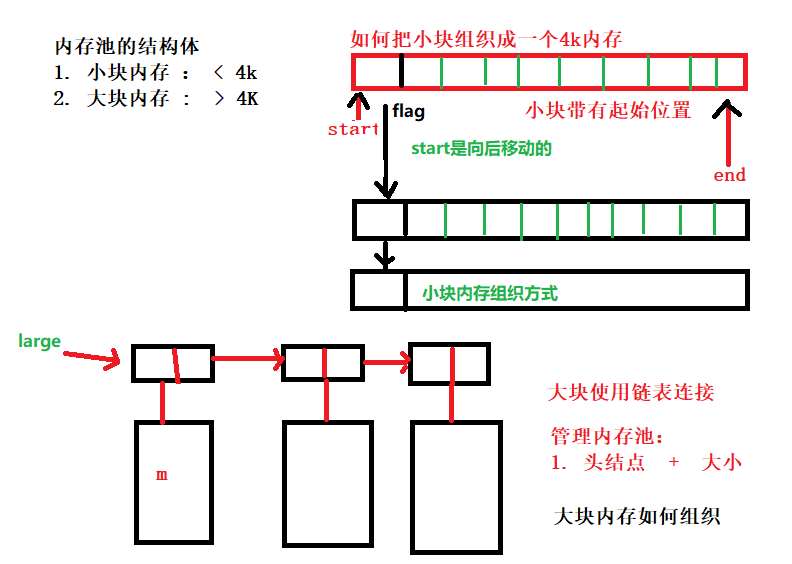
## Restful Http第三方接口实现，异步Http请求

## Nginx内存池

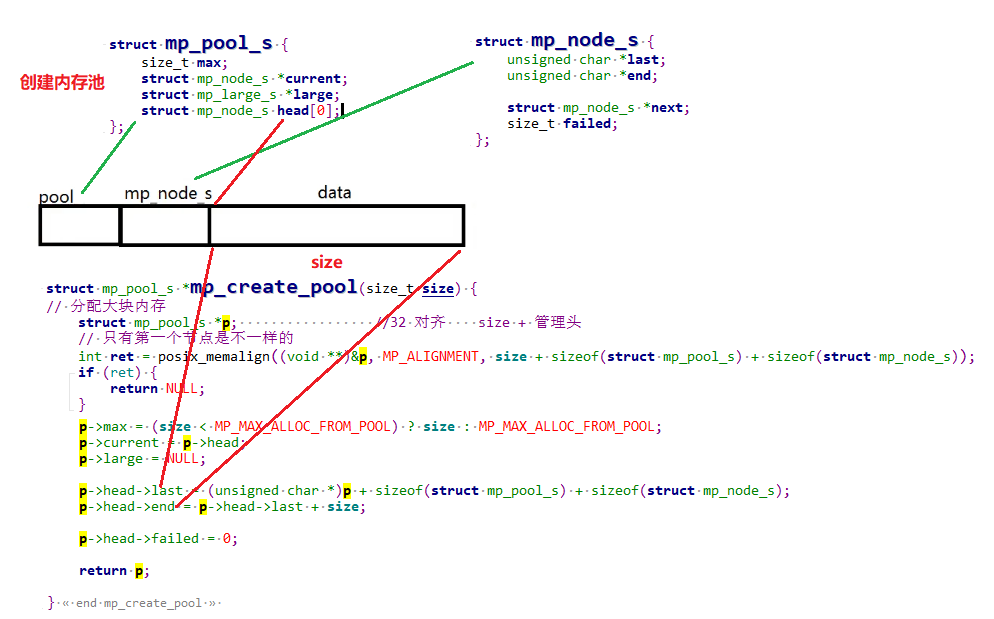
总的来说，提前将需要的内存空间分配好，如果小的直接标记使用；如果不够，则重新分配。

服务端接收到客户端的数据时，在高并发的情况下，有的大，有的小，那么怎么高效率的管理内存？

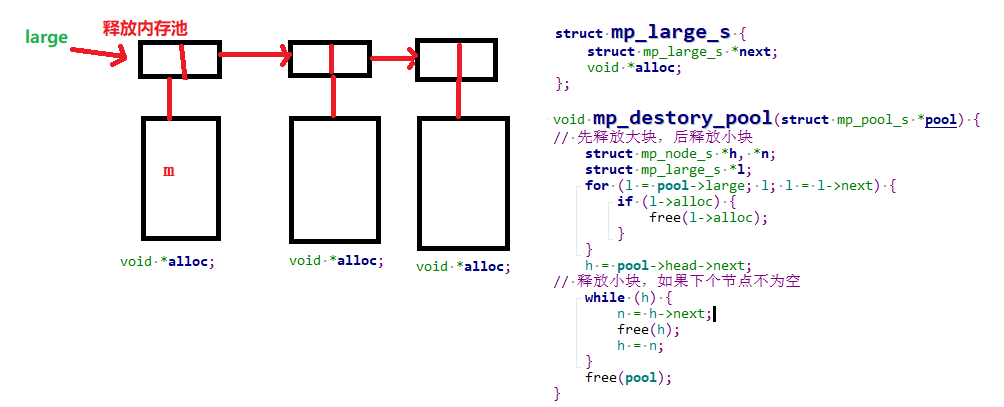
课堂上的内存池：



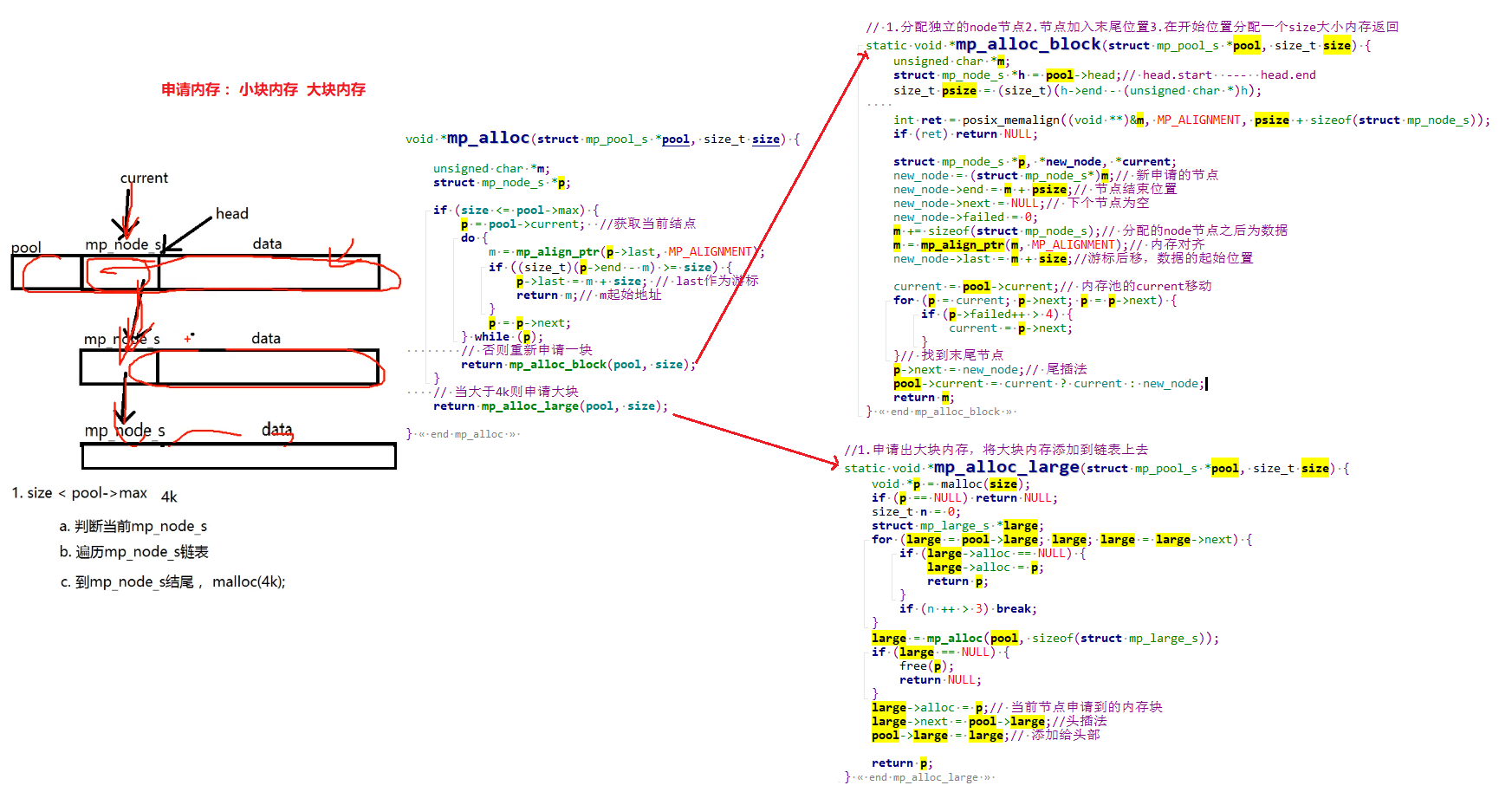
创建内存池：



销毁内存池：



申请大块或者小块内存：



课上讲的与nginx一样的，对于内存管理这块挺好的。

Nginx内存池：

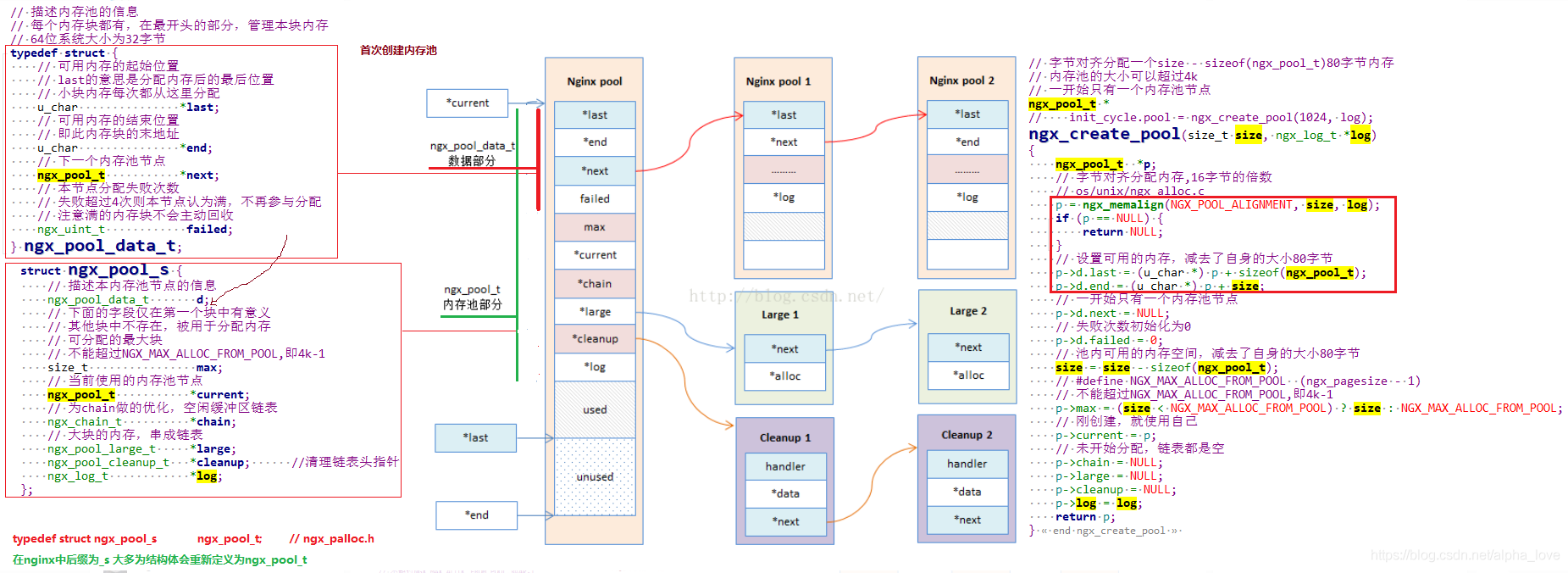
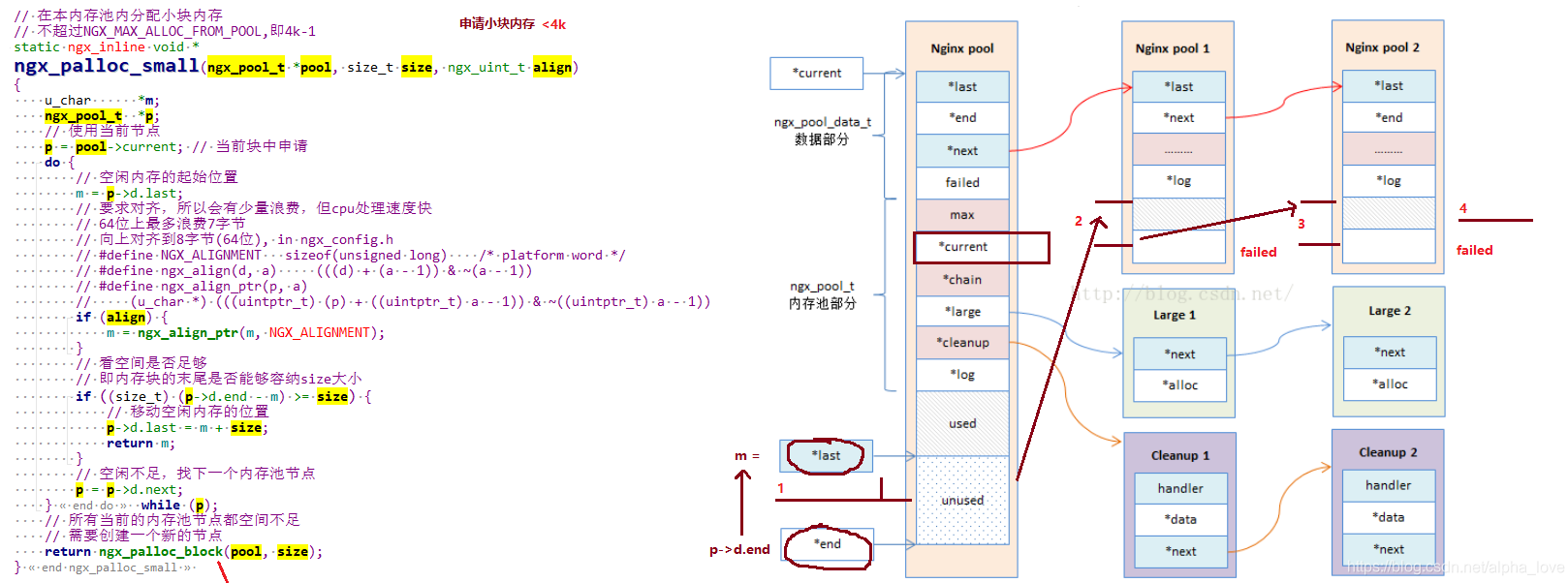
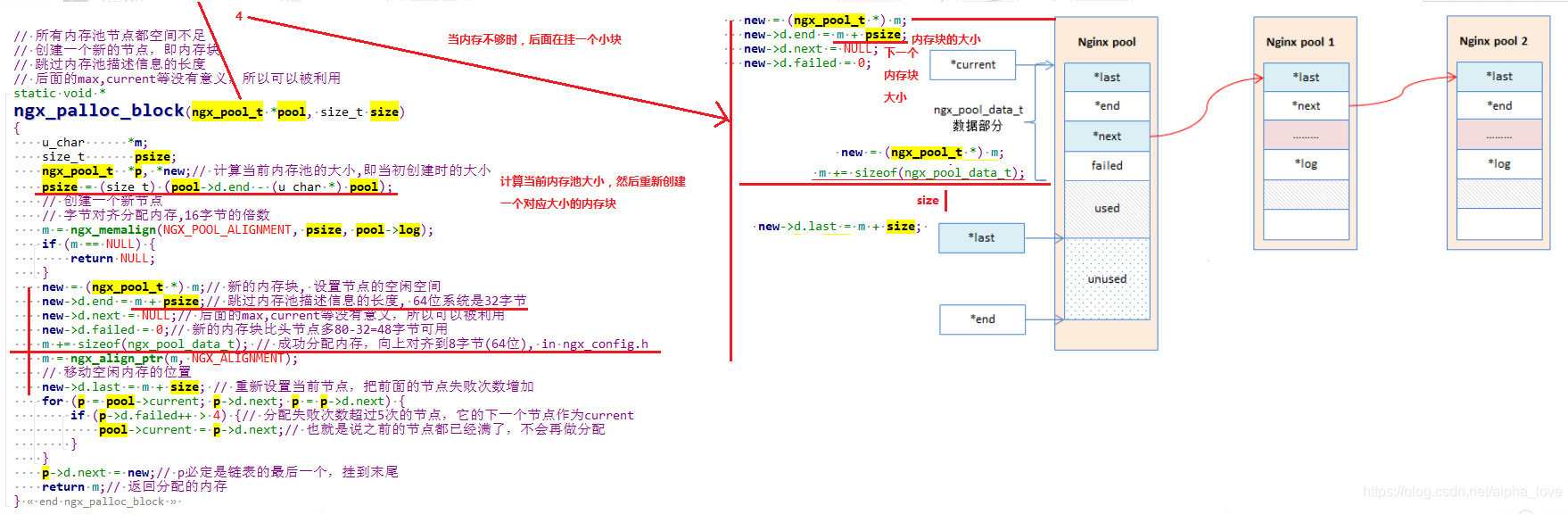
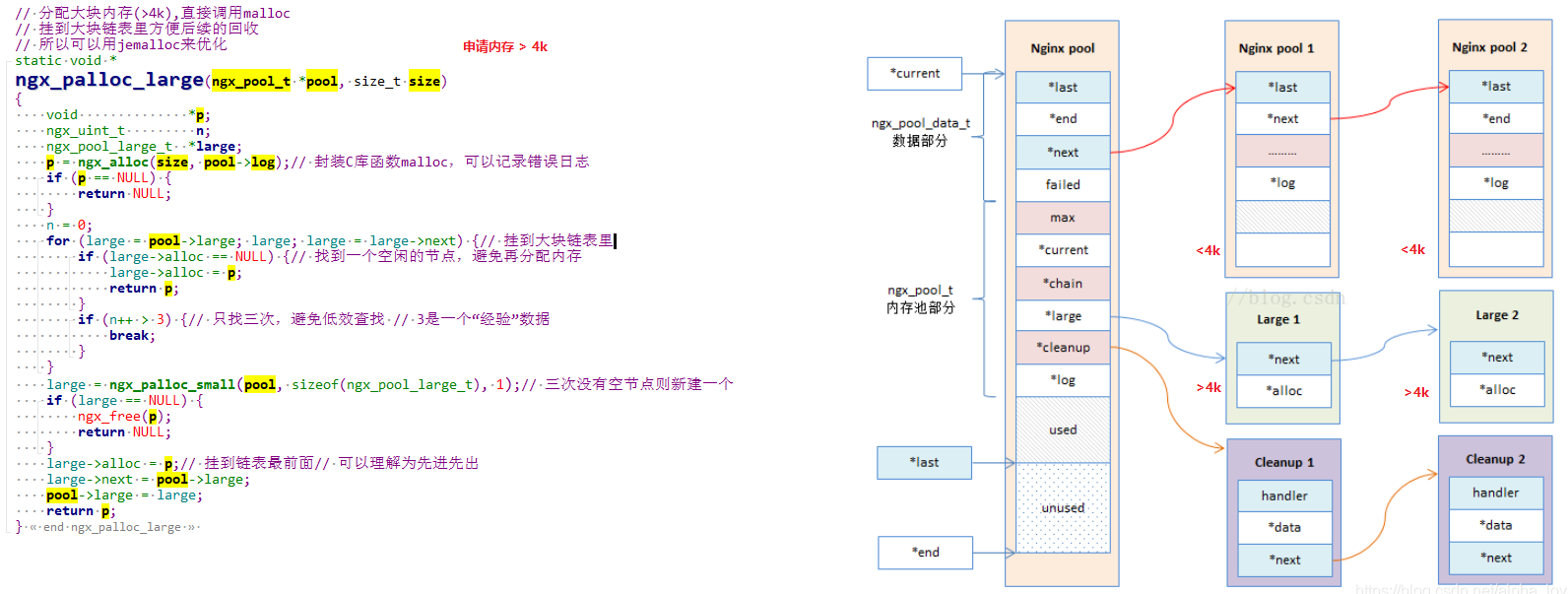
1. 在nginx中的每个http请求和tcp、udp都会去创建一个内存池，以后在处理过程中用到的内存都是从内存池中申请的。
2. nginx的内存池区分大内存块和小内存块，以一个内存页面为准，如果小于一个内存页面(getpagesize)则是小内存块，从内存池中直接申请，否则直接调用malloc去申请内存块
3. 在释放内存时，nginx没有专门提供针对释放小块内存的函数，小块内存会在ngx\_destory\_pool和ngx\_reset\_pool的时候一并释放。

内存分配逻辑:

分配一块内存，如果分配的内存size小于内存池的pool->max的限制，则属于小内存块分配，走小内存块分配逻辑；否则走大内存分配逻辑。小内存分配逻辑：

循环读取pool->d上的内存块，是否有足够的空间容纳需要分配的size，如果可以容纳，则直接分配内存；否则内存池需要申请新的内存块，调用ngx\_palloc\_block。

大内存分配逻辑：当分配的内存size大于内存池的pool->max的限制，则会直接调用ngx\_palloc\_large方法申请一块独立的内存块，并且将内存块挂载到pool->large的链表上进行统一管理。

2. cleanup机制 可以回调函数清理数据

Nginx的内存池cleanup机制，设计的非常巧妙。pool->cleanup本身是一个链表，每个ngx\_pool\_cleanup\_t的数据结构上，保存着内存数据的本身cleanup->data和回调清理函数cleanup->handler。通过cleanup的机制，我们就可以在内存池上保存例如文件句柄fd的资源。当我们调用ngx\_destroy\_pool方法销毁内存池的时候，首先会来清理pool->cleanup，并且都会执行c->handler(c->data)回调函数，用于清理资源。Nginx的这个机制，最显著的就是让文件描述符和需要自定义清理的数据的管理变得更加简单。分配一个cleanup结构：

/\*\*

 \* 分配一个可以用于回调函数清理内存块的内存

 \* 内存块仍旧在p->d或p->large上

 \* ngx\_pool\_t中的cleanup字段管理着一个特殊的链表，该链表的每一项都记录着一个特殊的需要释放的资源。

 \* 对于这个链表中每个节点所包含的资源如何去释放，是自说明的。这也就提供了非常大的灵活性。

 \* 意味着，ngx\_pool\_t不仅仅可以管理内存，通过这个机制，也可以管理任何需要释放的资源，

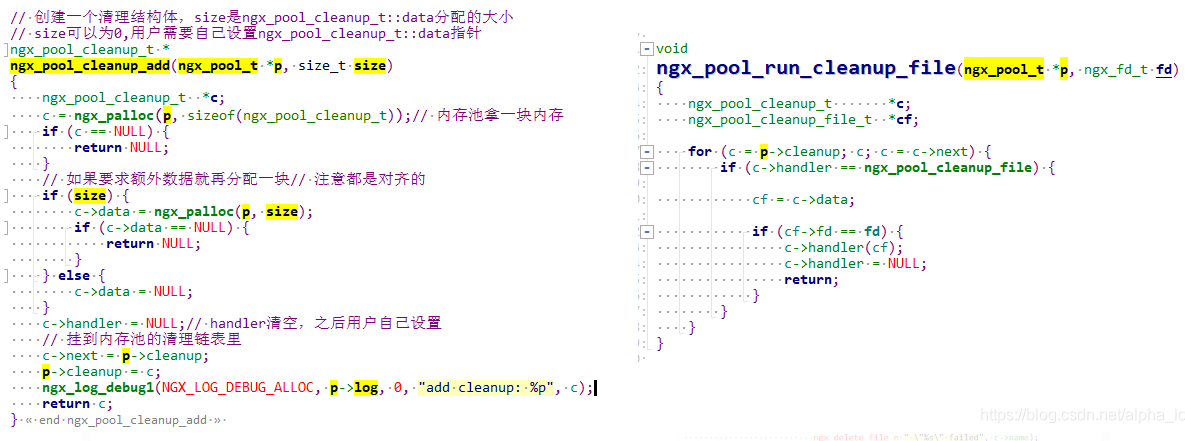
 \* 例如，关闭文件，或者删除文件等等的。下面我们看一下这个链表每个节点的类型

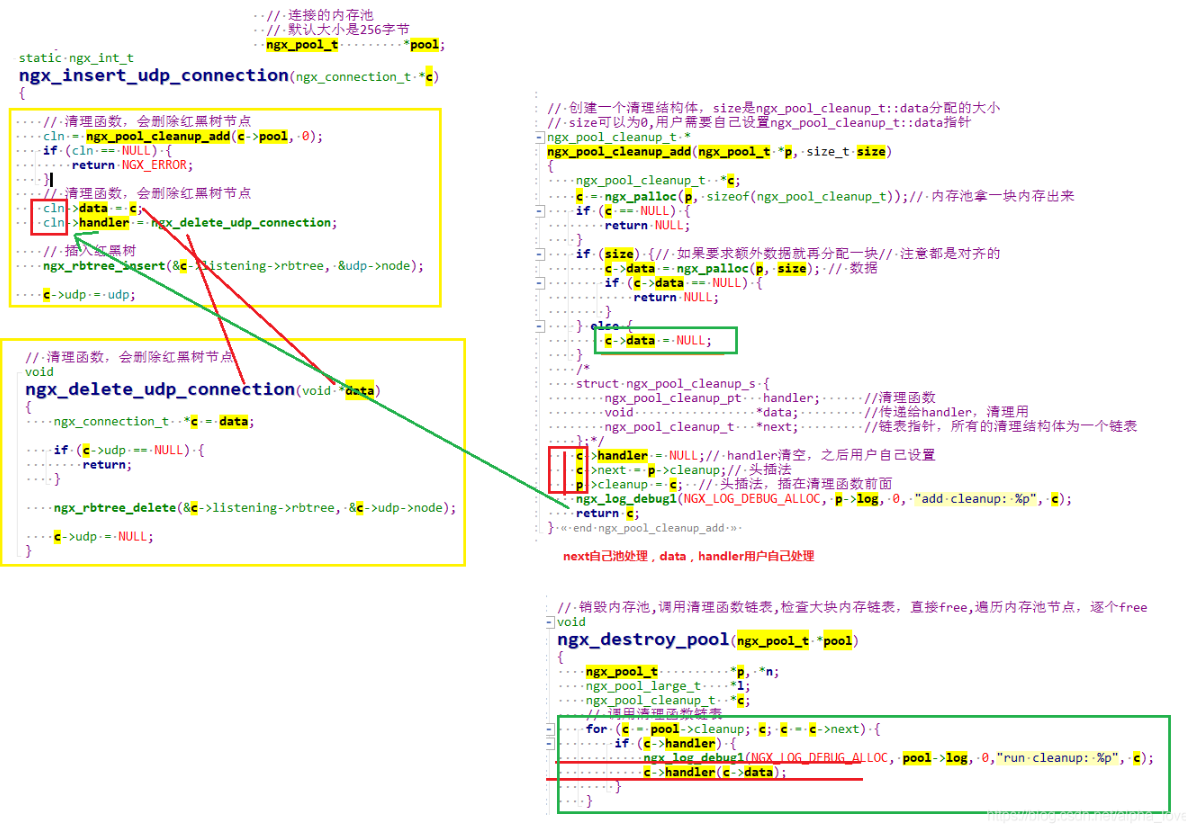
 \* 一般分两种情况：

 \* 1. 文件描述符

 \* 2. 外部自定义回调函数可以来清理内存

 \*/



举例：对资源进行释放，在调用内存池销毁函数时，将内存池中清理链表中的函数都执行一遍

# Cookie session token

https://www.liangzl.com/get-article-detail-16019.html

