

硕 士 研 究 生 读 书 报 告



题目 剖析轻量级高性能服务器Nginx

作者姓名 杨飞宇

作者学号 21651038

指导教师 李启雷

学科专业 软件工程

所在学院 软件学院

提交日期 二○一六 年 一 月

The Analysis Of Lightweight And High-performance Nginx Server

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Li

By

Feiyu Yang

Zhejiang University, P.R. China

2016

摘要

本文初步分析了Nginx在市场上的地位和使用情况，其发展历史和盛行的部分原因。也对Nginx平台进行了更深一步的研究和探索，全面分析了形成其独特优点的重要因素。最后也对Nginx的架构设计和源码实现有一定的分析和总结，这将有助于我们对市场上其他服务器的理解和符合特定业务需求的优秀服务器的开发。

**关键词：**Ngnix，服务器，架构

Abstract

The paper discusses the current usage and occupation of Nginx on the market and it’s development history and partial elements of the situation.Also it deeply analyses the platform and major factors which give Nginx the special advantages.At last it discusses and makes a summary of the architecture design and source code of Nginx.We can benefit from this paper and have a better understanding of other servers and develop specific softwares in a much more scientific way.

**Keywords：**software requirement, requirement analysis, system design

1 引言

二十世纪九十年代，随着互联网的迅速发展，一大批Web服务器相继应运而生。而以Apache为代表的Web服务器软件则开始长时间占领世界主流服务器的地位。

Apache的名字源于”a pathchy server“，即充满补丁的服务器。其开源、跨平台、稳定的特性，让它逐步被主流开发团队所接受，并不断有人为其贡献新的模块和功能。到如今，Apache已成为一个重量级的Web服务器。当其同时面对大量HTTP请求时，不得不产生更多进程来满足需求，这样的后果是内存的大量消耗和进程之间的来回切换带来的CPU资源浪费。

随着Apache缺点的不断被放大，Nginx逐渐成为世界主流开发团队的技术备选。面对不同的业务需求，Nginx的轻量级、高并发特性不断为其赢得软件开发工作者的青睐，进而成为这些年来和Apache抢占主要市场的对手之一。

2 Nginx发展历史

在2002年，俄罗斯系统管理人Igor Sysoev启动了Nginx项目，并于同年10月份发布了Nginx第一段公共简码。2003年，Ngnix首先被一个MP3下载站点使用，而后也服务于一个在线约会网站。在2006年之前，Nginx社区局限于俄语区，而后英语用户发现了它并为Nginx翻译文档、宣传项目，Nginx的受众才开始迅速扩大并为人所知晓。截止目前，Nginx已成为世界第二大Web服务器。

3 Nginx特性

Nginx既是一款轻量级的Web服务器，也可以承担反向代理服务器的职责，还能够充当电子邮件代理服务器。它占有较少的内存，能够支持高并发业务，速度快，扩展性强，同时支持热部署即在不中断服务的条件下升级原有软件。

形成这些优点的影响因素有很多，其核心因素是Nginx采用了一个master进程来管理多个worker进程的设计架构。

3.1 worker进程数与CPU核心数相等

本文均以Linux为基础平台来说明Nginx的运作方式。Nginx服务启动后，会在后台产生一个master进程和与CPU核心数相等的worker进程。worker进程之间的关系是平等的，worker进程是用来真正处理用户请求的进程，其数量与CPU核心数相同是为了更好地实现高并发。Nginx是异步非阻塞的，当worker进程数多于核心数，业务繁忙时为了切换进程而带来的CPU资源耗费是巨大的。而Linux平台下的CPU是抢占式的，仅仅使进程数与核心数相同是无法满足需求的。Nginx下各进程与核心分别绑定，真正做到了没有上下文切换和进程间切换带来的cathe失效等问题。

3.2 master进程对worker进程的管理

Nginx中为了更好地实现对worker进程的管理，master进程一般由root用户启用，使其权限略高与worker进程。当某一个worker进程出现错误时，master进程能够根据实际情况立即重启新的worker进程来接替。

master进程只负责管理，包括管理启停服务、重载配置文件、平滑升级等命令行服务，而所有的请求都由worker进程处理。这种设计方式也增加了服务器的健壮性。

3.3 充当反向代理服务器

用户向目标服务器即反向代理服务器发送请求，反向代理服务器根据请求内容将该请求转发给上游其他真正提供服务的服务器，并将上游服务器返回的结果返回给相应用户。

反向代理屏蔽了诸多细节，上游服务器的情况对用户均保持透明。这与正向代理恰好相反，正向代理中用户知道所要访问的上游服务器，但是无法直接实现反问。这时候就需要正向代理服务器的帮助，用户通过代理服务器取得上游服务器的内容，能够较灵活地突破所在网络的限制，并较大地提升与目标服务器的网络状况。

充当反向代理服务器是Nginx的一种常见用法。当Nginx充当前端服务器时，可直接向客户端提供静态文件服务，但是当业务逻辑复杂时，其反向代理的功能则显示出强大力量。Nginx支持高并发，能够承受高负载，能够在充当静态服务器的同时利用反向代理实现整个服务器系统的负载均衡。

当有新的HTTP请求到来时，Nginx不会直接将该请求转发到上游服务器。而是先将用户请求完整地接受到Nginx所在服务器内存或者硬盘中，再与上游服务器连接并转发请求。这与很多反向代理服务器边接受边转发是截然不同的，这种方式虽然从某种程度上增加了单个请求的处理时延，加大了内存和硬盘存储的压力，但是避免了较大请求所造成的上游服务器的压力。

用户与代理服务器之间的网络状况是很复杂的，通常情况下是网络负担较大的公网，网速慢，时延大，很多请求要较长的时间才能处理完。如果用户直接向目标服务器发起请求，业务繁忙时，很多请求不仅不能得到及时的处理，更会因为不断增长的网络连接数和积压的服务器作业而出现超时等错误。Nginx反向代理服务器和目标服务器之间多数情况使用的是网络状况良好的内网或专线，传输速度快，安全高效，大的客户请求能通过Nginx已做好的缓存迅速上传到目标服务器，避免了连接的长期保持，大大降低了上游服务器的负载，使得上游服务器能够专注于其核心业务的处理。

4 Nginx基础概念

4.1 connection

connection即为连接，是Nginx下对TCP连接的封装，其中包括了socket、读写事件等。利用Nginx中的connection，不仅可以处理前端的HTTP请求，还可以与后端交互。

通过一个完整的TCP连接，可以很好地理解Nginx对connection的处理。Nginx启动的时候，会载入和解析配置文件，监听相应的IP地址和端口号。接着，Nginx中的master进程会创建一个socket，设置好相应参数并绑定到指定的IP地址端口，然后进入listen状态，通过fork产生多个worker进程来进行accept服务。客户端通过三次握手与Nginx建立连接，并发送请求，相应的请求会被某个worker进程所accept，建立这个连接的socket，然后对其进行封装，并设置读写事件函数添加读写事件与用户进行数据交互。若Nginx做为反向代理服务器，则通过添加读写事件向上游服务器请求数据，然后将得到的结果返回给客户端。

Linux下，每个进程在PCB即进程控制块中都有一张文件描述符表，文件描述符fd是该表的索引，表中的每一项都有一个指向已打开文件的指针。在Nginx中，每一个socket都会占用掉一个fd，而fd数是有上限。fd的上限数和Nginx的worker进程数直接决定了服务器所支持的最大并发数。Nginx通过worker进程的独立连接池来管理相应的连接，连接池所保存的是一个ngx\_connection\_t结构数组。同时，为了更好地管理空闲ngx\_connection\_t，需要新的连接时都会通过一个free\_connections链表来取得空闲元素，并在使用完毕后归还回该链表。

Nginx中有一个重要的worker\_connections参数。在Nginx做为直接提供本地文件的服务器时，worker\_connections\*worker\_processes即worker进程数和每个进程能支持的并发连接数为Nginx服务器支持的最大并发连接数。但是当Nginx做为反向代理服务器时，情况则有所区别。反向代理服务器不仅要保持与客户端的连接，为了取得数据还要与上游服务器保持相应的连接，因此它所支持最大并发连接数应是前者的一半即（worker\_connections\*worker\_processes/2)。

4.2 accept

在服务器端，socket()返回的套接字用于listen监听和accept接受客户端的连接请求。accept()接受一个客户端的连接请求，并返回一个新的套接字，而与客户端的本次通信则通过这个新的套接字来完成。

当服务器中有多个空闲进程来争抢某一个新来连接时，那么某一进程的accept的机会总会较多，其空闲连接数也会不断减少直到耗完。当某个新的请求到达时，该进程会继续accept这一连接，而本身空闲连接数已不足且无法将这个请求再转接给其他进程，这样会造成大量的TCP连接无法得到处理。

Nginx避免以上情况的出现，只有获得accept\_mutex选项的进程才能去添加accept事件，通过这种方式也能实现对进程添加accept事件的控制。进程要想获得accept\_mutex锁，必须达到ngx\_accept\_disabled变量的相应条件。

ngx\_accept\_disabled的值为对应worker进程的总连接数的八分之一减去剩下的空闲连接数，ngx\_accept\_disabled的值越大表示worker进程让出mutex锁的机会越大，相应地其他worker进程获得这个请求的机会越大。这一机制让Nginx下多进程的连接竞争趋于稳定，有效保证多进程之间的连接平衡。

4.3 request

在Nginx下，request指HTTP请求。HTTP请求信息由三部分组成，请求方法URI协议/版本，请求头，正文。

请求方法包括GET、POST、PUT、HEAD等，而最常用的就是GET和POST。版本指明请求所使用的HTTP版本，例如HTTP/1.1。

请求头包含着许多有关客户端环境和请求正文的有用信息，其中Host、User-Agent、Accept-Encoding等选项给出相应的请求信息。

通常情况下的HTTP服务器会一行一行地分析请求行与请求头，并输出响应行和响应头。请求行中的方法、URI、HTTP版本等先被读取分析，而后根据请求头信息来读取请求体内容，并生成相应的响应数据返回给客户端。

Nginx下，request的起点从ngx\_http\_init\_request开始，该方法会设置读事件ngx\_http\_process\_request\_line来处理请求行。接着ngx\_http\_read\_request\_header读取请求数据，ngx\_http\_parse\_request\_line负责解析。Nginx采用了状态机来解析请求行，对method的比较采用字符串转整形而后比较的方式，这进一步提高了Nginx对事务处理的效率。请求行处理完毕之后，ngx\_http\_process\_request\_headers来完成对请求头的处理，与请求行的处理类似，使用ngx\_http\_read\_request\_header读取头信息，ngx\_http\_parse\_header\_line逐行解析，并使用ngx\_http\_request\_t的请求头链表headers\_in来保存所有的请求头信息。部分HTTP请求是需要特殊处理的，这些请求头和相应的处理函数会通过一张名为ngx\_http\_headers\_in的映射表连接起来。每当一个请求头被解析完毕，就会去这张映射表中查找对应的处理关系，如果存在则调用相应的处理函数来处理这一请求。

请求头的结束是以两个换行符来指明的，ngx\_http\_process\_request负责接下来的请求处理。但真正负责处理的是ngx\_http\_handler，由ngx\_http\_request\_handler

调用。ngx\_http\_handler会根据实际情况，分别调用ngx\_http\_request\_t中的read\_event\_handler和write\_event\_handler。Nginx将一个HTTP请求的处理分为多个阶段，ngx\_http\_core\_run\_phases函数来负责多阶段的请求处理，并将产生的响应头放入ngx\_http\_request\_t的headers\_out中。

Nginx通过filter来进行最后的数据过滤，header filter和body filter分别对响应头和响应体过滤。filter为一个链表结构，在header filter中，其最后一个filter需要遍历所有响应头，而body filter中最后一个filter为ngx\_http\_write\_filter，其负责HTTP处理请求的最后一项输出工作。

4.4 keepalive

TCP协议中，客户端与服务端的通信都要经过三次握手建立连接，在传输完数据之后通过四次交互释放连接。当网络业务繁忙时，为了建立和释放连接的操作会浪费大量网络资源。HTTP1.0和HTTP1.1都是支持长连接的，允许客户端在一个连接的情况下发送多次请求。

在客户端请求时，若请求需要body，比如POST请求，则nginx就需要客户端在请求时指明body的大小即content-length这一参数。

nginx的响应体长度需要根据客户端所采用的HTTP协议。在HTTP1.0中，若响应头中有content-length的长度，客户端就可以知道body的长度，客户端在接受时就能够根据具体长度来进行处理，若没有指明body长度，则会一直接受直到服务器断开连接。在HTTP1.1中，可以有chunked传输和非chunked传输两种。chunked传输情况下，body是流式的分块的，每一块的开始都会标识出当前块的长度，则无需content-length参数的帮助。当是非chunked传输时，有content-length则按其接受body，否则一直保持接收数据直到服务器断开连接。

是否支持长连接，这与客户端请求所设置的参数和所采用的协议有关。当请求中有connection这一项时，close表示关闭长连接，keep-alive表示打开长连接。如果这一项是缺失的，在HTTP1.0中是默认关闭，而HTTP1.1中是默认打开。

Nginx对待长连接的请求会设置keep-alive已保持对后续的请求处理，若超过最大等待时间则会关闭这一连接。在Nginx的响应头中，若打开长连接，则connection参数的值为Keep-Alive，否则为Close。

4.5 pipe

pipeline是HTTP1.1中引入的新特性，即流水线作业，可以将其视为keep-alive的升级版本。

在长连接中，后续请求必须在前面请求的响应被接受之后才能发起，则两次响应的间隔至少为2\*RTT。对于pipeline，请求可连续发送，响应时间能缩短到1\*RTT。

Nginx对pipeline是直接支持的，Nginx对请求的处理是依次进行的，利用buffer缓存读取的数据，每次都从中取出下一个请求，这样避免了连续请求中下一请求到来的等待。

4.6 lingering\_close

Nginx要关闭连接时，并不是直接关闭，而是先关闭TCP连接的写再等待一段时间，然后在关闭连接的读。

在请求服务中，客户端和服务器都可能会出错，Nginx在返回错误信息后，也需要关闭当前连接。Nginx调用write方法将错误信息返回发送给客户端，write方法返回成功仅仅表示信息已写入，客户端可能还未收到相应的信息。若此时，Nginx直接调用close方法来关闭TCP连接。内核会去检查TCP的read buffer里面是否还有客户端发来的数据留在内核态而未被用户态的进程读取，若有则发送RST报文来关闭TCP连接并丢弃write buffer里的数据，若没有则等待write buffer里面的数据发送完再经过四次交互断开连接。而在某些情况下，write buffer里面的数据无法在write调用之后close调用之前发送完毕，则会导致客户端只能收到RST报文而无法接受到错误信息。

lingering\_close的存在则很好的解决了这些问题，Nginx通过lingering\_timeout选项来设置读超时时间，若在这段时间内还未读取到数据则关闭连接。Nginx还支持设置一个总的读取时间，通过lingering\_time来设置，这段时间即Nginx关闭写之后保留socket的时间，客户端需要在这段时间内发送完所有数据，否则这段时间之后，Nginx会直接关闭连接。

5 基本数据结构

5.1 ngx\_str\_t

ngx\_str\_t是Nginx中对字符串的封装，其结构如下。

typedef struct {

size\_t len;

u\_char \*data;

} ngx\_str\_t;

data指向字符串数据的第一个字符，字符串的结束用长度来表示，而不是由’\0’来表示结束。len指明了长度，避免计算字符串长度的时间开销。Nginx通过这种方式也能重复去引用一段字符串内存，data是指针，可以指向相应的任何内存区域，节省了空间和拷贝带来的性能影响。

5.2 ngx\_pool\_t

该结构用来管理Nginx中的一系统资源，从这些资源的申请到释放，都由其负责。在内存管理中，若需要申请一块区域，则向ngx\_pool\_t对象来获取内存，而无需担心如何去分配和释放这些内存。在某个时间节点，ngx\_pool\_t会被系统销毁，相应的内存也就被回收了。使用文件时，文件需要打开和关闭，通过ngx\_pool\_t可以登记相关文件，并在销毁时关闭文件。

该结构也带来一个明显的缺点，内存无法得到及时的回收。比如变量a和b先后申请，a在使用完成之后理应收回取内存，而必须等到b使用完之后统一释放，这段时间区间内对a内存的利用则是不充分的。

5.3 ngx\_array\_t

Nginx中的数组结构是ngx\_array\_t，其定义如下。

typedef struct ngx\_array\_s ngx\_array\_t;

struct ngx\_array\_s {

void \*elts;

ngx\_uint\_t nelts;

size\_t size;

ngx\_uint\_t nalloc;

ngx\_pool\_t \*pool;

};

elts是实际数据存储区域的指针，nelts表示实际元素的个数，size表示单个元素的大小，nalloc为数组的容量，表示数组在不扩容的情况下最多存储的元素个数，当扩容操作时，相当于原内存空间2倍的连续内存会被分配，并将原数据拷贝到新区域中。而pool则为分配内存的内存池。

5.4 ngx\_hash\_t

Nginx有自己对hash表的实现，hash表解决冲突的方法有线性探测、二次探测和开链法等，ngx\_hash\_t则使用的是开链法，这与STL中hash表的使用方法相同。

Nginx中hash表的特点，只能一次初始化，构建hash表后无法进行插入和删除，hash表的开链实际是开了一段连续存储空间，类似一个数组。这是因为每次初始化ngx\_hash\_t时都会进行放入元素的个数计算，节省了一定的内存。

5.5 ngx\_hash\_wildcard\_t

ngx\_hash\_wildcard\_t是Nginx中为了解决带有通配符的域名匹配问题而设置的。它可以支持两种类型带有通配符的域名，一种是通配符在前，例如“\*.[qwe.cn](http://qwe.cn)”，另一种是类似“www.abc.\*”的通配符在后的域名。

5.6 ngx\_hash\_keys\_arrays\_t

在构建ngx\_hash\_wildcard\_t时，需要对通配符的key进行预处理。为了方便处理，需要三个hash表，一个包含普通的key的hash表，一个包含前向通配符的hash表，一个包含后向通配符的hash表。

5.7 ngx\_chain\_t

Nginx的filter模块在处理从别的filter模块或者是handler模块传递过来的数据中，这些数据就是以链表形式ngx\_chain\_t被传递的，其结构定义如下：

typedef struct ngx\_chain\_s ngx\_chain\_t;

struct ngx\_chain\_s {

ngx\_buf\_t \*buf;

ngx\_chain\_t \*next;

};

buf指向实际数据，而next指向下一个节点。

5.8 ngx\_buf\_t

该结构是ngx\_chain\_t中每个节点中的实际数据。该数据结构是抽象的，代表某种具体的数据。这些数据可能是指示链表读取者对读取数据进行不同处理的元数据，也可能是文件的一部分或内存中的缓冲区。

5.9 ngx\_list\_t

ngx\_list\_t类似于传统list结构的链表，能够轻松添加、删除元素，但其节点比较特殊，是一个固定大小的数组。初始化的时候，需要设定元素占用的空间大小以及每个节点数组的容量大小。在添加操作时，一个元素会被添加在最末节点的数组中，如果该节点空间已满，一个新的节点将被添加到list中。

定义如下

typedef struct {

ngx\_list\_part\_t \*last;

ngx\_list\_part\_t part;

size\_t size;

ngx\_uint\_t nalloc;

ngx\_pool\_t \*pool;

} ngx\_list\_t;

last指向最后一个节点，part表示该链表首个存放具体元素的节点，size指明链表中存放的具体元素所需内存大小，nalloc是每个节点所含的固定大小的数组容量，pool表示该结构所使用的内存池。

6 模块概述

Nginx由一部分核心和一系列模块组成，这样的构造是为了便于模块功能的开发和系统功能的扩展。Nginx将各功能模块组织成一条链，然后按照链上的每一环去处理请求的各个部分直到给出响应。

event module搭建了独立于操作系统的事件处理机制的框架，及提供了各具体事件的处理。

phase handler也称为handler模块，主要负责处理客户端请求并产生待响应内容。

output filter也称为filter模块，主要是负责对输出的内容进行处理，可以对输出进行修改。例如，可以实现对输出的所有html页面增加预定义的footbar一类的工作，或者对输出的图片的URL进行替换之类的工作。

upstream模块实现反向代理的功能，将真正的请求转发到后端服务器上，并从后端服务器上读取响应，发回客户端。upstream模块是一种特殊的handler，只不过响应内容不是真正由自己产生的，而是从后端服务器上读取的。

load-balancer是负载均衡模块，实现特定的算法，在众多的后端服务器中，选择一个服务器出来作为某个请求的转发服务器。

7 总结

Nginx 是一个优秀的事件驱动框架，在HTTP处理上很出色，不仅能用于Web服务，而且能够低负载处理高并发请求。其优秀的性能与开发者的巧妙设计密不可分，精妙的抽象设计，利用操作系统恰当地使用资源等，这些细节都让Nginx的性能更上一层楼。

学习这些系统设计的方法论，并加以编程实现，有助于服务器端开发者能力的提升，更有助于符合未来发展趋势的服务器开发。

参考文献

# [1]深入理解Nginx．陶辉[M]．北京：机械工业出版社，2013．

# [2]深入剖析Nginx. 高群凯[M]．北京：人民邮电出版社，2013．

# [3]学习Nginx HTTP Server. [陶利军](https://book.douban.com/search/%E9%99%B6%E5%88%A9%E5%86%9B) 翻译 Clement Nedelcu 中文版 [M]．北京：清华大学出版社，2012．