# **Nginx、Apache工作原理以及nginx为何比Apache高效**

Nginx才短短几年，就拿下了web服务器大笔江山，众所周知，Nginx在处理大并发静态请求方面，效率明显高于httpd，甚至能轻松解决C10K问题。

在高并发连接的情况下，Nginx是Apache服务器不错的替代品。Nginx同时也可以作为7层负载均衡服务器来使用。根据我的测试结果，Nginx0.7.14+[PHP](https://www.2cto.com/kf/web/php/" \t "https://www.2cto.com/net/201702/_blank)5.2.6(FastCGI)可以承受3万以上的并发连接数，相当于同等环境下Apache的10倍。

一般来说，4GB内存的服务器+Apache(prefork模式)一般只能处理3000个并发连接，因为它们将占用3GB以上的内存，还得为[系统](https://www.2cto.com/os/" \t "https://www.2cto.com/net/201702/_blank)预留1GB的内存。我曾经就有两台Apache服务器，因为在配置文件中设置的MaxClients为4000，当Apache并发连接数达到3800时，导致服务器内存和Swap空间用满而崩溃。

而这台Nginx0.7.14+PHP5.2.6(FastCGI)服务器在3万并发连接下，开启的10个Nginx进程消耗150M内存(15M\*10=150M)，开启的64个php-cgi进程消耗1280M内存(20M\*64=1280M)，加上系统自身消耗的内存，总共消耗不到2GB内存。如果服务器内存较小，完全可以只开启25个php-cgi进程，这样php-cgi消耗的总内存数才500M。

在3万并发连接下，访问Nginx0.7.14+PHP5.2.6(FastCGI)服务器的PHP程序，仍然速度飞快。

为什么Nginx在处理高并发方面要优于httpd，我们先从两种web服务器的工作原理以及工作模式说起。

apache三种工作模式

我们都知道Apache有三种工作模块，分别为prefork、worker、event。

prefork：多进程，每个请求用一个进程响应，这个过程会用到select机制来通知。

worker：多线程，一个进程可以生成多个线程，每个线程响应一个请求，但通知机制还是select不过可以接受更多的请求。

event：基于异步I/O模型，一个进程或线程，每个进程或线程响应多个用户请求，它是基于事件驱动(也就是epoll机制)实现的。

4.2prefork的工作原理

如果不用“--with-mpm”显式指定某种MPM,prefork就是Unix平台上缺省的MPM.它所采用的预派生子进程方式也是Apache1.3中采用的模式。prefork本身并没有使用到线程，2.0版使用它是为了与1.3版保持兼容性;另一方面，prefork用单独的子进程来处理不同的请求，进程之间是彼此独立的,这也使其成为最稳定的MPM之一。

4.3worker的工作原理

相对于prefork，worker是2.0版中全新的支持多线程和多进程混合模型的MPM。由于使用线程来处理，所以可以处理相对海量的请求，而系统资源的开销要小于基于进程的服务器。但是，worker也使用了多进程,每个进程又生成多个线程，以获得基于进程服务器的稳定性，这种MPM的工作方式将是Apache2.0的发展趋势。

4.4event基于事件机制的特性

一个进程响应多个用户请求，利用callback机制，让套接字复用，请求过来后进程并不处理请求，而是直接交由其他机制来处理，通过epoll机制来通知请求是否完成;在这个过程中，进程本身一直处于空闲状态，可以一直接收用户请求。可以实现一个进程程响应多个用户请求。支持持海量并发连接数，消耗更少的资源。

如何提高Web服务器的并发连接处理能力

有几个基本条件：

1.基于线程，即一个进程生成多个线程，每个线程响应用户的每个请求。

2.基于事件的模型，一个进程处理多个请求，并且通过epoll机制来通知用户请求完成。

3.基于磁盘的AIO(异步I/O)

4.支持mmap内存映射，mmap传统的web服务器，进行页面输入时，都是将磁盘的页面先输入到内核缓存中，再由内核缓存中复制一份到web服务器上，mmap机制就是让内核缓存与磁盘进行映射，web服务器，直接复制页面内容即可。不需要先把磁盘的上的页面先输入到内核缓存去。

刚好，Nginx支持以上所有特性。所以Nginx官网上说，Nginx支持50000并发，是有依据的。

Nginx优异之处

传统上基于进程或线程模型架构的web服务通过每进程或每线程处理并发连接请求，这势必会在网络和I/O操作时产生阻塞，其另一个必然结果则是对内存或CPU的利用率低下。生成一个新的进程/线程需要事先备好其运行时环境，这包括为其分配堆内存和栈内存，以及为其创建新的执行上下文等。这些操作都需要占用CPU，而且过多的进程/线程还会带来线程抖动或频繁的上下文切换，系统性能也会由此进一步下降。另一种高性能web服务器/web服务器反向代理：Nginx(EngineX)，nginx的主要着眼点就是其高性能以及对物理计算资源的高密度利用，因此其采用了不同的架构模型。受启发于多种操作系统设计中基于“事件”的高级处理机制，nginx采用了模块化、事件驱动、异步、单线程及非阻塞的架构，并大量采用了多路复用及事件通知机制。在nginx中，连接请求由为数不多的几个仅包含一个线程的进程worker以高效的回环(run-loop)机制进行处理，而每个worker可以并行处理数千个的并发连接及请求。

Nginx工作原理

Nginx会按需同时运行多个进程：一个主进程(master)和几个工作进程(worker)，配置了缓存时还会有缓存加载器进程(cacheloader)和缓存管理器进程(cachemanager)等。所有进程均是仅含有一个线程，并主要通过“共享内存”的机制实现进程间通信。主进程以root用户身份运行，而worker、cacheloader和cachemanager均应以非特权用户身份运行。

在高连接并发的情况下，Nginx是Apache服务器不错的替代品

Nginx安装非常的简单,配置文件非常简洁(还能够支持perl语法),Bugs非常少的服务器:Nginx启动特别容易,并且几乎可以做到7\*24不间断运行，即使运行数个月也不需要重新启动.你还能够不间断服务的情况下进行软件版本的升级。

Nginx的诞生主要解决C10K问题

最后我们从各自使用的多路复用IO模型来分析：select模型：(apache使用，由于受模块等限制，用的不多)

单个进程能够监视的文件描述符的数量存在最大限制

select()所维护的存储大量文件描述符的数据结构，随着文件描述符数量的增长，其在用户态和内核的地址空间的复制所引发的开销也会线性增长

由于网络响应时间的延迟使得大量TCP连接处于非活跃状态，但调用select()还是会对所有的socket进行一次线性扫描，会造成一定的开销

poll：poll是unix沿用select自己重新实现了一遍，唯一解决的问题是poll没有最大文件描述符数量的限制

epoll模型：(nginx使用)

epoll带来了两个优势，大幅度提升了性能：

基于事件的就绪通知方式，select/poll方式，进程只有在调用一定的方法后，内核才会对所有监视的文件描述符进行扫描，而epoll事件通过epoll\_ctl()注册一个文件描述符，一旦某个文件描述符就绪时，内核会采用类似call back的回调机制，迅速激活这个文件描述符，epoll\_wait()便会得到通知

调用一次epoll\_wait()获得就绪文件描述符时，返回的并不是实际的描述符，而是一个代表就绪描述符数量的值，拿到这些值去epoll指定的一个数组中依次取得相应数量的文件描述符即可，这里使用内存映射(mmap)技术，避免了复制大量文件描述符带来的开销

当然epoll也有一定的局限性，epoll只有[Linux](https://www.2cto.com/os/linux/" \t "https://www.2cto.com/net/201702/_blank)2.6才有实现，而其他平台都没有，这和apache这种优秀的跨平台服务器，显然是有些背道而驰了。简单来说epoll是select的升级版，单进程管理的文件描述符没有最大限制。但epoll只有linux平台可使用。作为跨平台的Apache没有使用。