[**nginx线程池模式探讨**](http://shift-alt-ctrl.iteye.com/blog/2337476)

**博客分类：**

* [nginx](http://shift-alt-ctrl.iteye.com/category/343868)

    nginx的IO模型其实大家应该有所了解，简单而言，就是一个master进程和多个worker进程（进程数由配置决定），master进程负责accept请求并队列化，最后转发给worker进程并由其进行处理请求和响应的整个过程。不过，这是进程层面，每个进程单线程处理。nginx在1.7.11版本提供了多线程特性（multi-threading），不过这个多线程仅用在aio模型（IO模型）中对本地文件的操作上，出发点就是以非阻塞模式来提高文件IO的效率和并发能力；所以这个多线程，并不是nginx通过多线程的方式处理proxy request（这部分是通过epoll模式），而是用来处理本地的一些静态文件。

    这里涉及到几个指令：sendfile、aio、directio，它们均与本地文件的操作有关，我们先看看它们的意义：

**1、sendfile：**

    跟系统“sendfile()”参数具有相同的语义，sendfile的目的就是提高本地文件通过socket发送的效率；磁盘、网络驱动器、内存是三种不同的传输介质，如果从本地读取一个文件并通过socket发送出去，通常情况下是进过如下几个步骤：

    1）磁盘驱动器从根据CPU的调度，从磁盘读取一定长度（chunk）的字节数据 2）字节数据copy到内核内存中 3）将内核内存中的数据copy到进程工作区内存 4）进程通过socket将数据copy到网络驱动器缓存， 并通过相应的传输协议发送出去。

    我们能够简单看出，数据的发送过程涉及到多次copy，这受限于计算机系统的设计问题；那么sendfile的主要出发点就是减少数据的copy以提高发送效率，sendfile是linux系统级的调用，socket可以通过DMA（直接内存访问）方式直接访问文件数据，并通过传输协议发送，减少了2次数据copy（磁盘到内核，内核到工作区）。

    sendfile\_max\_chunk参数用于限定每次sendfile()调用发送的最大数据尺寸，如果不限制大小的话，将会独占整个worker进程，默认为“无限制”。

    对于nginx而言，代理静态本地的静态文件资源（通常是小文件）将是非常高效的，建议对一些静态文件比如html、图片等，开启此参数。

**Java代码  [收藏代码](javascript:void())**

1. location /video {
2. sendfile on;
3. sendfile\_max\_chunk 128k;
4. aio on;
5. }

**2、directio**

    开启对O\_DIRECT标记（BSD，linux）的使用，对应directio()这个系统调用，这个参数是针对大文件而设定的，sendfile针对的是小文件，通过directio可以指定限定的尺寸大小，对于超过此size的文件，将会使用directio（而不再使用sendfile）。根据directio的设计初衷，它具备sendfile的基本原理，只是不使用内核cache，而是直接使用DMA，而且使用之后内存cache（页对齐部分）也将被释放，因此directio通常适用于大文件读取，而且通常读取频率很低，对于高频的读取并不能提高效率（因为其不会重用cache，而是每次都DMA）。因为存在性能权衡问题，此参数默认为off。

**Java代码  [收藏代码](javascript:void())**

1. location /video {
2. sendfile on;
3. directio 8m;
4. aio on;
5. }

**3、aio**

    我们会联想到aio模型，其实语义也基本相同，就是异步文件IO，nginx默认关闭此特性，它需要在高版本的linux平台上才支持（2.6.22+）。在linux平台上，directio只能读取基于512字节边界对齐的blocks，文件结束的那些未对齐的block将使用阻塞模式读取，同样那些文件在开头没有对齐时整个文件都将阻塞式读取，这里所谓的对齐，就是文件数据在内存页中的cache情况。

    当aio和sendfile都开启时，将会对那些size大于directio设定值的文件使用aio机制，即当小于directio设定值的文件将直接使用sendfile（aio不参与）。

    aio简单而言，就是使用多线程异步模式读取较大的文件，以提高IO效率，但是事实上可能并没有任何提高，因为大文件的读取即不能使用cache、而且本身也是耗时的，即使是多线程，对于request的等待时间也是无法预估的，特别是并发请求较高的时候，但是aio提高IO的并发能力是确定的。

    默认情况下，多线程模式是关闭的，我们需要通过“--with-threads”配置来开启，此特性尽在支持epoll、kqueue的平台上兼容。对于线程池的设置，我们可以通过“thread\_pool”来声明，并在aio指令中指定。

**Java代码  [收藏代码](javascript:void())**

1. thread\_pool default\_pool threads=16;##main上下文
2. ...
3. location /video {
4. sendfile on;
5. sendfile\_max\_chunk 128k;
6. directio 8M;
7. aio threads=default\_pool;
8. }

    当线程池中所有的线程都处于busy状态，那么新的task请求将会加入到等待队列，我们可以在thread\_pool中使用“max\_queue”参数来指定队列的大小，默认队列大小为65536，当队列已满后续的请求将会抛出error。

**4、其他**

    nginx官方宣称使用多线程模式，在aio读取文件场景下，性能有9倍的提升，[参见文档](https://www.nginx.com/blog/thread-pools-boost-performance-9x/)；但是本人还是对这个测试具有一定怀疑态度，多线程 + aio在一定程度的确可以提高文件IO的读取性能，但是对于大文件而言，这似乎并没有想象的那么优秀，这受制于linux平台底层的本身特性，除非nginx自己对文件cache做了额外的操作。

    到目前为止，我们仍然建议大家：

    1）对于小文件的静态代理，我们应该开启sendfile，这对性能的提升是显著的。

    2）对于大文件读取（低频），我们可以尝试开启aio、directio，在提升并发能力的前提下，关注request的实际响应效率；既然官方推荐这么用，我们可以报以尝试的态度。

    3）对于高频单文件读取，aio、directio的性能或许提升并不显著，但应该不会降低性能。