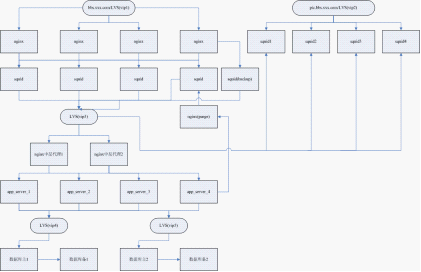
这个架构基于squid、nginx和lvs等技术，从架构上对bbs进行全面优化和保护，有如下特点：  
  
1、高性能：所有的点击基本上全部由前端缓存负责，提供最快速的处理。  
  
2、高保障度：不需考虑应用程序稳定与否、程序语言是何种、数据库是何种，都能从架构上保证稳定。  
  
3、高可用性：对应用程序的修改达到最简化：在程序的某些地方加入清缓存的语句即可，当然还需要做页面静态化的工作和统计工作。  
  
首先看图，这个图比较大：  
   
  
  
这个架构的特点和一些流程的说明：  
  
1、主域名和图片域名分离  
  
域名分离可以使流量分离，缓存策略分离等等，好处诸多。bbs初期一定要做好规划，将图片用另外的域名独立服务，即使没有足够机器，域名也要先分开。另外，图片服务器可以使用有别于主域名的另一个域名，一个好处是可以减少读取cookie对图片服务器的压力，另一个是提高安全性，避免cookie泄露。  
  
2、使用LVS作为前端、二级代理和数据库的访问入口  
  
使用LVS作为入口，比其他任何一种方式都来得更优质。首先LVS的负载能力很强，因为它工作在网络协议的第4层，使用虚拟ip技术，所以它本身并不担负任何流量的处理，仅仅是一个封包转发的功能；第二，LVS的配置相对简单而且稳定，一般去调整的几率比较低，也减少了因人为等因素而出现故障；第三，LVS可以处理任何端口的负载均衡，所以它基本可以做所有服务的负载均衡和容错。在这个架构中，除了处理http的80端口之外，LVS也处理了数据库mysql的3306端口，在数据库这个应用中是采用的双机热备策略。  
  
3、使用nginx+squid作为最前端的缓存组合  
  
在这个架构中，是最能体现app\_nginx\_squid\_nginx架构的优势的。在这个架构中的bbs运行在缓存上，用户每发布一张帖子，都需要使用purge指令清除该帖子的缓存，如果是squid在最前端，那么每次发布一张帖子，都需要在所有的squid中调用purge指令，这样在机器比较多的时候，purge将成为一个巨大的压力。  
  
所以在这里将nginx放在最前端并使用手工url\_hash的方式分流，将经常需要purge的帖子页面和列表页面按一个url对应一台squid的策略，分布到各台squid上，并提供了一台或一组backup的squid，个别squid出现异常时将自动使用backup的机器继续提供一段时间的服务直到其正常。在这样的架构下，purge就不再是关键问题，因为一个url只会对应到一台机器上，所以purge的时候，后端app\_server找到对应的机器就可以了。  
  
可以看到在前端中还有一台nginx(purge)的机器，这台机器是专用于purge的，只要发送purge指令和需要清除的url到这台机器，就可以找到相应的服务器并清除缓存了。另外，purge时还需要清理backup机器上的缓存，所以无论前端机器增加到多少，purge指令只会在2台机器上执行，如果backup机器使用到2-3台，purge指令就会在3-4台机器上执行，仍然在可接受范围之内。  
  
nginx作为前端，另有的好处：  
  
1/使用nginx的日志统计点击量非常方便  
2/nginx也可作为缓存，一般可以直接负责favicon.ico和logo等固定的小图片  
  
4、基于nginx的中层代理  
  
nginx中层代理的优势，在：  
  
[nginx和squid配合搭建的web服务器前端系统](http://www.sudone.com/archie/app_nginx_squid.html)  
  
这篇文章中有解释。  
  
在这个架构中，假如后端的app\_server上把帖子页和列表页直接生成了静态页面，那么使用中层代理再做一次url\_hash，将可以解决后端app\_server的硬盘容量的压力，但是如果使用到url\_hash的话，那做容错就相对麻烦了。所以建议不要采用生成静态页的方式，后端的压力一般不会非常的大，所以没有必要生成静态页。假如前端squid的命中率实在太低下，造成大量穿透，可以考虑使用二级代理暂顶。  
  
5、基于LVS的数据库双机热备  
  
在这个架构中，因为大量的并发和访问量都由前端的缓存处理掉了，所以后端的mysql主要压力来自于数据的写入，所以压力并不是非常大，并且负载比较稳定，一般不会随着访问量上升而提高过快，估计目前一台64位的机器，加满内存并使用高速的硬盘，前端负载数亿访问量时数据库都不会出现性能问题。在数据库这方面应主要考虑故障恢复，因为数据库崩溃的话，按照一般使用备份恢复的做法，耗时很长而且难免丢失数据，是很棘手的问题。使用双机热备的方案，出现故障时首先可由一台时刻同步着的备用数据库即刻充当主数据库，然后卸下的数据库可以有充分的时间对其进行维修，所以是个很安全有效的办法。  
  
当然，数据库的优化还是要细心做的，参考：  
  
[mysql性能的检查和调优方法](http://www.sudone.com/linux/mysql_debug.html)  
  
细心地调一遍，性能会好很多。  
  
6、图片服务器  
  
图片服务器我在这个架构中没有特别详细的介绍，在大型的bbs系统下，图片常常会出现容灾现象——图片数量严重超过了单台前端服务器容纳能力，导致前端服务器命中率低下。处理容灾问题也是非常棘手的，往后会有更详细的介绍。  
  
7、简单的点击量统计办法  
  
1/使用js的script标签访问另一（台）组服务器的空文件，然后定期向数据库更新  
2/在前端的nginx上直接开启日志功能，按需要统计点击量的链接规则进行记录，然后定期更新数据库

Nginx是一款轻量级的Web服务器，由俄罗斯的程序设计师Igor Sysoev所开发，最初供俄国大型的入口网站及搜寻引Rambler使用。 其特点是占有内存少，并发能力强，事实上Nginx的并发能力确实在同类型的网站服务器中表现较好。

Nginx虽然可以比Apache处理更大的连接数，但是HTTP GET FLOOD针对的不仅仅是WEB服务器，还有数据库服务器。大量HTTP请求产生了大量的数据库查询，可以在几秒之内使数据库停止响应，系统负载升高，最终导致服务器当机。

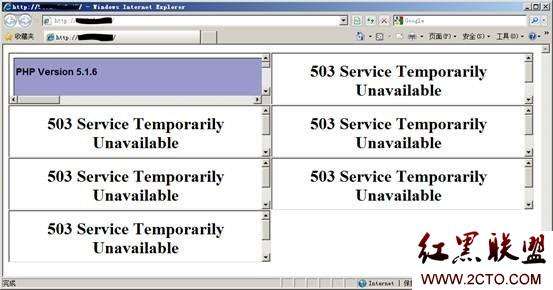
本文主要介绍Centos+Nginx下如何快速有效得防御CC攻击。至于如何安装Nginx就不详细介绍了，有兴趣的读者可以在Nginx官方网站（http://www.nginx.org/）下载源代码进行编译。如果你使用的是Centos5，也可以使用rpm包进行安装（http://centos.alt.ru/repository/centos/5/i386/nginx-stable-0.7.65-1.el5.i386.rpm）。

1. 主动抑制  
为了让Nginx支持更多的并发连接数，根据实际情况对工作线程数和每个工作线程支持的最大连接数进行调整。例如设置“worker\_processes 10”和“worker\_connections 1024”，那这台服务器支持的最大连接数就是10×1024=10240。  
worker\_processes 10;  
events {  
use epoll;  
worker\_connections 10240;  
}

Nginx 0.7开始提供了2个限制用户连接的模块：NginxHttpLimitZoneModule和NginxHttpLimitReqModule。  
NginxHttpLimitZoneModule可以根据条件进行并发连接数控制。  
例如可以定义以下代码：  
http {  
limit\_zone my\_zone $binary\_remote\_addr 10m;  
server {  
location /somedir/ {  
limit\_conn my\_zone 1;  
}  
}  
}  
其中“limit\_zone my\_zone $binary\_remote\_addr 10m”的意思是定义一个名称为my\_zone的存储区域、my\_zone中的内容为远程IP地址、my\_zone的大小为10M；“location /somedir/”的意思是针对somedir目录应用规则；“limit\_conn my\_zone 1”的意思是针对上面定义的my\_zone记录区记录的IP地址在指定的目录中只能建立一个连接。

NginxHttpLimitReqModule可以根据条件进行请求频率的控制。  
例如可以定义以下代码：  
http {  
limit\_req\_zone $binary\_remote\_addr zone=my\_req\_zone:10m rate=1r/s;  
...  
server {  
...  
location /somedir/ {   
limit\_req\_zone zone= my\_req\_zone burst=2;  
}  
其中“limit\_req\_zone $binary\_remote\_addr zone=my\_req\_zone:10m rate=1r/s”的意思是定义一个名称为my\_req\_zone的存储区域，my\_req\_zone内容为远程IP地址，my\_req\_zone大小为10M，my\_req\_zone中的平均请求速率只能为1个每秒；“location /somedir/”的意思是针对somedir目录应用规则；“limit\_req\_zone zone= my\_req\_zone burst=2”的意思是针对上面定义的my\_req\_zone记录区记录的IP地址在请求指定的目录中的内容时最高2个每秒的突发请求速率。  
当有连接触发上诉规则时，Nginx会报“503 Service Temporarily Unavailable”的错误，停止用户请求。返回一个503，对服务器来说影响不大，只占用一个nginx的线程而已，相对来说还是很划算的。

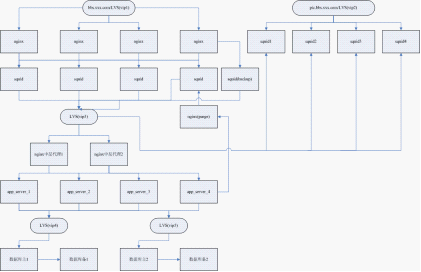
为了测试效果，我将以上代码放入Nginx的配置文件，并编写了一个php文件显示phpinfo；另外还写了一个html文件，其中嵌入了多个iframe调用php文件。当我打开这个html文件了，可以看到只有一个iframe中的php文件正常显示了，其他的iframe都显示503错误。



应用举例（Discuz！）  
Discuz!是使用比较多的一个php论坛程序。以Discuz!7.0为例，程序目录下有比较多的可以直接访问的php文件，但其中最容易受到攻击的一般有index.php（首页）、forumdisplay.php（板块显示）、viewthread.php（帖子显示）。攻击者一般会对这些页面发起大量的请求，导致HTTP服务器连接数耗尽、mysql数据库停止响应，最终导致服务器崩溃。  
为了防止上述页面被攻击，我们可以设定以下的规则进行防御：  
http {  
limit\_zone myzone\_bbs $binary\_remote\_addr 10m;  
limit\_req\_zone $binary\_remote\_addr zone=bbs:10m rate=1r/s;  
...  
server {  
...  
location ~ ^/bbs/(index|forumdisplay|viewthread).php$ {  
limit\_conn myzone\_bbs 3;  
limit\_req zone=bbs burst=2 nodelay;  
root html;  
fastcgi\_pass unix:/dev/shm/php-cgi.sock;  
fastcgi\_index index.php;  
fastcgi\_param SCRIPT\_FILENAME /usr/share/nginx/html$fastcgi\_script\_name;  
include fastcgi\_params;  
}  
}  
}  
应用这条规则后，bbs目录下的index.php、forumdisplay.php和viewthread.php这些页面同一个IP只许建立3个连接，并且每秒只能有1个请求（突发请求可以达到2个）。  
虽然这样的规则一般来说对正常的用户不会产生影响（极少有人在1秒内打开3个页面），但是为了防止影响那些手快的用户访问，可以在nginx中自定义503页面，503页面对用户进行提示，然后自动刷新。

**在nginx上处理一次小型的ddos事件**

nginx的负载能力超强，一般小的ddos是无法击垮一台nginx代理的，所以用nginx来过滤掉一些小型的ddos是完全没有问题。  
早上同事负责的一台服务器当机，重起之后又挂掉，他检查过后发现是有一个链接的访问量很大。  
我去看了一下，因为这个项目不是公司的项目，所以应用服务器是Windows+IIS，架构采用的是app\_squid架构。因为那个访问量大的链接业务上必须加上no-cache头，所以全部透入后台应用服务器，应用服务器招架不住所以频频出问题。  
我对IIS根本不熟，况且那台应用服务器已经基本没法正常服务，所以我就在cache上把受攻击的域名指向中层代理，中层代理再转发到应用服务器。这样架构变成了app\_nginx\_squid架构，然后我就好正常地开始分析。  
整理思路，然后按部就班：  
先打开日志，看看到底是怎么回事，看过之后发现确实是有一个链接访问量很大，用命令行统计一下，发现这个链接是正常的所有访问请求100倍之多。由上可知，确实就是这个链接访问量超大引起问题，但是这些访问量是不是正常的呢？从它的ip看，并不是同一个ip，这就比较难判断了，如果确实是真实流量，我把它毙掉，那就会有人要追杀我了。  
具体原因的干脆先不查，我把它的no-cache头先干掉好了，这样至少不会死机。作为维护人员，那肯定先要想办法让网站正常服务，然后才能去找原因的。  
proxy\_hide\_header Cache-Control;  
嗯，这下这个链接就缓存到前端squid了，应用服务器不会死机了，不过这只是临时方法，不是长久之计，因为这样做影响了业务功能。  
所以继续分析日志，多加了几项参数到日志中，这时看到这些大量的请求有一个共同点，那就是user-agent都是一样的，都是MSIE 5.01，而IE5并不是主流浏览器。这样看来，这些请求都是同一个客户端恶意发起的，不知道是用了什么垃圾软件。  
找到了原因和特征，当然就可以配置把它干掉了，判断一下user-agent，如果是MSIE 5.01就把它丢到另外一个地方去就可以了，比如指向sudone.com，看看能不能抗得住？！  
location = /v2/index.php {  
include proxy.conf;  
if ( $http\_user\_agent ~\* "MSIE 5.01" ) {  
proxy\_pass http://www.sudone.com;  
#access\_log /home/logs/1.log main;  
}  
proxy\_pass http://iis.xxx.com;  
}  
最后开回Cache-Control，访问一下页面，嗯，这回一切正常，业务也没有被修改。

这个架构基于squid、nginx和lvs等技术，从架构上对bbs进行全面优化和保护，有如下特点：  
  
1、高性能：所有的点击基本上全部由前端缓存负责，提供最快速的处理。  
  
2、高保障度：不需考虑应用程序稳定与否、程序语言是何种、数据库是何种，都能从架构上保证稳定。  
  
3、高可用性：对应用程序的修改达到最简化：在程序的某些地方加入清缓存的语句即可，当然还需要做页面静态化的工作和统计工作。  
  
首先看图，这个图比较大：  
   
  
  
这个架构的特点和一些流程的说明：  
  
1、主域名和图片域名分离  
  
域名分离可以使流量分离，缓存策略分离等等，好处诸多。bbs初期一定要做好规划，将图片用另外的域名独立服务，即使没有足够机器，域名也要先分开。另外，图片服务器可以使用有别于主域名的另一个域名，一个好处是可以减少读取cookie对图片服务器的压力，另一个是提高安全性，避免cookie泄露。  
  
2、使用LVS作为前端、二级代理和数据库的访问入口  
  
使用LVS作为入口，比其他任何一种方式都来得更优质。首先LVS的负载能力很强，因为它工作在网络协议的第4层，使用虚拟ip技术，所以它本身并不担负任何流量的处理，仅仅是一个封包转发的功能；第二，LVS的配置相对简单而且稳定，一般去调整的几率比较低，也减少了因人为等因素而出现故障；第三，LVS可以处理任何端口的负载均衡，所以它基本可以做所有服务的负载均衡和容错。在这个架构中，除了处理http的80端口之外，LVS也处理了数据库mysql的3306端口，在数据库这个应用中是采用的双机热备策略。  
  
3、使用nginx+squid作为最前端的缓存组合  
  
在这个架构中，是最能体现app\_nginx\_squid\_nginx架构的优势的。在这个架构中的bbs运行在缓存上，用户每发布一张帖子，都需要使用purge指令清除该帖子的缓存，如果是squid在最前端，那么每次发布一张帖子，都需要在所有的squid中调用purge指令，这样在机器比较多的时候，purge将成为一个巨大的压力。  
  
所以在这里将nginx放在最前端并使用手工url\_hash的方式分流，将经常需要purge的帖子页面和列表页面按一个url对应一台squid的策略，分布到各台squid上，并提供了一台或一组backup的squid，个别squid出现异常时将自动使用backup的机器继续提供一段时间的服务直到其正常。在这样的架构下，purge就不再是关键问题，因为一个url只会对应到一台机器上，所以purge的时候，后端app\_server找到对应的机器就可以了。  
  
可以看到在前端中还有一台nginx(purge)的机器，这台机器是专用于purge的，只要发送purge指令和需要清除的url到这台机器，就可以找到相应的服务器并清除缓存了。另外，purge时还需要清理backup机器上的缓存，所以无论前端机器增加到多少，purge指令只会在2台机器上执行，如果backup机器使用到2-3台，purge指令就会在3-4台机器上执行，仍然在可接受范围之内。  
  
nginx作为前端，另有的好处：  
  
1/使用nginx的日志统计点击量非常方便  
2/nginx也可作为缓存，一般可以直接负责favicon.ico和logo等固定的小图片  
  
4、基于nginx的中层代理  
  
nginx中层代理的优势，在：  
  
[nginx和squid配合搭建的web服务器前端系统](http://www.sudone.com/archie/app_nginx_squid.html)  
  
这篇文章中有解释。  
  
在这个架构中，假如后端的app\_server上把帖子页和列表页直接生成了静态页面，那么使用中层代理再做一次url\_hash，将可以解决后端app\_server的硬盘容量的压力，但是如果使用到url\_hash的话，那做容错就相对麻烦了。所以建议不要采用生成静态页的方式，后端的压力一般不会非常的大，所以没有必要生成静态页。假如前端squid的命中率实在太低下，造成大量穿透，可以考虑使用二级代理暂顶。  
  
5、基于LVS的数据库双机热备  
  
在这个架构中，因为大量的并发和访问量都由前端的缓存处理掉了，所以后端的mysql主要压力来自于数据的写入，所以压力并不是非常大，并且负载比较稳定，一般不会随着访问量上升而提高过快，估计目前一台64位的机器，加满内存并使用高速的硬盘，前端负载数亿访问量时数据库都不会出现性能问题。在数据库这方面应主要考虑故障恢复，因为数据库崩溃的话，按照一般使用备份恢复的做法，耗时很长而且难免丢失数据，是很棘手的问题。使用双机热备的方案，出现故障时首先可由一台时刻同步着的备用数据库即刻充当主数据库，然后卸下的数据库可以有充分的时间对其进行维修，所以是个很安全有效的办法。  
  
当然，数据库的优化还是要细心做的，参考：  
  
[mysql性能的检查和调优方法](http://www.sudone.com/linux/mysql_debug.html)  
  
细心地调一遍，性能会好很多。  
  
6、图片服务器  
  
图片服务器我在这个架构中没有特别详细的介绍，在大型的bbs系统下，图片常常会出现容灾现象——图片数量严重超过了单台前端服务器容纳能力，导致前端服务器命中率低下。处理容灾问题也是非常棘手的，往后会有更详细的介绍。  
  
7、简单的点击量统计办法  
  
1/使用js的script标签访问另一（台）组服务器的空文件，然后定期向数据库更新  
2/在前端的nginx上直接开启日志功能，按需要统计点击量的链接规则进行记录，然后定期更新数据库

这段时间，公司的web架构要升级，考虑用负载均衡；初期准备采用LVS+Keepalived，我比较有自信，刚刚在一个客户的局域网里实现了这个，所以直接把脚本移过来了；然而，杯具开始了，发现LVS怎么也实现不了后端二台web的转发。

后来关于此问题我请教了田逸兄，他怀疑我们的网络环境太复杂了，因为牵涉到内外网的问题，我们的每台机器上有5条静态路由，2个gateway，直接导致了LVS的不成功；我们试图跟network Engeneer沟通，结果是网络不能做一丝一毫改动，所以白白测试了二天。

后来改用了Nginx负载均衡器，5分钟就解决了问题，真真切切的体会到了Nginx对网络的依赖较小，理论上只要ping得通，网页访问正常，nginx就能连得通。为了以防万一，我采用的是Nginx+keepalived高可用架构。

在这里，我不是神话Nginx，只是说这是一种解决问题的方法而矣，LVS也有适用的场合，稳定性方面是众所周知的，所以只要提到web层的负载均衡，我就想到LVS，但LVS不仅仅是；如果网络环境比较复杂的朋友们，不妨换种思路解决问题。

当然用了Nginx后，大问题暂时没有；小问题就都来了，首先是SSL，这个目前支持得算是比较好的，在负载均衡器上开启ssl功能，监听443端口，将证书放在Nginx代理上，非后面的web服务器,轻构解决掉问题，详细见以下http.conf配置文件

1. server
2. {
3. listen 443;
4. server\_name www.cn7788.com;
6. ssl on;
7. ssl\_certificate /usr/local/nginx/keys/www.cn7788.com.crt;
8. ssl\_certificate\_key /usr/local/nginx/keys/www.cn7788.com.key;
10. ssl\_protocols SSLv3 TLSv1;
11. ssl\_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:-LOW:-SSLv2:-EXP;
12. }

但问题又来了，这么有个问题，跑在后方 apache 上的应用获取到的IP都是Nginx所在服务器的IP ，或者是本机 127.0.0.1 。最明显就是查看 apache 的访问日志。就会见到来来去去都是内网的IP；虽然可以通过Nginx日志来判断客户的client,但有些考虑不周全的应用，例如 Tattertools (一个博客程序) 就会犯误,后台的访问日志死活显示访客数 1，ip来自 127.0.0.1。这时候就要想办法来处理了。你可以通过修改 nginx proxy 的参数令后端应用获取到Nginx 发来的请求报文获取到外网的IP。

1. proxy\_set\_header Host $host;
2. proxy\_set\_header X-Real-IP $remote\_addr;
3. proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

这仅仅只是让Nginx获到到外网IP，Apache未必买帐呢，即Aapche端也需要设置，搜寻了一下，发现了apache这一个来自第三方的mod 配合Nginx proxy 使用。

说明：http://stderr.net/apache/rpaf/

下载：http://stderr.net/apache/rpaf/download/

最新版本是 mod\_rpaf-0.6.tar.gz

安装也相当简单。

# tar zxvf mod\_rpaf-0.6.tar.gz 下载后解压# cd mod\_rpaf-0.6

Apache 的目录按自己的环境修改，并选择相应的安装方式：

#/usr/local/apache/bin/apxs -i -c -n mod\_rpaf-2.0.so mod\_rpaf-2.0.c

完成后会在 http.conf 的 LoadModule 区域为你多加了一行。

LoadModule mod\_rpaf-2.0.so\_module modules/mod\_rpaf-2.0.so 经 apache 2.2.6 的实验，使用这一行启动 apache 的时候会报错的。

所以改为：

LoadModule rpaf\_module modules/mod\_rpaf-2.0.so

并在下方添加

1. RPAFenable On
2. RPAFsethostname On
3. RPAFproxy\_ips 127.0.0.1 192.168.1.101 192.168.102

**#填写Nginx所在的内网IP，Nginx的内网地址必写，不然一样失败的，这问题花了几个小时测试；有几个代理服务器的IP就写几个代理服务器的IP**

RPAFheader X-Forwarded-For

保存退出后重启apache，再看看 apache 的日志内容？不再是来来去去的那几个IP了吧，呵呵。

另外这里来个小插曲，我做的某个小项目本为是基于Nginx的1+3架构，突然要加一台机器是windows2003系统，专门作存放图片及 PDF等，但项目的要求是能在nginx后的三台web上有显示图片及pdf下载的需求；当时迷糊了下，因为程序是用到的Zend Framwork，所以一直用正则作跳转；后来才想明白，IE程序是先在nginx负载均衡器上提申请，所以nginx.conf是做分发而非正则跳转，此时最前端的nginx，既是负载匀衡器也是反向代理，明白这个就好做多了，语法如下；另外注意location /StockInfo与location ~^/StockInfo的差异性，Nginx默认的是正则优先的，by the way,proxy\_pass支持直接写IP的方式。

1. upstream mysrv {
2. ip\_hash;
3. server 192.168.110.62;
4. server 192.168.110.63;
5. }
7. upstream myjpg {
8. server 192.168.110.3:88;
9. }

12. server
13. {
14. listen 80;
15. server\_name web.tfzq.com;
16. proxy\_redirect off;
18. location ~ ^/StockInfo{
19. proxy\_pass http://myjpg;
20. }

再说下Nginx下的并发，这是个容易让人误会的概念。现在Nginx的文章满天飞，好像只要一涉及到web并发，就非将Apache换成nginx不可，其实完全没这必要；在内存足够的情况，Apache的抗并发能力也是很强的。玩了几年nginx了，遇到的最大并发也是以前在北京维护的CDN之广告网站，大约在3000-5000之间(这种情况建议用Nginx)，一般的资讯类金融网站也就100多，电子商务网站1100左右，web层的并发压力并没有想象中的大；相反，我感觉文件和数据层的压力越来越大，单个NFS服务器越来越难受了，所以我后期准备布署moosefs；而mysql数据库我一般用的是主从复制，压力也不小，目前只是从二方面来解决此问题：一、用公司最好的服务器来作数据库服务器；二、尽可能的优化，如果压力持续增长的话，后期我考虑从架构级方面优化了。对于一个网站而言，建议多从架构极的观念来看问题和解决问题