参考：<http://blog.csdn.net/yinwenjie/article/details/46620711>

[**架构设计：负载均衡层设计方案（2）——Nginx安装**](http://blog.csdn.net/yinwenjie/article/details/46620711)

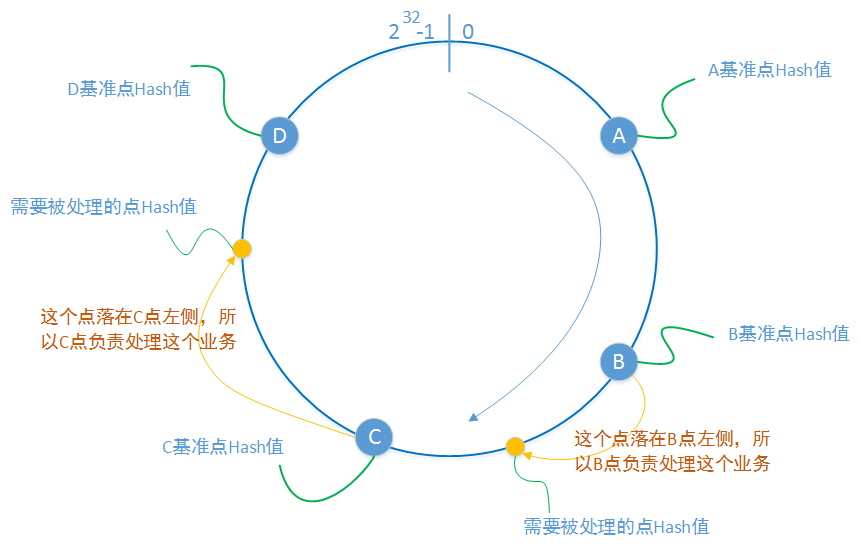
前一篇文章[《架构设计：负载均衡层设计方案（1）——负载场景和解决方式》](http://blog.csdn.net/yinwenjie/article/details/46605451)中我们描述了要搭设负载均衡层的业务场景和负载均衡层搭建和扩展思路。从这篇文章开始的后几篇文章，我们将详细介绍Nginx、LVS和Nginx+Keepalived、LVS+Keepalived和LVS+Nginx+Keepalived的安装细节，以及它们的性能优化方式。

Nginx和LVS都是可以独立工作的，Keepalived作为检测机制，不但可以和Nginx、LVS集成也可以和其他软件集成形成高可用方案（例如可以和[**MySQL数据库**](http://lib.csdn.net/base/mysql)集成、可以和Jetty服务器集成、还可以和自己写的程序集成）。所以首先我们先来详细讲述Nginx和LVS的核心工作原理、安装过程和优化方式，再分别讲解他们和Keepalived的集成方式。这样的方式应该可以使您更快的掌握其中的核心，并能最快的融会贯通。

# 1、Nginx重要算法介绍

Nginx是什么，请自行百度。我们先介绍几个关键的[**算法**](http://lib.csdn.net/base/datastructure)，如果您还不了解这些算法在Nginx中所起的作用，请不要着急，本文后半部分将说明它们的作用。

## 1.1、一致性Hash算法



一致性Hash算法是现代系统[**架构**](http://lib.csdn.net/base/architecture)中的最关键算法之一，在分布式计算系统、分布式存储系统、数据分析等众多领域中广泛应用。针对这个系列的博文，在负载均衡层、业务通信层、数据存储层都会有它的身影。

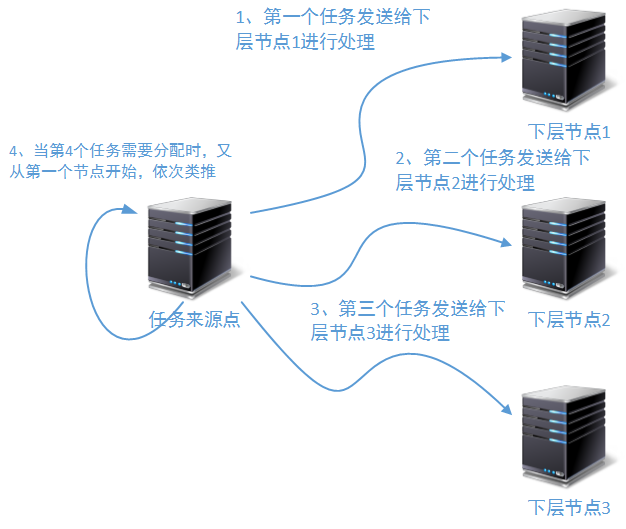
* hash算法的关键在于它能够根据不同的属性数据，生成一串不相同的hash值，并且能够将这个hash值转换为

0—232−1（32为32位计算机）

范围整数（即上图中的圆环）

* 一台服务器的某个或者某一些属性当然也可以进行hash计算（通常是这个服务器的IP地址和开放端口），并且根据计算分布在这个圆环上的某一个点。也就是图中圆环上的蓝色点。
* 一个处理请求当然也可以根据这个请求的某一个或者某一些属性进行hash计算（可以是这个请求的IP、端口、cookie值、URL值或者请求时间等等），并且根据计算结果分布在这个圆环上的某一个点上。也就是上图圆环上的黄色点。
* 我们约定落在某一个蓝点A左侧和蓝点B右侧的黄色点所代表的请求，都有蓝点A所代表的服务器进行处理，这样就完成解决了“谁来处理”的问题。在蓝色点稳定存在的前提下，来自于同一个Hash约定的请求所落在的位置都是一样的，这就保证了服务处理映射的稳定性。
* 当某一个蓝色点由于某种原因下线，其所影响到的黄色点也是有限的。即下一次客户端的请求将由其他的蓝色点所代表的服务器进行处理。

## 1.2、轮询与加权轮询

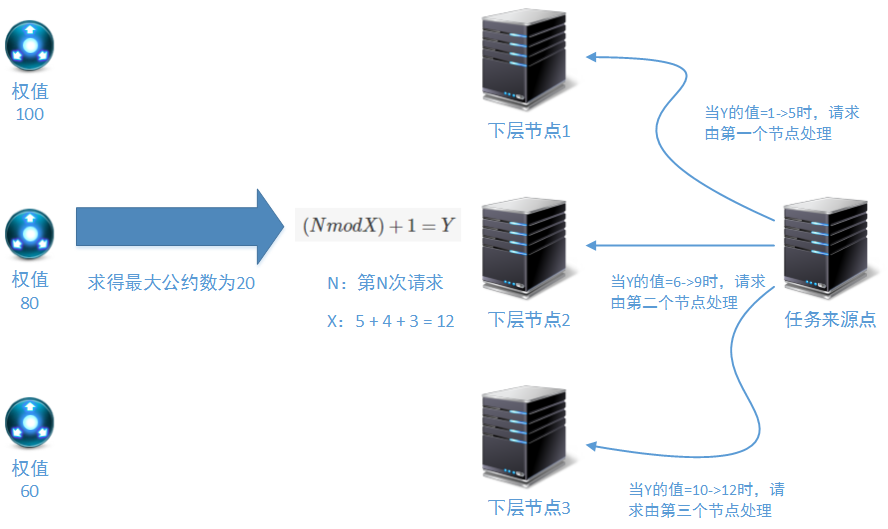


* 当有任务需要传递到下层节点进行处理时，任务来源点会按照一个固定的顺序（me：对应一致性哈希中圆环的顺时针或者逆时针），将任务依次分配到下层节点，如果下层可用的节点数量为X，那么第N个任务的分配规则为：

目标节点=(N mod X)+1

* 轮询处理在很多架构思想中都有体现：DNS解析多IP时、LVS向下转发消息时、Nginx向下转发消息时、Zookeeper向计算节点分配任务时。了解基本的轮询过程有助于我们在进行软件架构设计时进行思想借鉴。
* 但是上面的轮询方式是有缺陷的，由于各种客观原因我们可能无法保证任务处理节点的处理能力都是一样的（CPU、IO、内存频率等不同）。所以A节点业务能同时处理100个任务，但是B节点可能同时只能处理50个任务。
* 在这种情况下我们需要依据下层节点某个或者多个属性设置权值。这个属性可能是网络带宽、CPU繁忙程度或者就是各一个固定的权值。

那么加权轮询的分配依据是什么呢？有很多分配依据，例如：概率算法（此算法中包括蒙特卡罗算法，拉斯维加斯算法和舍伍德算法，在网络上有很多介绍资料）、最大公约数法。这里我们对最大公约数算法进行介绍，因为该方法简单实用：



* 首先按照某种规则计算得到每个处理节点的权值，上文已经说到计算规则可能是这个服务节点的CPU利用率、网络占用情况或者在配置文件中的固定权重。
* 求这些权值的最大公约数，在上图中三个节点的权值分别是100、80、60.那么求得的最大公约数就是20（如果您忘记了最大公约数的定义，请自行复习）。那么这三个节点的被除结果分别是5、4、3，求和值为12。
* 得到以上的计算结果，就可以开始进行请求分配了，公式同样为：

(N mod X)+1=Y

其中N表示当前的第N次任务；X表示整除后的求和结果；Y为处理节点。

总结一下：加权轮询是轮询方案的补充，通过将处理节点的属性转换成权值可以有效的描述处理节点的处理能力，实现更科学的处理任务分配。加权轮询的关键在于加权算法，最大公约数算法简单实用，定位效率高。

# 2、Nginx的安装

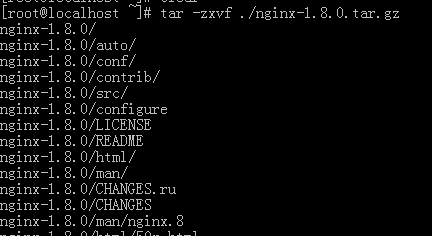
## 2.1、准备工作

[**操作系统**](http://lib.csdn.net/base/operatingsystem)：centOS 6.5。

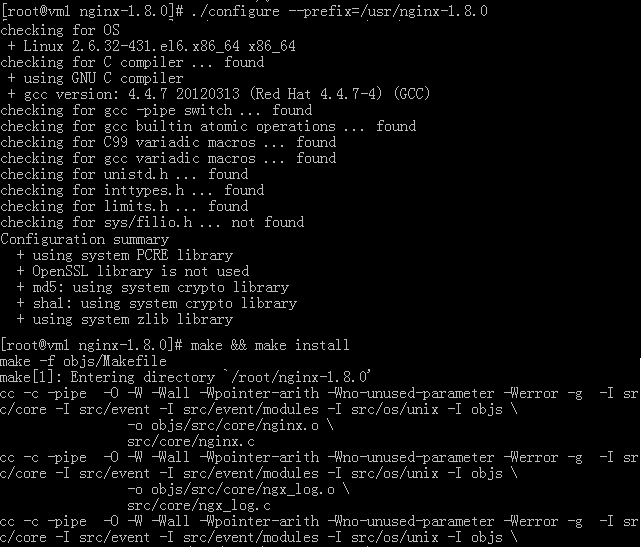
Nginx的下载地址：<http://nginx.org/en/download.html>。请下载stable的版本1.8.0。后续Nginx肯定还会有升级，官网上面会持续更新stable version。

最小必备组件：   
yum -y install make zlib zlib-devel gcc gcc-c++ ssh libtool

## 2.2、正式安装

* 下载nginx1.8.0版本   
  
* 解压nginx的tar文件   
  
* 进行源码编译   
  

我们看到这时编译检查报错，报错写得很清楚，为了支持HTTP重写模块，Nginx需要PCRElib的支持。那我们到<http://ncu.dl.sourceforge.net/project/pcre/pcre/8.37/pcre-8.37.tar.gz>下载一个稳定的pcre版本编译安装即可（不一定是8.37版本）。

* 再进行源码编译安装   
  ./configure –prefix=/usr/nginx-1.8.0   
  make && make install   
  
* 整个验证、编译、安装过程不应该报任何错误。如果您使用prefix设置了安装目标目录，那么可能您还需要在/etc/profix文件中设置环境变量：   
  这里写图片描述

## 2.3、安装验证和启动

下面介绍几个nginx常用的命令，如果您可以正常使用这些命令，那么说明nginx已经安装成功了。

nginx：直接在命令行键入nginx，就可以启动nginx。

nginx -t：检查配置文件是否正确。这个命令可以检查nginx.conf配置文件其格式、语法是否正确。如果配置文件存在错误，则会出现相应提示；如果nginx.conf文件正确，也会出现相应的成功提示。

nginx -s reload：重加载/重启nginx——以新的nginx.conf配置文件中的定义。

nginx -s stop：停止nginx。

外部访问nginx服务器时可能需要关闭防火墙。

# 3、进阶

Nginx在安装完成后，不用更改任何配置信息就可以直接运行。但是很显然这不会满足我们生产环境的要求。所以我们要重点介绍Nginx的配置文件，以及其中重要的配置项的含义。

## 3.1、重要配置项

如果您是按照本文的描述方式安装的Nginx，那么Nginx的主配置文件在：/usr/nginx-1.8.0/conf/nginx.conf的位置，如果您在编译安装的时候并没有指定安装目录，那么Nginx的主配置文件在：/usr/local/nginx/conf/nginx.conf的位置。

当然您还可以在启动Nginx的时候使用 –c（config） 的参数指定Nginx的配置文件位置（但是这种方式不建议）。

使用nginx –h 查看帮助

nginx –v 查看nginx的版本

nginx –V 查看安装的模块

我们重新整理了Nginx的配置文件，将其分块，以便于讲解：

#================================以下是全局配置项

#指定运行nginx的用户和用户组，默认情况下该选项关闭（关闭的情况就是nobody）

#user nobody nobody;

#运行nginx的进程数量，后文详细讲解

worker\_processes 1;

#nginx运行错误的日志存放位置。当然您还可以指定错误级别

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#指定主进程id文件的存放位置，虽然worker\_processes != 1的情况下，会有很多进程，管理进程只有一个

#pid logs/nginx.pid;

events {

#每一个进程可同时建立的（最大）连接数量，后问详细讲解

worker\_connections 1024;

#连接规则，可以采用[kqueue rtsig epoll select poll eventport ]，后文详细讲解

use epoll;

}

#================================以上是全局配置项

http {

#================================以下是Nginx后端服务配置项

upstream backendserver1 {

#nginx向后端服务器分配请求任务的方式，默认为轮询；如果指定了ip\_hash，就是hash算法（上文介绍的算法内容）

#ip\_hash

#后端服务器 ip:port ，如果有多个服务节点，这里就配置多个

server 192.168.220.131:8080;

server 192.168.220.132:8080;

#backup表示，这个是一个备份节点，只有当所有节点失效后，nginx才会往这个节点分配请求任务

#server 192.168.220.133:8080 backup;

#weight，固定权重，还记得我们上文提到的加权轮询方式吧。

#server 192.168.220.134:8080 weight=100;

}

#================================以上是Nginx后端服务配置项

#=================================================以下是 http 协议主配置

#安装nginx后，在conf目录下除了nginx.conf主配置文件以外，有很多模板配置文件，这里就是导入这些模板文件

include mime.types;

#HTTP核心模块指令，这里设定默认类型为二进制流，也就是当文件类型未定义时使用这种方式

default\_type application/octet-stream;

#日志格式

#log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

#连接日志文件存放的位置

#access\_log logs/access.log main;

#sendfile 规则开启

sendfile on;

#指定一个连接的等待时间（单位秒），如果超过等待时间，连接就会断掉。注意一定要设置，否则高并发情况下会产生性能问题。

keepalive\_timeout 65;

#开启数据压缩，后文详细介绍

gzip on;

#===============================================以上是 http 协议主配置（me：配置整个http协议）

#=================================================以下是一个服务实例的配置

server {

#这个代理实例的监听端口

listen 80;

#server\_name 取个唯一的实例名都要想半天？

server\_name localhost;

#文字格式

charset utf-8;

#access\_log logs/host.access.log main;

#location将按照规则分流满足条件的URL。"location /"您可以理解为“默认分流位置”（me：因为至少本规则都能匹配）。

location / {

#root目录，这个html表示nginx主安装目录下的“html”目录。

root html;

#目录中的默认展示页面

index index.html index.htm;

}

#location支持正则表达式，“~” 表示匹配正则表达式。

location ~ ^/business/ {

#反向代理。后文详细讲解。

proxy\_pass http://backendserver1;

}

#redirect server error pages to the static page /50x.html

#error\_page 404 /404.html;

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

#=================================================以上是一个服务实例的配置

}

以上是一个完整的nginx配置信息，可以直接粘贴使用，但不建议您这样做。因为上文中很多关键的配置属性都还没有讲解。这样粘贴复制不利于您的工作和学习。

### 3.1.1、use [ kqueue | rtsig | epoll | select | poll ]

科技在发展，社会在进步，满足摩尔定义的IT行业更是这样。以前所有的连接都是阻断式的（blocking I/O）也就是说一个TCP连接线程在发出Request后，代码就不会再往下执行了，直到得到远端的Response为止；服务器端也一样的，在处理完某一个客户端的Request之前，其他客户端的请求都会等待。这种处理方式使客户端和服务器端的通讯性能大打折扣。

很明显多线程貌似能够解决这个问题：一个线程处理不了，我可以再开线程来处理啊。但是多线程是有局限性的：

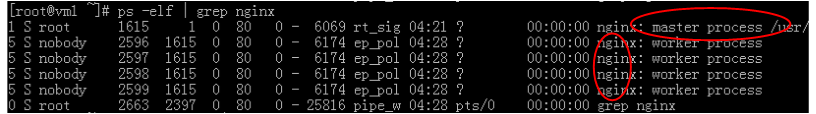
* 创建一个线程会消耗有限的资源。以[**Java**](http://lib.csdn.net/base/java)JVM为例，创建一个新的线程JVM会单独开放1MB的栈内存空间（通过-Xss参数可设置），虽然栈内存不受-Xmx和-Xms两个参数影响，但是可以说明线程的创建是需要消耗额外资源的（me：但是线程消耗的资源比创建进程的小）。
* 多线程工作时，计算机的CPU会耗费大量的资源让多线程在不同的状态下进行切换。在后续的文章中本书还会介绍依据这样的原理让计算机的CPU呈波形变化的编程方式。
* 在[**Linux**](http://lib.csdn.net/base/linux)操作系统下，单个用户能够创建的线程和进程总数、整个操作系统能够创建的线程总数都是有限的。通过limit -a命令，您可以查看相关的内核参数。
* 所以依靠线程来解决bio（me:big io）的问题是不靠谱的，只能起到缓解处理并行请求的作用。您可以想象一次并发10万个处理请求的问题，是不可能在计算机上同时创建10万个线程来解决的。

基于上面的描述，NIO（no blocking I/O）技术就这样诞生了。依靠event loop(me：事件流)机制（想看这个机制的详细分析，请持续关注我的博客^\_^），单个线程可以同时处理多个request请求，并在处理完产生response的时候，回调相关的远程事件。根据NIO实现机制的不同，技术名称也就不同了。我要说什么，您，应该懂了。

epoll、kqueue 等这些组件都是对多路复用网络I/O模型的实现，其中epoll是poll的升级版本，在linux环境下可以使用但限于linux2.6及以上版本。kqueue用在bsd上使用。

### 3.1.2、worker\_processes和worker\_connections

**worker\_processes**：操作系统启动多少个工作进程运行Nginx。注意是工作进程，不是有多少个nginx工程。在Nginx运行的时候，会启动两种进程，一种是主进程master process；一种是工作进程worker process。例如我在配置文件中将worker\_processes设置为4，启动Nginx后，使用进程查看命令观察名字叫做nginx的进程信息，我会看到如下结果：



图中可以看到1个nginx主进程，master process；还有四个工作进程，worker process。主进程负责监控端口，协调工作进程的工作状态，分配工作任务，工作进程负责进行任务处理。一般这个参数要和操作系统的CPU内核数成倍数。

**worker\_connections**：这个属性是指单个工作进程可以允许同时建立外部连接的数量（me：可见是进程来与外部连接，然后在开启线程来处理，所以web服务应该是多进程监听，进程再开启线程来处理用户的请求）。无论这个连接是外部主动建立的，还是内部建立的。这里需要注意的是，一个工作进程建立一个连接后，进程将打开一个文件副本。所以这个数量还受**操作系统设定的，进程最大可打开的文件数有关**。网上50%的文章告诉了您这个事实，并要求您修改worker\_connections属性的时候，一定要使用ulimit -n 修改操作系统对**进程**最大文件数的限制，但是这样更改只能在当次用户的当次shell会话中起作用，并不是永久了。接着您继续Google/百度，发现30%的文章还告诉您，要想使“进程最大可打开的文件数”永久有效，还需要修改/etc/security/limits.conf这个主配置文件，但是您应该如何正确检查“进程的最大可打开文件”的方式，却没有说。

下面本文告诉您全面的、正确的设置方式:

* **更改操作系统级别的“进程最大可打开文件数”的设置**：   
  首先您需要操作系统的root权限：

叫您的操作系统管理员给您。

修改limits.conf主配置文件

vim /etc/security/limits.conf

在主配置文件最后加入下面两句：

\* soft nofile 65535   
\* hard nofile 65535

注意“”是要加到文件里面的。这两句话的含义是soft（应用软件）级别限制的最大可打开文件数的限制，hard表示操作系统级别限制的最大可打开文件数的限制，“”表示所有用户都生效。保存这个文件（只有root用户能够有权限）。

保存这个文件后，配置是不会马上生效的，为了保证本次shell会话中的配置马上有效，我们需要通过ulimit命令更改本次的shell会话设置（当然您如果要重启系统，我也不能说什么）。

ulimit -n 65535

执行命令后，配置马上生效。您可以用ulimit -a 查看目前会话中的所有核心配置：

ulimit -a   
core file size (blocks, -c) 0   
data seg size (kbytes, -d) unlimited   
scheduling priority (-e) 0   
file size (blocks, -f) unlimited   
pending signals (-i) 7746   
max locked memory (kbytes, -l) 64   
max memory size (kbytes, -m) unlimited   
open files (-n) 65535   
pipe size (512 bytes, -p) 8   
POSIX message queues (bytes, -q) 819200   
real-time priority (-r) 0   
stack size (kbytes, -s) 10240   
cpu time (seconds, -t) unlimited   
max user processes (-u) 7746   
virtual memory (kbytes, -v) unlimited   
file locks (-x) unlimited

请注意open files这一项。

* **更改Nginx软件级别的“进程最大可打开文件数”的设置：**   
  刚才更改的只是操作系统级别的“进程最大可打开文件”的限制，作为Nginx来说，我们还要对这个软件进行更改。打开nginx.conf主配置文件。您需要配合worker\_rlimit\_nofile属性。如下：

user root root;   
worker\_processes 4;   
**worker\_rlimit\_nofile 65535;**

#error\_log logs/error.log;   
#error\_log logs/error.log notice;   
#error\_log logs/error.log info;

#pid logs/nginx.pid;   
events {   
use epoll;   
**worker\_connections 65535;**   
}

这里只粘贴了部分代码，其他的配置代码和主题无关，也就不需要粘贴了。请注意代码行中加粗的两个配置项，请一定两个属性全部配置。配置完成后，请通过nginx -s reload命令重新启动Nginx。

* **验证Nginx的“进程最大可打开文件数”是否起作用：**

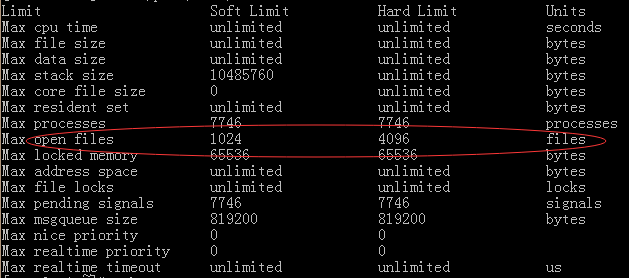
那么我们如何来验证配置是否起作用了呢？在linux系统中，所有的进程都会有一个临时的核心配置文件描述，存放路径在/pro/进程号/limit。

首先我们来看一下，没有进行参数优化前的进程配置信息：

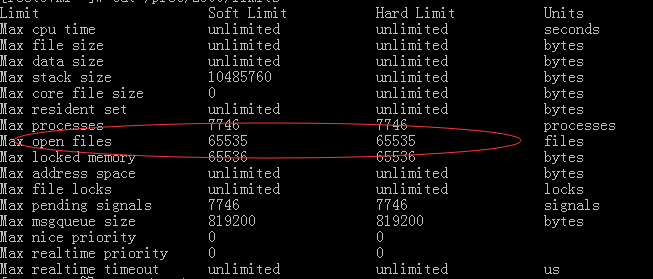
ps -elf | grep nginx   
1 S root 1594 1 0 80 0 - 6070 rt\_sig 05:06 ? 00:00:00 nginx: master process /usr/nginx-1.8.0/sbin/nginx   
5 S root 1596 1594 0 80 0 - 6176 ep\_pol 05:06 ? 00:00:00 nginx: worker process   
5 S root 1597 1594 0 80 0 - 6176 ep\_pol 05:06 ? 00:00:00 nginx: worker process   
5 S root 1598 1594 0 80 0 - 6176 ep\_pol 05:06 ? 00:00:00 nginx: worker process   
5 S root 1599 1594 0 80 0 - 6176 ep\_pol 05:06 ? 00:00:00 nginx: worker process

可以看到，nginx工作进程的进程号是：1596 1597 1598 1599。我们选择一个进程，查看其核心配置信息：

cat /proc/1598/limits



请注意其中的Max open files ，分别是1024和4096。那么更改配置信息，并重启Nginx后，配置信息就是下图所示了：



### 3.1.3、max client的计算方式：

这个小结我们主要来说明两个在网上经常说的公式：

* max\_client = worker\_processes \* worker\_connections
* max\_client = worker\_processes \* worker\_connections / 4

这两个公式分别说明，在Nginx充当服务器（例如nginx上面装载[**PHP**](http://lib.csdn.net/base/php)）的时候，Nginx可同时承载的连接数量是最大工作线程 \* 每个线程允许的连接数量；当Nginx充当反向代理服务的时候，其可同时承载的连接数量是最大工作线程 \* 每个线程允许的连接数量 / 4。

第一个问题很好理解，关键是第二个问题：为什么会除以4。网上的帖子给出的答案是。浏览器->Nginx、Nginx->后端服务器、后端服务器->Nginx、Nginx->浏览器，所以需要除以四，我想说TCP协议是双向全双工协议，为什么需要这样建立连接呢？所以这个说法肯定是错误的。

在nginx官方文档上有这样一句话：

Since a browser opens 2 connections by default to a server and nginx uses the fds (file descriptors) from the same pool to connect to the upstream backend。

翻译成中文的描述就是，浏览器会建立两条连接到Nginx（注意两条连接都是浏览器建立的），Nginx也会建立对应的两条连接到后端服务器（me：两条连接浏览器两条连接后端服务器）。这样就是四条连接了。

### 3.1.4、gzip

后文补讲，放在这里算是一个扣子^\_^

## 3.2、健康检查模块

后文补讲，放在这里算是一个扣子^\_^

## 3.3、图片处理模块

后文补讲，放在这里算是一个扣子^\_^

## 3.4、Nginx的Rewrite功能

后文补讲，放在这里算是一个扣子^\_^

# 4、后文介绍

我原本计划一篇文章就把Nginx的主要特性都进行介绍，奈何Nginx的强大功能确实太多了。为了保证您对知识的全面消化，这边文章就写到这里了。LVS的讲解再往后拖一周左右吧，下篇文章我们继续讲解Nginx的强大功能，包括gzip功能、强大的rewrite功能，以及两个扩展模块：健康检查模块和图片处理模块，至于Nginx集成PHP作为服务器的特性，为了保证这个系列文章的中心线络就不再讲了，那又是一套完整的知识体系。敬请期待我的下一篇博客，谢谢。另外，目前一周一篇文章的频率我觉得是比较合适的，后面的文章我争取保持这个速度。