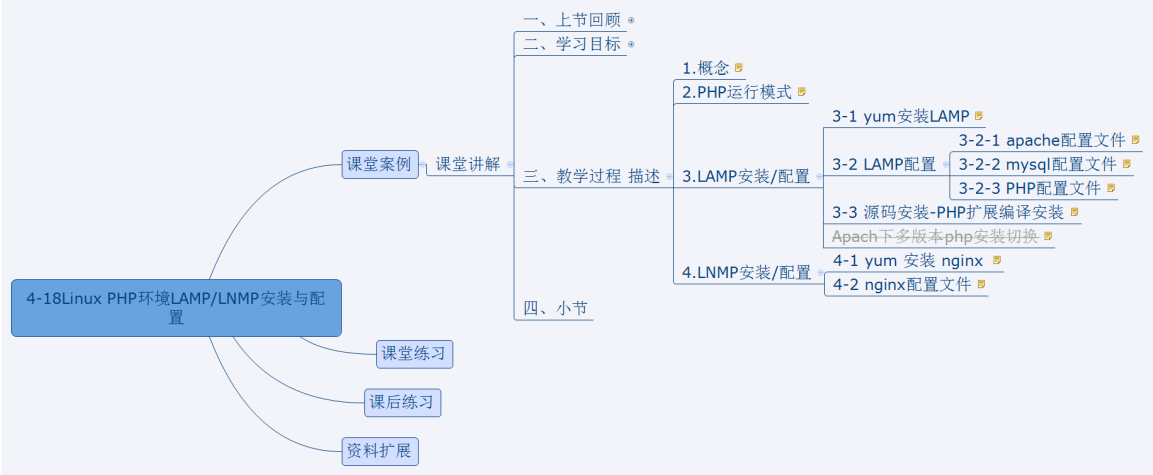
# 4-18Linux PHP环境LAMP/LNMP安装与配置




# 1. 课堂案例

## 课堂讲解

### 一、上节回顾

**iptables防火墙**

**管理管理的相关命令**

**服务管理的相关命令**

### 二、学习目标

**centos下 LAMP环境安装与配置**

**centos下LNMP 环境安装与配置**

### 三、教学过程 描述

#### 1.概念

Linux+Apache/Nginx+Mysql+PHP

一组常用来搭建动态网站或者服务器的开源软件，本身都是各自独立的程序，但是因为常被放在一起使用，拥有了越来越高的兼容度，共同组成了一个强大的Web应用程序平台

因此：LAMP/LNMP 是linux 下一系统软件的组合

为什么使用LAMP/LNMP：
流行、免费、开源、轻量

## 2.PHP运行模式

关于PHP目前比较常见的五大运行模式：

1）CGI（通用网关接口 / Common Gateway Interface）

2）FastCGI（常驻型CGI / Long-Live CGI）

3）CLI（命令行运行 / Command Line Interface）

4）Web模块模式（Apache等Web服务器运行的模式）

5）ISAPI（Internet Server Application Program Interface）

备注：在PHP5.3以后，PHP不再有ISAPI模式，安装后也不再有php5isapi.dll这个文件。要在IIS6上使用高版本PHP，必须安装FastCGI 扩展，然后使IIS6支持FastCGI。

CGI模式

□□CGI即通用网关接口（Common Gateway Interface），它是一段程序，通俗的讲CGI就象是一座桥，把网页和Web服务器中的执行程序连接起来，它把HTML接收的指令传递给服务器的执行程序，再把服务器执行程序的结果返还给HTML页。CGI的跨平台性能极佳，几乎可以在任何操作系统上实现。CGI已经是比较老的模式了，这几年都很少用了。

□□

每有一个用户请求，都会先要创建CGI的子进程，然后处理请求，处理完后结束这个子进程，这就是Fork-And-Execute模式。
当用户请求数量非常多时，会大量挤占系统的资源如内存，CPU时间等，造成效能低下。所以用CGI方式的服务器有多少连接请求就会有多少CGI子进程，子进程反复加载是CGI性能低下的主要原因。

□□如果不想把 PHP 嵌入到服务器端软件(如 Apache)作为一个模块安装的话，可以选择以 CGI 的模式安装。或者把 PHP 用于不同的 CGI 封装以便为代码创建安全的 chroot 和 setuid 环境。这样每个客户机请求一个PHP文件，Web服务器就调用php.exe(win下是php.exe,linux是php)去解释这个文件，然后再把解释的结果以网页的形式返回给客户机。 这种安装方式通常会把 PHP 的可执行文件安装到 web 服务器的 cgi-bin 目录。CERT 建议书 CA-96.11 建议不要把任何的解释器放到 cgi-bin 目录。

□□这种方式的好处是把Web
Server和具体的程序处理独立开来，结构清晰，可控性强，同时缺点就是如果在高访问需求的情况下，CGI的进程Fork就会成为很大的服务器负担，想
象一下数百个并发请求导致服务器Fork出数百个进程就明白了。这也是为什么CGI一直背负性能低下，高资源消耗的恶名的原因。

FastCGI模式

□□FastCGI是CGI的升级版本，FastCGI像是一个常驻 (long-live)型的 CGI，它可以一直执行着，只要激活后，不会每次都要花费时间去 Fork 一次（这是 CGI 最为人诟病的 fork-and-execute 模式）。

□□FastCGI是一个可伸缩地、高速地在HTTP server和动态脚本语言间通信的接口。多数流行的HTTP server都支持FastCGI，包括Apache、Nginx和lighttpd等，同时，FastCGI也被许多脚本语言所支持，其中就有PHP。

□□
FastCGI接口方式采用C/S结构，可以将HTTP服务器和脚本解析服务器分开，同时在脚本解析服务器上启动一个或者多个脚本解析守护进程。当HTTP服务器每次遇到动态程序时，可以将其直接交付给FastCGI进程来执行，然后将得到的结果返回给浏览器。这种方式可以让HTTP服务器专一地处理静态请求或者将动态脚本服务器的结果返回给客户端，这在很大程度上提高了整个应用系统的性能。

□□【原理】

1）Web Server启动时载入FastCGI进程管理器(IIS ISAPI或Apache Module)；

2）FastCGI进程管理器自身初始化，启动多个CGI解释器进程 (可见多个php-cgi.exe或php-cig)并等待来自Web Server的连接；

3）当客户端请求到达Web Server时，FastCGI进程管理器选择并连接到一个CGI解释器。Web server将CGI环境变量和标准输入发送到FastCGI子进程php-cgi；

4）FastCGI子进程完成处理后将标准输出和错误信息从同一连接返回Web Server。当FastCGI子进程关闭连接时，请求便告处理完成。FastCGI子进程接着等待并处理来自FastCGI进程管理器（运行在 WebServer中）的下一个连接。在正常的CGI模式中，php-cgi.exe在此便退出了。

□□在CGI模式中，你可以想象
CGI通常有多慢。每一个Web请求PHP都必须重新解析php.ini、重新载入全部dll扩展并重初始化全部数据结构。使用FastCGI，所有这些都只在进程启动时发生一次。一个额外的好处是，持续数据库连接（Persistent database connection）可以工作。

备注：PHP的FastCGI进程管理器是PHP-FPM（PHP-FastCGI Process Manager）

□□【优点】

1）从稳定性上看，FastCGI是以独立的进程池来运行CGI，单独一个进程死掉，系统可以很轻易的丢弃，然后重新分配新的进程来运行逻辑；

2）从安全性上看，FastCGI支持分布式运算。FastCGI和宿主的Server完全独立，FastCGI怎么down也不会把Server搞垮；

3）从性能上看，FastCGI把动态逻辑的处理从Server中分离出来，大负荷的IO处理还是留给宿主Server，这样宿主Server可以一心一意作IO，对于一个普通的动态网页来说，
逻辑处理可能只有一小部分，大量的是图片等静态。

□□【缺点】

□□
说完了好处，也来说说缺点。从我的实际使用来看，用FastCGI模式更适合生产环境的服务器。但对于开发用机器来说就不太合适。因为当使用 Zend Studio调试程序时，由于 FastCGI会认为PHP进程超时，从而在页面返回 500错误。这一点让人非常恼火，所以我在开发机器上还是换回了ISAPI模式。对某些服务器的新版本支持不好，对分布式负载均衡没要求的模块化安装是否是更好的选择。目前的FastCGI和Server沟通还不够智能，一个FastCGI进程如果执行时间过长会被当成是死进程杀掉重起，这样在处理长时间任务的时候很麻烦，这样做也使得FastCGI无法允许联机调试。因为是多进程

，所以比CGI多线程消耗更多的服务器内存，PHP-
CGI解释器每进程消耗7至25兆内存，将这个数字乘以50或100就是很大的内存数。

CLI模式

□□PHP-CLI是PHP Command Line Interface的简称，如同它名字的意思，就是PHP在命令行运行的接口，区别于在Web服务器上运行的PHP环境（PHP-CGI，ISAPI等）。 也就是说，PHP不单可以写前台网页，它还可以用来写后台的程序。
PHP的CLI
Shell脚本适用于所有的PHP优势，使创建要么支持脚本或系统甚至与GUI应用程序的服务端，在Windows和Linux下都是支持PHP-CLI模式的。

□□【优点】

1）使用多进程，子进程结束以后，内核会负责回收资源；

2）使用多进程，子进程异常退出不会导致整个进程Thread退出，父进程还有机会重建流程；

3）一个常驻主进程，只负责任务分发，逻辑更清楚。

□□我们在Linux下经常使用"php –
m"查找PHP安装了那些扩展就是PHP命令行运行模式；有兴趣的同学可以输入"php –
h"去深入研究该运行模式。

模块模式

□□
模块模式是以mod_php5模块的形式集成，此时mod_php5模块的作用是接收Apache传递过来的PHP文件请求，并处理这些请求，然后将处理后的结果返回给Apache。如果我们在Apache启动前在其配置文件中配置好了PHP模块（mod_php5），
PHP模块通过注册apache2的ap_hook_post_config挂钩，在Apache启动的时候启动此模块以接受PHP文件的请求。

□□
除了这种启动时的加载方式，Apache的模块可以在运行的时候动态装载，这意味着对服务器可以进行功能扩展而不需要重新对源代码进行编译，甚至根本不需要停止服务器。我们所需要做的仅仅是给服务器

发送信号HUP或者AP_SIG_GRACEFUL通知服务器重新载入模块。但是在动态加载之前，我们需要将模块编译成为动态链接库。此时的动态加载就是加载动态链接库。
Apache中对动态链接库的处理是通过模块mod_so来完成的，因此mod_so模块不能被动态加载，它只能被静态编译进Apache的核心。这意味着它是随着Apache一起启动的。

□□
Apache是如何加载模块的呢？我们以前面提到的mod_php5模块为例。首先我们需要在Apache的配置文件httpd.conf中添加一行：

LoadModule php5_module modules/mod_php5.so

□□
这里我们使用了LoadModule命令，该命令的第一个参数是模块的名称，名称可以在模块实现的源码中找到。第二个选项是该模块所处的路径。如果需要在服务器运行时加载模块，可以通过发送信号HUP或者AP_SIG_GRACEFUL给服务器，一旦接受到该信号，Apache将重新装载模块，而不需要重新启动服务器。

□□
该运行模式是我们以前在windows环境下使用apache服务器经常使用的，而在模块化（DLL）中，PHP是与Web服务器一起启动并运行的。（它是apache在CGI的基础上进行的一种扩展，加快PHP的运行效率）。

ISAPI模式

□□ISAPI（Internet Server Application Program Interface）是微软提供的一套面向Internet服务的API接口，一个ISAPI的DLL，可以在被用户请求激活后长驻内存，等待用户的另一个请求，还可以在一个DLL里设置多个用户请求处理函数，此外，ISAPI的DLL应用程序和WWW服务器处于同一个进程中，效率要显著高于CGI。（由于微软的排他性，只能运行于windows环境）

□□
PHP作为Apache模块，Apache服务器在系统启动后，预先生成多个进程副本驻留在内存中，一旦有请求出现，就立即使用这些空余的子进程进行处理，这样就不存在生成子进程造成的延迟了。这些服务器副本在处理完一次HTTP请求之后并不立即退出，而是停留在计算机中等待下次请求。对于客户浏览器的请求反应更快，性能较高。

## 3.LAMP安装/配置

### 3-1 yum安装LAMP

yum安装和源代码编译在使用的时候没啥区别，但是安装的过程就大相径庭了，yum只需要3个命令就可以完成，源代码需要13个包，还得加压编译，步骤很麻烦，而且当做有时候会出错，源代码编译安装大概需要2个小时，好处在于可以自己配置地址等一些参数，yum安装半个小时搞定，一般不会出错，更新也很方便。

一般机器都带yum命令，并且yum包源都是可以用的，就是说不用你自己下载东西，直接yum -y install 后面加上你所需要安装的软件，他会自动下载自动安装，非常方便。例如 yum -y install httpd自动下载并安装apache服务器。

lamp环境只需要安装httpd,mysql,php

首先更新一下
yum -y update

用yum安装Apache,Mysql,PHP.

1. 安装Apache
yum install httpd httpd-devel
安装完成后，用/etc/init.d/httpd start 启动apache
设为开机启动:chkconfig httpd on

2 . 安装mysql
yum install mysql mysql-server mysql-devel

同样，完成后，用/etc/init.d/mysqld start 启动mysql

设置mysql密码
mysql>; USE mysql;
mysql>; UPDATE user SET Password=PASSWORD('newpassword') WHERE user='root';
mysql>; FLUSH PRIVILEGES;

设置mysql密码还可以用：mysql_secure_installation 命令

允许远程登录
mysql -u root -p
Enter Password: <your new password>
mysql>GRANT ALL PRIVILEGES ON *.* TO '用户名'@'%' IDENTIFIED BY '密码' WITH GRANT OPTION;
完成后就能用mysql-front远程管理mysql了。
设为开机启动
chkconfig mysqld on

3. 安装php
yum install php php-mysql php-common php-gd php-mbstring php-mcrypt php-devel php-xml
/etc/init.d/httpd start

4. 测试一下
在/var/www/html/新建个test.php文件，将以下内容写入，然后保存。
<?
phpinfo();
?>

5. 防火墙配置
a.添加.允许访问端口{80: http}.
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT

b.关闭防火墙{不推荐}.
service iptables stop

c.重置加载防火墙
service iptables restart

6. 然后在客户端浏览器里打开http://serverip/test.php，若能成功显示，则表示安装成功

至此，安装完毕。

## 3-2 LAMP配置

### 3-2-1 apache配置文件

Apache的主配置文件：/etc/httpd/conf/httpd.conf
默认站点主目录：/var/www/html/

常见操作：
新建虚拟目录 demo 演示
多IP配置：demo 演示
多域名配置 demo 演示

配置文件包括三部分：

1: Global Environment 全局配置
2: 'Main' server configuration 主服务配置
3: Virtual Hosts 虚拟机配置

常见需要关注的配置：
ServerRoot "/etc/httpd" #用于指定Apache的运行目录

PidFile run/httpd.pid #记录httpd守护进程的pid号码

Timeout 60 #服务器与客户端断开的时间

KeepAlive Off

#是否持续连接(因为每次连接都得三次握手，如果是访问量不大，建议打开此项，如果网站访问量比较大关闭此项比较好)，修改为：KeepAlive On 表示允许程序性联机

MaxKeepAliveRequests 100　　#表示一个连接的最大请求数

KeepAliveTimeout 15　　#断开连接前的时间

<IfModule prefork.c>
　StartServers　　8
　MinSpareServers　5
　MaxSpareServers　20
　ServerLimit　　256
　MaxClients　　256
　MaxRequestsPerChild　4000
</IfModule>

系统默认的模块，表示为每个访问启动一个进程(即当有多个连接公用一个进程的时候，在同一时刻只能有一个获得服务)。

StartServer　开始服务时启动8个进程，

MinSpareServers最小空闲5个进程，

MaxSpareServers最多空闲20个进程。

MaxClient限制同一时刻客户端的最大连接请求数量超过的要进入等候队列。

MaxRequestsPerChild每个进程生存期内允许服务的最大请求数量，0表示永不结束

<IfModule worker.c>
　StartServers　　4
　MaxClients　　300
　MinSpareThreads　25
　MaxSpareThreads　75
　ThreadsPerChild　25
　MaxRequestsPerChild　0
</IfModule>

为Apache配置线程访问，即每对WEB服务访问启动一个线程，这样对内存占用率比较小。

ServerLimit服务器允许配置进程数的上限。

ThreadLimit每个子进程可能配置的线程上限

StartServers启动两个httpd进程，

MaxClients同时最多能发起250个访问，超过的要进入队列等待，其大小有ServerLimit和ThreadsPerChild的乘积决定

ThreadsPerChild每个子进程生存期间常驻执行线程数，子线程建立之后将不再增加

MaxRequestsPerChild每个进程启动的最大线程数，如达到限制数时进程将结束，如置为0则子线程永不结束

Listen 80　　#监听的端口，如有多块网卡，默认监听所有网卡

Include conf.d/*.conf　　#加载的配置文件

User apache　　#以什么用户运行

Group apache　　#以什么用户组运行

## 3-2-2 mysql配置文件

```
[client]
port = 3306
socket = /tmp/mysql.sock
[mysqld]
port = 3306
socket = /tmp/mysql.sock
basedir = /usr/local/mysql
datadir = /data/mysql
pid-file = /data/mysql/mysql.pid
user = mysql
bind-address = 0.0.0.0
```

server-id = 1 #表示是本机的序号为1,一般来讲就是master的意思
skip-name-resolve
#
禁止MySQL对外部连接进行DNS解析，使用这一选项可以消除MySQL进行DNS解析的时间。但需要注意，如果开启该选项，
# 则所有远程主机连接授权都要使用IP地址方式，否则MySQL将无法正常处理连接请求
#skip-networking

back_log = 600
#
MySQL能有的连接数量。当主要MySQL线程在一个很短时间内得到非常多的连接请求，这就起作用，
#
然后主线程花些时间(尽管很短)检查连接并且启动一个新线程。back_log值指出在MySQL暂时停止回答新请求之前的短时间内多少个请求可以被存在堆栈中。
#
如果期望在一个短时间内有很多连接，你需要增加它。也就是说，如果MySQL的连接数据达到max_connections时，新来的请求将会被存在堆栈中，
#
以等待某一连接释放资源，该堆栈的数量即back_log，如果等待连接的数量超过back_log，将不被授予连接资源。
# 另外，这值(back_log)限于您的操作系统对到来的TCP/IP连接的侦听队列的大小。
#
你的操作系统在这个队列大小上有它自己的限制(可以检查你的OS文档找出这个变量的最大值)，试图设定back_log高于你的操作系统的限制将是无效的。

max_connections = 1000
#
MySQL的最大连接数，如果服务器的并发连接请求量比较大，建议调高此值，以增加并行连接数量，当然这建立在机器能支撑的情况下，因为如果连接数越多，介于MySQL会为每个连接提供连接缓冲区，就会开销越多的内存，所以要适当调整该值，不能盲目提高设值。可以过'conn%'通配符查看当前状态的连接数量，以定夺该值的大小。

max_connect_errors = 6000
#
对于同一主机，如果有超出该参数值个数的中断错误连接，则该主机将被禁止连接。如需对该主机进行解禁，执行：FLUSH HOST。

open_files_limit = 65535
#
MySQL打开的文件描述符限制，默认最小1024;当open_files_limit没有被配置的时候，比较max_connections*5和ulimit -n的值，哪个大用哪个，

# 当open_file_limit被配置的时候，比较open_files_limit和max_connections*5的值，哪个大用哪个。

table_open_cache = 128
#
MySQL每打开一个表，都会读入一些数据到table_open_cache缓存中，当MySQL在这个缓存中找不到相应信息时，才会去磁盘上读取。默认值64
# 假定系统有200个并发连接，则需将此参数设置为200*N(N为每个连接所需的文件描述符数目)；
#
当把table_open_cache设置为很大时，如果系统处理不了那么多文件描述符，那么就会出现客户端失效，连接不上

max_allowed_packet = 4M
#
接受的数据包大小；增加该变量的值十分安全，这是因为仅当需要时才会分配额外内存。例如，仅当你发出长查询或MySQLd必须返回大的结果行时MySQLd才会分配更多内存。
#
该变量之所以取较小默认值是一种预防措施，以捕获客户端和服务器之间的错误信息包，并确保不会因偶然使用大的信息包而导致内存溢出。

binlog_cache_size = 1M
#
一个事务，在没有提交的时候，产生的日志，记录到Cache中；等到事务提交需要提交的时候，则把日志持久化到磁盘。默认

binlog_cache_size大小32K
max_heap_table_size = 8M
# 定义了用户可以创建的内存表(memory table)的大小。这个值用来计算内存表的最大行数值。这个变量支持动态改变

tmp_table_size = 16M
#
MySQL的heap(堆积)表缓冲大小。所有联合在一个DML指令内完成，并且大多数联合甚至可以不用临时表即可以完成。
# 大多数临时表是基于内存的(HEAP)表。具有大的记录长度的临时表(所有列的长度的和)或包含BLOB列的表存储在硬盘上。
#
如果某个内部heap(堆积)表大小超过tmp_table_size，MySQL可以根据需要自动将内存中的heap表改为基于硬盘的MyISAM表。还可以通过设置tmp_table_size选项来增加临时表的大小。也就是说，如果调高该值，MySQL同时将增加heap表的大小，可达到提高联接查询速度的效果

read_buffer_size = 2M
#
MySQL读入缓冲区大小。对表进行顺序扫描的请求将分配一个读入缓冲区，MySQL会为它分配一段内存缓冲区。

read_buffer_size变量控制这一缓冲区的大小。
#
如果对表的顺序扫描请求非常频繁，并且你认为频繁扫描进行得太慢，可以通过增加该变量值以及内存缓冲区大小提高其性能

read_rnd_buffer_size = 8M
#
MySQL的随机读缓冲区大小。当按任意顺序读取行时(例如，按照排序顺序)，将分配一个随机读缓存区。进行排序查询时，
#
MySQL会首先扫描一遍该缓冲，以避免磁盘搜索，提高查询速度，如果需要排序大量数据，可适当调高该值。但MySQL会为每个客户连接发放该缓冲空间，所以应尽量适当设置该值，以避免内存开销过大

sort_buffer_size = 8M

# MySQL执行排序使用的缓冲大小。如果想要增加ORDER BY的速度，首先看是否可以让MySQL使用索引而不是额外的排序阶段。

# 如果不能，可以尝试增加sort_buffer_size变量的大小

join_buffer_size = 8M

#

联合查询操作所能使用的缓冲区大小，和sort_buffer_size一样，该参数对应的分配内存也是每连接独享

thread_cache_size = 8

#

这个值(默认8)表示可以重新利用保存在缓存中线程的数量，当断开连接时如果缓存中还有空间，那么客户端的线程将被放到缓存中，

#

如果线程重新被请求，那么请求将从缓存中读取,如果缓存中是空的或者是新的请求，那么这个线程将被重新创建,如果有很多新的线程，

#

增加这个值可以改善系统性能.通过比较Connections和Threads_created状态的变量，可以看到这个变量的作用。(–>表示要调整的值)

# 根据物理内存设置规则如下:

# 1G —> 8

# 2G —> 16

# 3G —> 32

# 大于3G —> 64

query_cache_size = 8M

#MySQL的查询缓冲大小(从4.0.1开始，MySQL提供了查询缓冲机制)使用查询缓冲，MySQL将SELECT语句和查询结果存放在缓冲区中，

#

今后对于同样的SELECT语句(区分大小写)，将直接从缓冲区中读取结果。根据MySQL用户手册，使用查询缓冲最多可以达到238%的效率。

#
通过检查状态值'Qcache_%'，可以知道query_cache_size设置是否合理：如果Qcache_lowmem_prunes的值非常大，则表明经常出现缓冲不够的情况，
#
如果Qcache_hits的值也非常大，则表明查询缓冲使用非常频繁，此时需要增加缓冲大小；如果Qcache_hits的值不大，则表明你的查询重复率很低，
#
这种情况下使用查询缓冲反而会影响效率，那么可以考虑不用查询缓冲。此外，在SELECT语句中加入SQL_NO_CACHE可以明确表示不使用查询缓冲

query_cache_limit = 2M
#指定单个查询能够使用的缓冲区大小，默认1M

key_buffer_size = 4M
#指定用于索引的缓冲区大小，增加它可得到更好处理的索引(对所有读和多重写)，到你能负担得起那样多。如果你使它太大，
#
系统将开始换页并且真的变慢了。对于内存在4GB左右的服务器该参数可设置为384M或512M。通过检查状态值

Key_read_requests和Key_reads，
# 可以知道key_buffer_size设置是否合理。比例key_reads/key_read_requests应该尽可能的低，
#　　至少是1:100，1:1000更好(上述状态值可以使用SHOW　　STATUS　　LIKE　　'key_read%'获得)。注意：该参数值设置的过大反而会是服务器整体效率降低

ft_min_word_len = 4
# 分词词汇最小长度，默认4

transaction_isolation = REPEATABLE-READ
# MySQL支持4种事务隔离级别，他们分别是：
# READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, SERIALIZABLE.

# 如没有指定，MySQL默认采用的是REPEATABLE-READ，ORACLE默认的是READ-COMMITTED

log_bin = mysql-bin
binlog_format = mixed
expire_logs_days = 30 #超过30天的binlog删除
log_error = /data/mysql/mysql-error.log #错误日志路径
slow_query_log = 1
long_query_time = 1 #慢查询时间 超过1秒则为慢查询
slow_query_log_file = /data/mysql/mysql-slow.log
performance_schema = 0
explicit_defaults_for_timestamp
#lower_case_table_names = 1 #不区分大小写

skip-external-locking #MySQL选项以避免外部锁定。该选项默认开启
default-storage-engine = InnoDB #默认存储引擎

innodb_file_per_table = 1
# InnoDB为独立表空间模式，每个数据库的每个表都会生成一个数据空间
# 独立表空间优点：
# 1. 每个表都有自己独立的表空间。
# 2. 每个表的数据和索引都会存在自己的表空间中。
# 3. 可以实现单表在不同的数据库中移动。
# 4. 空间可以回收（除drop table操作处，表空不能自己回收）
# 缺点：
# 单表增加过大，如超过100G
# 结论：
#
共享表空间在Insert操作上少有优势。其它都没独立表空间表现好。当启用独立表空间时，请合理调整：innodb_open_files

innodb_open_files = 500
# 限制Innodb能打开的表的数据，如果库里的表特别多的情况，请增加这个。这个值默认是300
innodb_buffer_pool_size = 64M
# InnoDB使用一个缓冲池来保存索引和原始数据, 不像MyISAM.
# 这里你设置越大,你在存取表里面数据时所需要的磁盘I/O越少.

# 在一个独立使用的数据库服务器上,你可以设置这个变量到服务器物理内存大小的80%
# 不要设置过大,否则,由于物理内存的竞争可能导致操作系统的换页颠簸.
# 注意在32位系统上你每个进程可能被限制在 2-3.5G 用户层面内存限制,
# 所以不要设置的太高.

innodb_write_io_threads = 4
innodb_read_io_threads = 4
# innodb使用后台线程处理数据页上的读写 I/O(输入输出)请求,根据你的 CPU 核数来更改,默认是4
#
注:这两个参数不支持动态改变,需要把该参数加入到my.cnf里，修改完后重启MySQL服务,允许值的范围从 1-64

nnodb_thread_concurrency = 0
# 默认设置为 0,表示不限制并发数，这里推荐设置为0，更好去发挥CPU多核处理能力，提高并发量

innodb_purge_threads = 1
#
InnoDB中的清除操作是一类定期回收无用数据的操作。在之前的几个版本中，清除操作是主线程的一部分，这意味着运行时它可能会堵塞其它的数据库操作。
#
从MySQL5.5.X版本开始，该操作运行于独立的线程中,并支持更多的并发数。用户可通过设置innodb_purge_threads配置参数来选择清除操作是否使用单
# 独线程,默认情况下参数设置为0(不使用单独线程),设置为 1 时表示使用单独的清除线程。建议为1

innodb_flush_log_at_trx_commit = 2
# 0：如果innodb_flush_log_at_trx_commit的值为0,log buffer每秒就会被刷写日志文件到磁盘，提交事务的时候不做任何操作（执行是由mysql的master thread线程来执行的。
# 主线程中每秒会将重做日志缓冲写入磁盘的重做日志文件(REDO LOG)中。不论事务是否已经提交）默认的日志文件是

ib_logfile0,ib_logfile1

# 1:当设为默认值1的时候，每次提交事务的时候，都会将log buffer刷写到日志。

#

2:如果设为2,每次提交事务都会写日志，但并不会执行刷的操作。每秒定时会刷到日志文件。要注意的是，并不能保证100%每秒一定都会刷到磁盘，这要取决于进程的调度。

#

每次事务提交的时候将数据写入事务日志，而这里的写入仅是调用了文件系统的写入操作，而文件系统是有 缓存的，所以这个写入并不能保证数据已经写入到物理磁盘

#

默认值1是为了保证完整的ACID。当然，你可以将这个配置项设为1以外的值来换取更高的性能，但是在系统崩溃的时候，你将会丢失1秒的数据。

#

设为0的话，mysqld进程崩溃的时候，就会丢失最后1秒的事务。设为2,只有在操作系统崩溃或者断电的时候才会丢失最后1秒的数据。InnoDB在做恢复的时候会忽略这个值。

# 总结

#

设为1当然是最安全的，但性能页是最差的(相对其他两个参数而言，但不是不能接受)。如果对数据一致性和完整性要求不高，完全可以设为2，如果只最求性能，例如高并发写的日志服务器，设为0来获得更高性能

innodb_log_buffer_size = 2M

#

此参数确定些日志文件所用的内存大小，以M为单位。缓冲区更大能提高性能，但意外的故障将会丢失数据。MySQL开发人员建议设置为1－8M之间

innodb_log_file_size = 32M

#

此参数确定数据日志文件的大小，更大的设置可以提高性能，但也会增加恢复故障数据库所需的时间

innodb_log_files_in_group = 3

# 为提高性能，MySQL可以以循环方式将日志文件写到多个文件。推荐设置为3

innodb_max_dirty_pages_pct = 90

# innodb主线程刷新缓存池中的数据，使脏数据比例小于90%

innodb_lock_wait_timeout = 120

#

InnoDB事务在被回滚之前可以等待一个锁定的超时秒数。InnoDB在它自己的锁定表中自动检测事务死锁并且回滚事务。

InnoDB用LOCK TABLES语句注意到锁定设置。默认值是50秒

bulk_insert_buffer_size = 8M

# 批量插入缓存大小， 这个参数是针对MyISAM存储引擎来说的。适用于在一次性插入100-1000+条记录时，提高效率。默认值是8M。可以针对数据量的大小，翻倍增加。

myisam_sort_buffer_size = 8M

# MyISAM设置恢复表之时使用的缓冲区的尺寸，当在REPAIR TABLE或用CREATE INDEX创建索引或ALTER TABLE过程中排序 MyISAM索引分配的缓冲区

myisam_max_sort_file_size = 10G

#

如果临时文件会变得超过索引，不要使用快速排序索引方法来创建一个索引。注释:这个参数以字节的形式给出

myisam_repair_threads = 1

# 如果该值大于1，在Repair by sorting过程中并行创建MyISAM表索引(每个索引在自己的线程内)

interactive_timeout = 28800

#

服务器关闭交互式连接前等待活动的秒数。交互式客户端定义为在mysql_real_connect()中使用CLIENT_INTERACTIVE选项的客户端。默认值:28800秒(8小时)

wait_timeout = 28800
#
服务器关闭非交互连接之前等待活动的秒数。在线程启动时，根据全局wait_timeout值或全局interactive_timeout值初始化会话
wait_timeout值，
#
取决于客户端类型(由mysql_real_connect()的连接选项CLIENT_INTERACTIVE定义)。参数默认值：28800秒（8小时）
#
MySQL服务器所支持的最大连接数是有上限的，因为每个连接的建立都会消耗内存，因此我们希望客户端在连接到MySQL Server处理完相应的操作后，
# 应该断开连接并释放占用的内存。如果你的MySQL Server有大量的闲置连接，他们不仅会白白消耗内存，而且如果连接一直在累加而不断开，
# 最终肯定会达到MySQL Server的连接上限数，这会报'too many connections'的错误。对于wait_timeout的值设定，应该根据系统的运行情况来判断。
# 在系统运行一段时间后，可以通过show processlist命令查看当前系统的连接状态，如果发现有大量的sleep状态的连接进程，则说明该参数设置的过大，
# 可以进行适当的调整小些。要同时设置interactive_timeout和wait_timeout才会生效。

[mysqldump]
quick
max_allowed_packet = 16M #服务器发送和接受的最大包长度
[myisamchk]
key_buffer_size = 8M
sort_buffer_size = 8M
read_buffer = 4M
write_buffer = 4M

### 3-2-3 PHP配置文件

[PHP]
; PHP还是一个不断发展的工具，其功能还在不断地删减
; 而php.ini的设置更改可以反映出相当的变化，
; 在使用新的PHP版本前，研究一下php.ini会有好处的

;;;;;;;;;;;;;;;;;;;
; 关于这个文件 ;

;;;;;;;;;;;;;;;;;;;

; 这个文件控制了PHP许多方面的观点。为了让PHP读取这个文件，它必须被命名为
; 'php.ini'。PHP 将在这些地方依次查找该文件：当前工作目录；环境变量PHPRC
; 指明的路径；编译时指定的路径。
; 在windows下，编译时的路径是Windows安装目录。
; 在命令行模式下，php.ini的查找路径可以用 -c 参数替代。

; 该文件的语法非常简单。空白字符和用分号';'开始的行被简单地忽略（就象你可能
; 猜到的一样）。章节标题（例如：[Foo]）也被简单地忽略，即使将来它们可能
; 有某种的意义。
;
; 指示被指定使用如下语法：
; 指示标识符 = 值
; directive = value
; 指示标识符 是 *大小写敏感的* - foo=bar 不同于 FOO = bar。
;
; 值可以是一个字符串，一个数字，一个 PHP 常量 (如：E_ALL or M_PI), INI 常量中的
; 一个 (On, Off, True, False, Yes, No and None)，或是一个表达式
; (如: E_ALL & ~E_NOTICE), 或是用引号括起来的字符串(" foo" ).
;
; INI 文件的表达式被限制于位运算符和括号。
; | bitwise OR
; & bitwise AND
; ~ bitwise NOT
; ! boolean NOT
;
; 布尔标志可用 1, On, True or Yes 这些值置于开的状态。
; 它们可用 0, Off, False or No 这些值置于关的状态。
;
; 一个空字符串可以用在等号后不写任何东西表示，或者用 None 关键字:
;
; foo = ; 将foo置为空字符串
; foo = none ; 将foo置为空字符串
; foo = " none" ; 将foo置为字符串'none'
;
; 如果你值设置中使用常量，而这些常量属于动态调入的扩展库（不是 PHP 的扩展，就是
; Zend 的扩展），你仅可以调入这些扩展的行*之后*使用这些常量。
;

; 所有在 php.ini-dist 文件里设定的值与内建的默认值相同(这是说，如果 php.ini
; 没被使用或者你删掉了这些行，默认值与之相同)。

;;;;;;;;;;;;;;;;;;;;
; 语言选项 ;
;;;;;;;;;;;;;;;;;;;;

engine = On
; 使 PHP scripting language engine(PHP 脚本语言引擎)在 Apache下有效。
short_open_tag = On
; 允许 tags 将被识别。
asp_tags = Off
; 允许ASP-style tags
precision = 14
; 浮点类型数显示时的有效位数

y2k_compliance = Off
; 是否打开 2000年适应 (可能在非Y2K适应的浏览器中导致问题)

output_buffering = Off
; 输出缓存允许你甚至在输出正文内容之后发送 header(标头，包括cookies)行
; 其代价是输出层减慢一点点速度。你可以使用输出缓存在运行时打开输出缓存，
; 或者在这里将指示设为 On 而使得所有文件的输出缓存打开。

output_handler = ; 你可以重定向你的脚本的所有输出到一个函数，
; 那样做可能对处理或以日志记录它有用。
; 例如若你将这个output_handler 设为" ob_gzhandler" ，
; 则输出会被透明地为支持gzip或deflate编码的浏览器压缩。
; 设一个输出处理器自动地打开输出缓冲。

implicit_flush = Off
; 强制flush(刷新)让PHP 告诉输出层在每个输出块之后自动刷新自身数据。
; 这等效于在每个 print() 或 echo() 调用和每个 HTML 块后调用flush()函数。
; 打开这项设置会导致严重的运行时冲突，建议仅在debug过程中打开。

allow_call_time_pass_reference = On
; 是否让强迫函数调用时按引用传递参数。这一方法遭到抗议，

; 并可能在将来版本的PHP/Zend里不再支持。
; 受到鼓励的指定哪些参数按引用传递的方法是在函数声明里。
; 你被鼓励尝试关闭这一选项并确认你的脚本仍能正常工作，以保证在将来版本的语言里
; 它们仍能工作。(你将在每次使用该特点时得到一个警告，而参数将按值而不是按引用
; 传递)。

; Safe Mode 安全模式
safe_mode = Off
safe_mode_exec_dir =
safe_mode_allowed_env_vars = PHP_
; ？Setting certain environment variables
; ？may be a potential security breach.
; 该指示包含用逗号分隔的前缀列表。安全模式中，用户仅可以替换
; 以在此列出的前缀开头的环境变量的值。
; 默认地，用户将仅能 设定以PHP_开头的环境变量，(如: PHP_FOO=BAR)。
; 注意: 如果这一指示为空，PHP 将让用户更改任意环境变量!

safe_mode_protected_env_vars = LD_LIBRARY_PATH
; 这条指示包含一个用逗号分隔的环境变量列表，那是最终用户将不能用putenv () 更改的。
; 这些变量甚至在safe_mode_allowed_env_vars 设置为允许的情况下得到保护。

disable_functions =
; 这条指示让你可以为了安全的原因让特定函数失效。
; 它接受一个用逗号分隔的函数名列表。
; 这条指示 *不受* 安全模式是否打开的影响。

; 语法高亮模式的色彩。
; 只要能被接受的东西就能工作。

highlight.string = #DD0000
highlight.comment = #FF8000
highlight.keyword = #007700
highlight.bg = #FFFFFF
highlight.default = #0000BB
highlight.html = #000000

; Misc 杂项
expose_php = Off

; 决定 PHP 是否标示它装在服务器上的事实(例如:加在它 — PHP— 给Web服务
; 发送的信号上)。
; (我个人的意见, 在出现什么power-by的header的时候, 把这关掉。)
; 它不会有安全上的威胁, 但它使检查你的服务器上是否安装了PHP成为了可能。

;;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;

max_execution_time = 30 ; 每个脚本的最大执行时间, 按秒计
memory_limit = 8388608 ; 一个脚本最大可使用的内存总量 (这里是8MB)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
; 出错控制和登记 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 错误报告是按位的。或者将数字加起来得到想要的错误报告等级。
; E_ALL - 所有的错误和警告
; E_ERROR - 致命性运行时错
; E_WARNING - 运行时警告(非致命性错)
; E_PARSE - 编译时解析错误
; E_NOTICE - 运行时提醒(这些经常是你的代码的bug引起的,
;也可能是有意的行为造成的。(如:基于未初始化的变量自动初始化为一个
;空字符串的事实而使用一个未初始化的变量)

; E_CORE_ERROR - 发生于PHP启动时初始化过程中的致命错误
; E_CORE_WARNING - 发生于PHP启动时初始化过程中的警告(非致命性错)
; E_COMPILE_ERROR - 编译时致命性错
; E_COMPILE_WARNING - 编译时警告(非致命性错)
; E_USER_ERROR - 用户产生的出错消息
; E_USER_WARNING - 用户产生的警告消息
; E_USER_NOTICE - 用户产生的提醒消息
; 例子:
; error_reporting = E_ALL & ~E_NOTICE ; 显示所有的错误, 除了提醒
; error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR ; 仅显示错误

error_reporting = E_ALL & ~E_NOTICE ; 显示所有的错误，除了提醒

display_errors = On ; 显示出错误信息(作为输出的一部分)

; 在最终发布的web站点上，强烈建议你关掉这个特性，并使用

; 错误日志代替(参看下面)。

; 在最终发布的web站点继续让 display_errors 有效可能

; 暴露一些有关安全的信息，例如你的web服务上的文件路径、

; 你的数据库规划或别的信息。

display_startup_errors = Off ; 甚至当display_erroes打开了，发生于PHP的启动的步骤中

; 的错误也不会被显示。

; 强烈建议保持使 display_startup_errors 关闭，

; 除了在改错过程中。

log_errors = Off ;

在日志文件里记录错误(服务器指定的日志，stderr标准错误输出，或error_log(下面的))

; 正如上面说明的那样，强烈建议你在最终发布的web站点以日志记录错误

; 取代直接错误输出。

track_errors = Off ; 保存最近一个 错误/警告 消息于变量 $php_errormsg (boolean)

;error_prepend_string = " " ; 于错误信息前输出的字符串

;error_append_string = " " ; 于错误信息后输出的字符串

;error_log = filename ; 记录错误日志于指定文件

;error_log = syslog ; 记录错误日志于系统日志 syslog (NT 下的事件日志, Windows 95下无效)

warn_plus_overloading = Off ; 当将‘ +’ 用于字符串时警告

;;;;;;;;;;;;;;;;;;

; Data Handling ;

;;;;;;;;;;;;;;;;;;

variables_order = " EGPCS" ; 这条指示描述了PHP 记录

; GET, POST, Cookie, Environment and Built-in 这些变量的顺序。

; (以 G, P, C, E & S 代表，通常以 EGPCS 或 GPC 的方式引用)。

; 按从左到右记录，新值取代旧值。

register_globals = On ; 是否将这些 EGPCS 变量注册为全局变量。

; 若你不想让用户数据不在全局范围内混乱的话，你可能想关闭它。
; 这和 track_vars 连起来用更有意义 — 这样你可以通过
; $HTTP_*_VARS[] 数组访问所有的GPC变量。

register_argc_argv = On ; 这条指示告诉 PHP 是否声明 argv和argc 变量
;（注:这里argv为数组,argc为变量数）
;（其中包含用GET方法传来的数据）。
; 若你不想用这些变量，你应当关掉它以提高性能。

track_vars = On ; 使$HTTP_*_VARS[]数组有效，这里*在使用时用
; ENV, POST, GET, COOKIE or SERVER替换
post_max_size = 8M ; PHP将接受的POST数据最大大小。

gpc_order = " GPC" ; 这条指示被人反对。用 variables_order 代替。

; Magic quotes
magic_quotes_gpc = On ; 在输入的GET/POST/Cookie数据里使用魔术引用
;（原文就这样，呵呵，所谓magic quotes 应该是指用转义符加在引用性的控制字符上，如 '....）
magic_quotes_runtime= Off ; 对运行时产生的数据使用魔术引用，
; 例如:用SQL查询得到的数据，用exec()函数得到的数据，等等
magic_quotes_sybase = Off ; 采用 Sybase形式的魔术引用（用 " 脱出 ' 而不用 '）

; 自动在 PHP 文档之前和之后添加文件
auto_prepend_file =
auto_append_file =

; 象4.04b4一样，PHP 默认地总是在 " Content-type:" 头标输出一个字符的编码方式。
; 让输出字符集失效，只要设置为空。
; PHP 的内建默认值是 text/html
default_mimetype = " text/html"
;default_charset = " iso-8859-1"

;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;

include_path = ; include 路径设置，UNIX: " /path1:/path2" Windows: " \path1;\path2"

doc_root = ; php 页面的根路径，仅在非空时有效

user_dir = ; 告知 php 在使用 /~username 打开脚本时到哪个目录下去找，仅在非空时有效

;upload_tmp_dir = ; 存放用HTTP协议上载的文件的临时目录（在没指定时使用系统默认的）

upload_max_filesize = 2097152 ; 文件上载默认地限制为2 Meg

extension_dir = c:\php\ ; 存放可加载的扩充库（模块）的目录

enable_dl = On ; 是否使dl()有效。

; 在多线程的服务器上 dl()函数*不能*很好地工作，

; 例如IIS or Zeus，并在其上默认为禁止

;;;;;;;;;;;;;;;;

; File Uploads ;

;;;;;;;;;;;;;;;;

file_uploads = On ; 是否允许HTTP方式文件上载

;upload_tmp_dir = ; 用于HTTP上载的文件的临时目录（未指定则使用系统默认）

upload_max_filesize = 2M ; 上载文件的最大许可大小

; Fopen wrappers ;

;;;;;;;;;;;;;;;;;;

allow_url_fopen = On ; 是否允许把URLs当作http:.. 或把文件当作ftp:...

;;;;;;;;;;;;;;;;;;;;;;

; 动态扩展 ;

; Dynamic Extensions ;

;;;;;;;;;;;;;;;;;;;;;;

; 若你希望一个扩展库自动加载，用下面的语法：

; extension=modulename.extension

; 例如，在windows上，

; extension=msql.dll

; or 在UNIX下，

; extension=msql.so

; 注意，这只应当是模块的名字，不需要目录信息放在里面。

; 用上面的 extension_dir 指示指定扩展库的位置。

;Windows 扩展

;extension=php_nsmail.dll
extension=php_calendar.dll
;extension=php_dbase.dll
;extension=php_filepro.dll
extension=php_gd.dll
;extension=php_dbm.dll
;extension=php_mssql.dll
;extension=php_zlib.dll
;extension=php_filepro.dll
;extension=php_imap4r2.dll
;extension=php_ldap.dll
;extension=php_crypt.dll
;extension=php_msql2.dll
;extension=php_odbc.dll
; 注意，MySQL的支持现在是内建的，因此，不需要用它的dll

;;;;;;;;;;;;;;;;;;;;
; 模块设定 ;
; Module Settings ;
;;;;;;;;;;;;;;;;;;;;

[Syslog]
define_syslog_variables = Off ; 是否定义各种的系统日志变量
; 如：$LOG_PID, $LOG_CRON, 等等。
; 关掉它是个提高效率的好主意。
; 运行时，你可以调用函数define_syslog_variables()，来定义这些变量

[mail function]
SMTP = localhost ;仅用于win32系统
sendmail_from = me@localhost.com ;仅用于win32系统
;sendmail_path = ;仅用于unix, 也可支持参数（默认的是'sendmail -t -i'）

[Debugger]
debugger.host = localhost
debugger.port = 7869
debugger.enabled = False

[Logging]

; 这些配置指示用于示例的日志记录机制。

; 看 examples/README.logging 以得到更多的解释

;logging.method = db

;logging.directory = /path/to/log/directory

[Java]

;java.class.path = .\php_java.jar

;java.home = c:\jdk

;java.library = c:\jdk\jre\bin\hotspot\jvm.dll

;java.library.path = .\

[SQL]

sql.safe_mode = Off

[ODBC]

;uodbc.default_db = Not yet implemented

;uodbc.default_user = Not yet implemented

;uodbc.default_pw = Not yet implemented

uodbc.allow_persistent = On ; 允许或禁止 持久连接

uodbc.check_persistent = On ; 在重用前检查连接是否还可用

uodbc.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

uodbc.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

uodbc.defaultlrl = 4096 ; 控制 LONG 类型的字段。返回变量的字节数，0 代表通过（？）0 means passthru

uodbc.defaultbinmode = 1 ; 控制 二进制数据。0 代表?????Handling of binary data. 0 means passthru, 1 return as is, 2 convert to char

; 见有关 odbc_binmode 和 odbc_longreadlen 的文档以得到 uodbc.defaultlrl 和 uodbc.defaultbinmode 的解释。

[MySQL]

mysql.allow_persistent = On ; 允许或禁止 持久连接

mysql.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

mysql.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

mysql.default_port = ; mysql_connect() 使用的默认端口，如不设置，mysql_connect()

; 将使用变量 $MYSQL_TCP_PORT，或在/etc/services 下的mysql-tcp 条目(unix),

; 或在编译是定义的 MYSQL_PORT(按这样的顺序)

; Win32环境，将仅检查MYSQL_PORT。

mysql.default_socket = ; 用于本地 MySql 连接的默认的套接字名。为空，使用 MYSQL 内建值

mysql.default_host = ; mysql_connect() 默认使用的主机（安全模式下无效）

mysql.default_user = ; mysql_connect() 默认使用的用户名（安全模式下无效）

mysql.default_password = ; mysql_connect() 默认使用的密码（安全模式下无效）

; 注意，在这个文件下保存密码通常是一个*坏*主意

; *任何*可以使用PHP访问的用户可以运行

; 'echo cfg_get_var(" mysql.default_password" )'来显示那个密码!

; 而且当然地，任何有读该文件权力的用户也能看到那个密码。

[mSQL]

msql.allow_persistent = On ; 允许或禁止 持久连接

msql.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

msql.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

[PostgresSQL]

pgsql.allow_persistent = On ; 允许或禁止 持久连接

pgsql.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

pgsql.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

[Sybase]

sybase.allow_persistent = On ; 允许或禁止 持久连接

sybase.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

sybase.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

;sybase.interface_file = " /usr/sybase/interfaces"

sybase.min_error_severity = 10 ; 显示的错误的最低严重性

sybase.min_message_severity = 10 ; 显示的消息的最低重要性

sybase.compatability_mode = Off ; 与旧版的PHP 3.0 兼容的模式。若打开，这将导致 PHP 自动地

; 把根据结果的 Sybase 类型赋予它们，

; 而不是把它们全当成字符串。

; 这个兼容模式不会永远留着，

; 因此，将你的代码进行需要的修改，

; 并将该项关闭。

[Sybase-CT]

sybct.allow_persistent = On ; 允许或禁止 持久连接

sybct.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

sybct.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

sybct.min_server_severity = 10 ; 显示的错误的最低严重性

sybct.min_client_severity = 10 ; 显示的消息的最低重要性

[bcmath]

bcmath.scale = 0 ; 用于所有bcmath函数的10十进制数数字的个数number of decimal digits for all bcmath functions

[browscap]

;browscap = extra/browscap.ini

browscap = C:\WIN\SYSTEM\inetsrv\browscap.ini

[Informix]

ifx.default_host = ; ifx_connect() 默认使用的主机（安全模式下无效）

ifx.default_user = ; ifx_connect() 默认使用的用户名（安全模式下无效）

ifx.default_password = ; ifx_connect() 默认使用的密码（安全模式下无效）

ifx.allow_persistent = On ; 允许或禁止 持久连接

ifx.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制

ifx.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制

ifx.textasvarchar = 0 ; 若打开，select 状态符返回一个 ‘ text blob’ 字段的内容，而不是它的id

ifx.byteasvarchar = 0 ; 若打开，select 状态符返回一个 ‘ byte blob’ 字段的内容，而不是它的id

ifx.charasvarchar = 0 ; 追踪从固定长度的字符列里剥离的空格。

; 可能对 Informix SE 用户有效。

ifx.blobinfile = 0 ; 若打开，text和byte blobs 的内容被导出到一个文件

; 而不是保存到内存。

ifx.nullformat = 0 ; NULL（空）被作为空字段返回，除非，这里被设为1。

; 这种情况下（为1），NULL作为字串NULL返回。

[Session]

session.save_handler = files ; 用于保存/取回数据的控制方式

session.save_path = C:\win\temp ; 在 save_handler 设为文件时传给控制器的参数，

; 这是数据文件将保存的路径。

session.use_cookies = 1 ; 是否使用cookies

session.name = PHPSESSID

; 用在cookie里的session的名字

session.auto_start = 0 ; 在请求启动时初始化session

session.cookie_lifetime = 0 ; 为按秒记的cookie的保存时间，

; 或为0时，直到浏览器被重启

session.cookie_path = / ; cookie的有效路径

session.cookie_domain = ; cookie的有效域

session.serialize_handler = php ; 用于连接数据的控制器

; php是 PHP 的标准控制器。

session.gc_probability = 1 ; 按百分比的'garbage collection（碎片整理）'进程
; 在每次 session 初始化的时候开始的可能性。
session.gc_maxlifetime = 1440 ; 在这里数字所指的秒数后，保存的数据将被视为
; '碎片(garbage)'并由gc 进程清理掉。
session.referer_check = ; 检查 HTTP引用以使额外包含于URLs中的ids无效
session.entropy_length = 0 ; 从文件中读取多少字节
session.entropy_file = ; 指定这里建立 session id
; session.entropy_length = 16
; session.entropy_file = /dev/urandom
session.cache_limiter = nocache ; 设为{nocache,private,public},以决定 HTTP 的
; 缓存问题
session.cache_expire = 180 ; 文档在 n 分钟后过时
session.use_trans_sid = 1 ; 使用过渡性的 sid 支持，若编译时许可了
; --enable-trans-sid
url_rewriter.tags = " a=href,area=href,frame=src,input=src,form=fakeentry"

[MSSQL]
;extension=php_mssql.dll
mssql.allow_persistent = On ; 允许或禁止 持久连接
mssql.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制
mssql.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制
mssql.min_error_severity = 10 ; 显示的错误的最低严重性
mssql.min_message_severity = 10 ; 显示的消息的最低重要性
mssql.compatability_mode = Off ; 与旧版的PHP 3.0 兼容的模式。

[Assertion]
; ？？？？？
;assert.active = On ; ？assert(expr); active by default
;assert.warning = On ; issue a PHP warning for each failed assertion.
;assert.bail = Off ; don't bail out by default.
;assert.callback = 0 ; user-function to be called if an assertion fails.
;assert.quiet_eval = 0 ; eval the expression with current error_reporting(). set to true if you want error_reporting(0) around the eval().

[Ingres II]
ii.allow_persistent = On ; 允许或禁止 持久连接
ii.max_persistent = -1 ; 持久连接的最大数。-1 代表无限制
ii.max_links = -1 ; 连接的最大数目（持久和非持久）。-1 代表无限制
ii.default_database = ; 默认 database (format : [node_id::]dbname[/srv_class]

ii.default_user = ; 默认 user

ii.default_password = ; 默认 password

[Verisign Payflow Pro]

pfpro.defaulthost = " test.signio.com" ; 默认的 Signio 服务器

pfpro.defaultport = 443 ; 连接的默认端口

pfpro.defaulttimeout = 30 ; 按秒计的默认超时时间

; pfpro.proxyaddress = ; 默认的代理的 IP 地址（如果需要）

; pfpro.proxyport = ; 默认的代理的端口

; pfpro.proxylogon = ; 默认的代理的登录（logon 用户名）

; pfpro.proxypassword = ; 默认的代理的密码

[Sockets]

sockets.use_system_read = On ; 使用系统的read() 函数替代 php_read()封装

; Local Variables:（局部变量）

; tab-width: 4

; End

## 3-3 源码安装-PHP扩展编译安装

以安装swoole扩展为例:

□□步骤1: wget  pecl.php.net/get/swoole-1.7.21.tgz  (下载swoole打包文件)

□□步骤2: tar zxvf swoole-1.7.21.tgz  (解压swoole压缩文件)

□□步骤3: cd  swoole-1.7.21  (进入swoole目录)

□□步骤4:  /usr/local/php/bin/phpize  (或直接使用phpize 。phpize 命令是用来准备 PHP 扩展库的编译环境的，每个人的phpize命令的路径可能不同，不要直接复制执行)

□□步骤5:  ./configure  --with-php-config=/usr/local/php/bin/php-config  (或直接使用./configure 。./configure 作用是对即将安装的软件进行配置，检查当前的环境是否满足要安装软件的依赖关系)

□□步骤6: make (编译)

□□步骤7: make test (这一步就是对上一步 make 的检查了，要确保 make 是没有错误的)

□□步骤8: make install

□□执行完make install会看到类似这样的信息:

Installing shared extensions: /usr/local/php/lib/php/extensions/no-debug-non-zts-20100525/，这条路径就是PHP扩展(extension_dir)的路径，安装成功的swoole.so就在该目录里。

□□步骤9: vim /usr/local/php/lib/php.ini 添加extension=swoole.so

□□步骤10: service php-fpm restart (重启PHP的FastCGI进程管理器,现在我已经把nginx，php-fpm的服务启动脚本已经写进/etc/rc.d/init.d)

### Apach下多版本php安装切换

经常用于本地开发环境中，某些扩展对PHP版本有要求，此时需要在本地环境中安装多版本PHP，然后在apache中配置，对接哪一个版本。

但由于配置比较复杂，首先需要针对每一个版本的php
扩展都需要重新编译，容易出错。不建议新手操作

有兴趣的同学可以百度下！！！

## 4.LNMP安装/配置

### 4-1 yum 安装 nginx

安装Nginx

# 查看相关信息

yum info nginx

yum info httpd

# 移除 httpd,也就是 Apache
yum remove httpd -y

# 安装 nginx
yum install nginx -y

#设置 nginx 自启动
chkconfig nginx on

# 查看服务自启动情况
chkconfig

# 启动nginx服务
service nginx start

# 查看端口监听状态
netstat -ntl

# 此时你可以访问试试了
# 例如: http://192.168.1.111:8080

# 如果访问不了,请 ping 一下试试
# 或者查看 iptables 防火墙状态
service iptables status

# 关闭防火墙,简单粗暴的
service iptables stop
如果你没有权限执行这些操作，你可能需要使用 sudo 权限

配置Nginx反向代理

/etc/nginx/nginx.conf
user nginx;
worker_processes 1;

error_log /var/log/nginx/error.log;

```
pid        /var/run/nginx.pid;

events {
    use epoll;
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    gzip  on;

    # Load config files from the /etc/nginx/conf.d directory
    # The default server is in conf.d/default.conf
    include /etc/nginx/conf.d/*.conf;

    include     upstream.conf;
    include     cncounter.com.conf;

}
```

做负载的配置: /etc/nginx/upstream.conf

```
upstream www.cncounter.com {
    server 127.0.0.1:8080;
}
```

站点配置文件: /etc/nginx/cncounter.com.conf

```
server
```

```
{
    listen       80;
    server_name    www.cncounter.com;
    index index.jsp;
    root    /usr/local/cncounter_webapp/cncounter/;
    location ~ ^/NginxStatus/ {
    stub_status on;
    access_log off;
    }

    location / {
        root    /usr/local/cncounter_webapp/cncounter/;
        proxy_redirect off ;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header REMOTE-HOST $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size 50m;
        client_body_buffer_size 256k;
        proxy_connect_timeout 30;
        proxy_send_timeout 30;
        proxy_read_timeout 60;
        proxy_buffer_size 256k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;
        proxy_temp_file_write_size 256k;
        proxy_next_upstream error timeout invalid_header http_500 http_503 http_404;
        proxy_max_temp_file_size 128m;
        proxy_pass    http://www.cncounter.com
    }
}
```

重启服务

```
service nginx stop

service nginx start
```

## 4-2 nginx配置文件

```
user nginx nginx ;

#Nginx用户及组:用户 组。window下不指定
```

worker_processes 8;

#工作进程:数目。根据硬件调整，通常等于CPU数量或者2倍于CPU。

error_log  logs/error.log;

error_log  logs/error.log  notice;

error_log  logs/error.log  info;

#错误日志:存放路径。

pid logs/nginx.pid;

#pid(进程标识符):存放路径。

worker_rlimit_nofile 204800;

#指定进程可以打开的最大描述符:数目。
#这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数(ulimit -n)与nginx
#进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。

#现在在Linux 2.6内核下开启文件打开数为65535，worker_rlimit_nofile就相应应该填写65535。

#这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。

events

{
use epoll;

#使用epoll的I/O 模型。linux建议epoll，FreeBSD建议采用kqueue，window下不指定。
#补充说明:
#与apache相类，nginx针对不同的操作系统，有不同的事件模型
#A）标准事件模型
#Select、poll属于标准事件模型，如果当前系统不存在更有效的方法，nginx会选择select或poll
#B）高效事件模型
#Kqueue：使用于FreeBSD 4.1+，OpenBSD 2.9+，NetBSD 2.0 和 MacOS X.使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。
#Epoll：使用于Linux内核2.6版本及以后的系统。
#/dev/poll：使用于Solaris 7 11/99+，HP/UX 11.22+ (eventport)，IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。
#Eventport：使用于Solaris 10。为了防止出现内核崩溃的问题，有必要安装安全补丁。

worker_connections 204800;

#没个工作进程的最大连接数量。根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为。worker_processes*worker_connections

keepalive_timeout 60; #keepalive超时时间。

client_header_buffer_size 4k;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。

#分页大小可以用命令getconf PAGESIZE 取得。
#[root@web001 ~]# getconf PAGESIZE
#4096
#但也有client_header_buffer_size超过4k的情况，但是client_header_buffer_size该值必须设置为“系统分页大小”的整倍数。

open_file_cache max=65535 inactive=60s;
#这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。

```
open_file_cache_valid 80s;
#这个是指多长时间检查一次缓存的有效信息。

open_file_cache_min_uses 1;

#open_file_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。

}


##设定http服务器，利用它的反向代理功能提供负载均衡支持
http
{
include mime.types;  #设定mime类型,类型由mime.type文件定义
default_type application/octet-stream;
log_format main '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" "$http_x_forwarded_for"';
log_format log404 '$status [$time_local] $remote_addr $host$request_uri $sent_http_location';

#日志格式设置。
$remote_addr与$http_x_forwarded_for用以记录客户端的ip地址；
$remote_user：用来记录客户端用户名称；
$time_local： 用来记录访问时间与时区；
$request： 用来记录请求的url与http协议；
$status： 用来记录请求状态；成功是200，
$body_bytes_sent ：记录发送给客户端文件主体内容大小；
$http_referer：用来记录从那个页面链接访问过来的；
$http_user_agent：记录客户浏览器的相关信息；

#通常web服务器放在反向代理的后面，这样就不能获取到客户的IP地址了，通过$remote_add拿到的IP地址是反向代理服务器的iP地址。反向代#理服务器在转发请求的http头信息中，可以增加x_forwarded_for信息，用以记录原有客户端的IP地址和原来客户端的请求的服务器地址。
```

access_log  logs/host.access.log  main;
access_log  logs/host.access.404.log  log404;

#用了log_format指令设置了日志格式之后，需要用access_log指令指定日志文件的存放路径；

server_names_hash_bucket_size 128;

#保存服务器名字的hash表是由指令server_names_hash_max_size 和server_names_hash_bucket_size所控制的。参数hash bucket size总是等于#hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash #bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次#是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.

client_header_buffer_size 4k;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令get conf PAGESIZE取得。

large_client_header_buffers 8 128k;

#客户请求头缓冲大小。nginx默认会用client_header_buffer_size这个buffer来读取header值，如果header过大，它会使large_client_header_buffers来读取。

open_file_cache max=102400 inactive=20s;

#这个指令指定缓存是否启用。
#例: open_file_cache max=1000 inactive=20s;

open_file_cache_valid 30s;

open_file_cache_min_uses 2;

open_file_cache_errors on;

open_file_cache_errors
#语法:open_file_cache_errors on | off 默认值:open_file_cache_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.

open_file_cache_min_uses

#语法:open_file_cache_min_uses number 默认值:open_file_cache_min_uses 1 使用字段:http, server, location
这个指令指定了在open_file_cache指令无效的参数中一定的时间范围内可以使用的最小文件数,如果使用更大的值,文件描述符在cache中总是打开状态.

open_file_cache_valid

#语法:open_file_cache_valid time 默认值:open_file_cache_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open_file_cache中缓存项目的有效信息.

client_max_body_size 300m;

#设定通过nginx上传文件的大小

sendfile on;

#sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络IO处理速度，降低系统uptime。

tcp_nopush on;

#此选项允许或禁止使用socke的TCP_CORK的选项，此选项仅在使用sendfile的时候使用

proxy_connect_timeout 90;
#后端服务器连接的超时时间_发起握手等候响应超时时间

proxy_read_timeout 180;
#连接成功后_等候后端服务器响应时间_其实已经进入后端的排队之中等候处理(也可以说是后端服务器处理请求的时间)

proxy_send_timeout 180;
#后端服务器数据回传时间_就是在规定时间之内后端服务器必须传完所有的数据

proxy_buffer_size 256k;

#设置从被代理服务器读取的第一部分应答的缓冲区大小，通常情况下这部分应答中包含一个小的应答头，默认情况下这个值的大小为指令proxy_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小

proxy_buffers 4 256k;
#设置用于读取应答(来自被代理服务器)的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k

proxy_busy_buffers_size 256k;
proxy_temp_file_write_size 256k;

#设置在写入proxy_temp_path时数据的大小，预防一个工作进程在传递文件时阻塞太长

proxy_temp_path /data0/proxy_temp_dir;
#proxy_temp_path和proxy_cache_path指定的路径必须在同一分区

proxy_cache_path /data0/proxy_cache_dir levels=1:2 keys_zone=cache_one:200m inactive=1d max_size=30g;
#设置内存缓存空间大小为200MB，1天没有被访问的内容自动清除，硬盘缓存空间大小为30GB。

keepalive_timeout 120;
#keepalive超时时间。

tcp_nodelay on;
client_body_buffer_size 512k;
#如果把它设置为比较大的数值，例如256k，那么，无论使用firefox还是IE浏览器，来提交任意小于256k的图片，都很正常。如果注释该指令，使用默认的client_body_buffer_size设置，也就是操作系统页面大小的两倍，8k或者16k，问题就出现了。
无论使用firefox4.0还是IE8.0，提交一个比较大，200k左右的图片，都返回500 Internal Server Error错误

proxy_intercept_errors on;
#表示使nginx阻止HTTP应答代码为400或者更高的应答。

upstream bakend {
server 127.0.0.1:8027;
server 127.0.0.1:8028;
server 127.0.0.1:8029;
hash $request_uri;
}

#nginx的upstream目前支持4种方式的分配

1、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

2、weight
指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。
例如：
upstream bakend {
server 192.168.0.14 weight=10;
server 192.168.0.15 weight=10;
}

2、ip_hash
每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。
例如：
upstream bakend {
ip_hash;
server 192.168.0.14:88;
server 192.168.0.15:80;

}

3、fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

```
upstream backend {
server server1;
server server2;
fair;
}
```

4、url_hash（第三方）

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash_method是使用的hash算法

```
upstream backend {
server squid1:3128;
server squid2:3128;
hash $request_uri;
hash_method crc32;
}
```

tips:

```
upstream bakend{#定义负载均衡设备的Ip及设备状态}{
ip_hash;
server 127.0.0.1:9090 down;
server 127.0.0.1:8080 weight=2;
server 127.0.0.1:6060;
server 127.0.0.1:7070 backup;
}
```

在需要使用负载均衡的server中增加

```
proxy_pass http://bakend/;
```

每个设备的状态设置为:

1.down表示单前的server暂时不参与负载

2.weight为weight越大，负载的权重就越大。

3.max_fails：允许请求失败的次数默认为1.当超过最大次数时，返回proxy_next_upstream模块定义的错误

4.fail_timeout:max_fails次失败后，暂停的时间。

5.backup： 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

nginx支持同时设置多组的负载均衡，用来给不用的server来使用。

client_body_in_file_only设置为On 可以讲client post过来的数据记录到文件中用来做debug

client_body_temp_path设置记录文件的目录 可以设置最多3层目录

location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡

##配置虚拟机

server

{

listen 80;#配置监听端口

server_name image.***.com;#配置访问域名

location ~* \.(mp3|exe)$ {

#对以“mp3或exe”结尾的地址进行负载均衡

proxy_pass http://img_relay$request_uri;#设置被代理服务器的端口或套接字，以及URL

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

#以上三行，目的是将代理服务器收到的用户的信息传到真实服务器上

}

location /face {

```
    if ($http_user_agent ~* "xnp") {
        rewrite ^(.*)$ http://211.151.188.190:8080/face.jpg redirect
    }
proxy_pass http://img_relay$request_uri;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
error_page 404 502 = @fetch;
}

location @fetch {
    access_log /data/logs/face.log log404;
    rewrite ^(.*)$ http://211.151.188.190:8080/face.jpg redirect
}

location /image {
    if ($http_user_agent ~* "xnp") {
        rewrite ^(.*)$ http://211.151.188.190:8080/face.jpg redirect
    }
    proxy_pass http://img_relay$request_uri;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    error_page 404 502 = @fetch;
}

location @fetch {
    access_log /data/logs/image.log log404;
    rewrite ^(.*)$ http://211.151.188.190:8080/face.jpg redirect
    }
}
```

##其他举例

```
server

{

listen 80;
```

```
server_name *.***.com *.***.cn;

location ~* \.(mp3|exe)$ {

proxy_pass http://img_relay$request_uri;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

}

location / {

if ($http_user_agent ~* "xnp") {

rewrite ^(.*)$ http://i1.***img.com/help/noimg.gif redirect;

}

proxy_pass http://img_relay$request_uri;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

#error_page 404 http://i1.***img.com/help/noimg.gif;

error_page 404 502 = @fetch;

}

location @fetch {
```

```
access_log /data/logs/baijiaqi.log log404;

rewrite ^(.*)$ http://i1.***img.com/help/noimg.gif redirect;

}

}
server

{

listen 80;

server_name *.***img.com;



location ~* \.(mp3|exe)$ {

proxy_pass http://img_relay$request_uri;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

}

location / {

if ($http_user_agent ~* "xnp") {

rewrite ^(.*)$ http://i1.***img.com/help/noimg.gif;

}

proxy_pass http://img_relay$request_uri;
```

```
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
#error_page 404 http://i1.***img.com/help/noimg.gif;
error_page 404 = @fetch;
}
#access_log off;
location @fetch {
access_log /data/logs/baijiaqi.log log404;
rewrite ^(.*)$ http://i1.***img.com/help/noimg.gif redirect;
}
}
server
{
listen 8080;
server_name ngx-ha.***img.com;
location / {
stub_status on;
access_log off;
```

```
}

}

server {

listen 80;

server_name imgsrc1.***.net;

root html;

}


server {

listen 80;

server_name ***.com w.***.com;

# access_log /usr/local/nginx/logs/access_log main;

location / {

rewrite ^(.*)$ http://www.***.com/ ;

}

}

server {

listen 80;

server_name *******.com w.*******.com;
```

```
# access_log /usr/local/nginx/logs/access_log main;

location / {

rewrite ^(.*)$ http://www.*******.com/;

}

}

server {

listen 80;

server_name ******.com;

# access_log /usr/local/nginx/logs/access_log main;

location / {

rewrite ^(.*)$ http://www.******.com/;

}

}

location /NginxStatus {
stub_status on;
access_log on;
auth_basic "NginxStatus";
auth_basic_user_file conf/htpasswd;
}

#设定查看Nginx状态的地址
```

```
location ~ /\.ht {
deny all;
}

#禁止访问.htxxx文件

}
```

注释:变量

Ngx_http_core_module模块支持内置变量，他们的名字和apache的内置变量是一致的。

首先是说明客户请求title中的行，例如$http_user_agent,$http_cookie等等。

此外还有其它的一些变量

$args此变量与请求行中的参数相等

$content_length等于请求行的“Content_Length”的值。

$content_type等同与请求头部的”Content_Type”的值

$document_root等同于当前请求的root指令指定的值

$document_uri与$uri一样

$host与请求头部中“Host”行指定的值或是request到达的server的名字（没有Host行）一样

$limit_rate允许限制的连接速率

$request_method等同于request的method，通常是“GET”或“POST”

$remote_addr客户端ip

$remote_port客户端port

$remote_user等同于用户名，由ngx_http_auth_basic_module认证

$request_filename当前请求的文件的路径名，由root或alias和URI request组合而成

$request_body_file

$request_uri含有参数的完整的初始URI

$query_string与$args一样

$sheeme http模式（http,https）尽在要求是评估例如

Rewrite ^(.+)$ $sheme://example.com$; Redirect;

$server_protocol等同于request的协议，使用“HTTP/或“HTTP/

$server_addr
request到达的server的ip，一般获得此变量的值的目的是进行系统调用。为了避免系统调用，有必要在listen指令中指明ip，并使用bind参数。

$server_name请求到达的服务器名

$server_port请求到达的服务器的端口号

$uri等同于当前request中的URI，可不同于初始值，例如内部重定向时或使用index

四、小节

## 2. 课堂练习

## 3. 课后练习

## 4. 资料扩展