

12/30 Edit by Sky

本文档内容来自互联网和平日工作使用经验，本实例的平台都是 redhat5

文档名字为 Ashley 工作-----Nginx+Resin+Memcached 架构篇 (1.0) (更新中)

本文档先各自介绍 Nginx Resin Memcached 然后再叙述如何三者协同工作

其余文档还有

Ashley 工作-----Sun 服务器 (基于 Solaris 10) 日常应用基本处理实例篇 (1.0) (更新中)

Ashley 工作-----MogileFS 安装使用 (架图片服务器) (1.0) (更新中)

Ashley 工作-----基于 Solaris 10 Mysql 日常应用篇 (1.0)

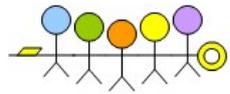
Ashley 工作-----基于 Solaris 10 Oracle 10g 日常应用篇 (1.0)



Nginx 简介.....	4
Nginx 安装、调试.....	6
Nginx 安装.....	6
1. 安装前注意事项.....	6
2. 安装前资源的准备.....	6
3. 安装 Nginx.....	6
4. 验证安装.....	7
5. Nginx 的常用参数和控制.....	7
6. 编写管理 Nginx 的脚步.....	8
7. 使用注意事项.....	11
Nginx 配置文件详解.....	11
Nginx 中的全局变量.....	14
通过例子学习.....	14
Nginx 监控.....	18
Nginx Rewrite 基础.....	19
Rewrite 基本标记.....	19
Rewrite 正则表达式匹配.....	19
Rewrite 条件判断.....	19
Rewrite 应用实例.....	19
伪地址.....	19
域名跳转.....	20
Resin 简介.....	20
Resin 安装.....	21
Resin 安装前提.....	21
Resin 安装.....	21
Resin 配置文件.....	22
单服务器篇.....	22
Resin 中 JVM 的分析.....	28
JVM 虚拟内存.....	28
JVM—垃圾收集.....	29
JVM 垃圾收集算法.....	29
JDK 的四种垃圾收集器.....	29
监控 JVM 的工具和使用.....	31
日常遇到的 OOM 和频繁发生 GC 情况分析.....	34
Resin 中除 JVM 外的优化.....	35
Memcached 介绍.....	36
What is Memcached?.....	36
Memcached 运行图:	36
Memcached 特点.....	37
MemCached 安装.....	38
安装前提.....	38
安装前资源的准备.....	38
安装步骤.....	38



安装中可能出现的问题.....	39
正在更新中.....	39
附录.....	39
Nginx 编译模块名称解析.....	39



Nginx 简介

Nginx 是俄罗斯人编写的十分轻量级的 HTTP 服务器, Nginx, 它的发音为 “engine X”, 是一个高性能的 HTTP 和反向代理服务器, 同时也是一个 IMAP/POP3/SMTP 代理服务器. Nginx 是由俄罗斯人 Igor Sysoev 为俄罗斯访问量第二的 Rambler.ru 站点开发的, 它已经在该站点运行超过两年半了. Igor Sysoev 在建立的项目时, 使用基于 BSD 许可。

据说他当初是 F5 的成员之一, 英文主页: <http://nginx.net> 。

俄罗斯的一些大网站已经使用它超过两年多了, 一直表现不凡, 相信想了解 nginx 的朋友都读过阿叶大哥的利用 nginx 实现负载均衡. 直到 2007 年 4 月, 俄罗斯大约有 20%左右的虚拟主机是由 nginx 服务或代理的. Google 在线安全博客中统计 nginx 服务或代理了大约所有 Internet 虚拟主机的 4%。而 netcraft 的统计显示, nginx 服务的主机在过去的一年里以四倍的速度增长短短的几年里, 它的排名已跃进第 9。

(参见: <http://survey.netcraft.com/Reports/200707/>)

Nginx 以事件驱动的方式编写, 所以有非常好的性能, 同时也是一个非常高效的反向代理、负载平衡。其拥有匹配 Lighttpd 的性能, 同时还没有 Lighttpd 的内存泄漏问题, 而且 Lighttpd 的 mod_proxy 也有一些问题并且很久没有更新。

Nginx 并不支持 cgi 方式运行, 原因是可以减少因此带来的一些程序上的漏洞。那么我们必须使用 FastCGI 方式来执行 PHP 程序。

现在, Igor 将源代码以类 BSD 许可证的形式发布. Nginx 因为它的稳定性、丰富的模块库、灵活的配置和低系统资源的消耗而闻名. 业界一致认为它是 Apache2.2+mod_proxy_balancer 的轻量级代替者, 不仅是因为响应静态页面的速度非常快, 而且它的模块数量达到 Apache 的近 2/3。对 proxy 和 rewrite 模块的支持很彻底, 还支持 mod_fcgi、ssl、vhosts, 适合用来做 mongrel clusters 的前端 HTTP 响应。

Nginx 做为 HTTP 服务器, 有以下几项基本特性:

1. 处理静态文件, 索引文件以及自动索引; 打开文件描述符缓冲。
2. 无缓存的反向代理加速, 简单的负载均衡和容错。
3. FastCGI, 简单的负载均衡和容错。
4. 模块化的结构。包括 gzipping, byte ranges, chunked responses 以及 SSI-filter 等 filter。如果由 FastCGI 或其它代理服务器处理单页中存在的多个 SSI, 则这项处理可以并行运行, 而不需要相互等待。
5. 支持 SSL 和 TLSSNI.。



Nginx 专为性能优化而开发,性能是其最重要的考量,实现上非常注重效率。它支持内核 Poll 模型,能经受高负载的考验,有报告表明能支持高达 50,000 个并发连接数。

Nginx 具有很高的稳定性。其它 HTTP 服务器,当遇到访问的峰值,或者有人恶意发起慢速连接时,也很可能会导致服务器物理内存耗尽频繁交换,失去响应,只能重启服务器。例如当前 apache 一旦上到 200 个以上进程,web 响应速度就明显非常缓慢了。而 Nginx 采取了分阶段资源分配技术,使得它的 CPU 与内存占用率非常低。nginx 官方表示保持 10,000 个没有活动的连接,它只占 2.5M 内存,所以类似 DOS 这样的攻击对 nginx 来说基本上是毫无用处的。就稳定性而言,nginx 比 lighthttpd 更胜一筹。

Nginx 支持热部署。它的启动特别容易,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。你还能够在不间断服务的情况下,对软件版本进行进行升级。

Nginx 采用 master-slave 模型,能够充分利用 SMP 的优势,且能够减少工作进程在磁盘 I/O 的阻塞延迟。当采用 select()/poll() 调用时,还可以限制每个进程的连接数。

Nginx 代码质量非常高,代码很规范,手法成熟,模块扩展也很容易。特别值得一提的是强大的 Upstream 与 Filter 链。Upstream 为诸如 reverse proxy,与其他服务器通信模块的编写奠定了很好的基础。而 Filter 链最酷的部分就是各个 filter 不必等待前一个 filter 执行完毕。它可以把前一个 filter 的输出做为当前 filter 的输入,这有点像 Unix 的管线。这意味着,一个模块可以开始压缩从后端服务器发送过来的请求,且可以在模块接收完后端服务器的整个请求之前把压缩流转向客户端。

Nginx 采用了一些 OS 提供的最新特性如对 sendfile (Linux 2.2+), accept-filter (FreeBSD 4.1+), TCP_DEFER_ACCEPT (Linux 2.4+) 的支持,从而大大提高了性能。

当然,nginx 还很年轻,多多少少存在一些问题,比如: Nginx 是俄罗斯人创建,目前文档方面还不是很完善。因为文档大多是俄语,所以文档方面这也是个障碍。尽管 nginx 的模块比较多,但它们还不够完善。对脚本的支持力度不够。

这些问题,nginx 的作者和社区都在努力解决,我们有理由相信 nginx 将继续以高速的增长率来分享轻量级 HTTP 服务器市场,会有一个更美好的未来。



Nginx 安装、调试

Nginx 安装

1. 安装前注意事项

- 1) 目前官方 Nginx 并不支持 Windows，您只能在包括 Linux、UNIX、BSD 系统下安装和使用；
- 2) Nginx 本身只是一个 HTTP 和反向代理服务器，它无法像 Apache 一样通过安装各种模块来支持不同的页面脚本，例如 PHP、CGI 等。
 - 3) 为了确保能在 Nginx 中使用正则表达式进行更灵活的配置，安装之前需要确定系统是否安装有 PCRE (Perl Compatible Regular Expressions) 包。
 - 4) 需要安装在 2.6 以上内核版本的 GNU/Linux 系统中。
 - 5) 编译前，请先确认 gcc、make、patch 等编译工具是否已安装，并可正常使用。

2. 安装前资源的准备

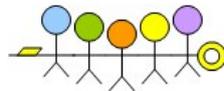
主要软件清单列表：

- 1) Nginx 安装文件
Nginx 0.6.32 nginx-0.6.32.tar.gz
(Nginx 官方站点为 <http://www.nginx.net>，国内较好的中文站点为 <http://www.nginx.cn>)
- 2) Nginx URL 哈希模块
Upstream Hash Module nginx_upstream_hash-0.3.tar.gz
- 3) 正则表达式模块
PCRE 7.7 pcre-7.7.tar.gz
(到 <http://www.pcre.org> 下载 PCRE 库的最新稳定版本)

3. 安装 Nginx

(本安装环境是在 Rethad5 上，所有的安装文件和其他软件都在/usr/local/src 上)

```
#cd /usr/local/src
#gunzip nginx-0.6.32.tar.gz
#gunzip nginx_upstream_hash-0.3.tar.gz
#gunzip pcre-7.7.tar.gz
#tar -xvf nginx-0.6.32.tar
#tar -xvf nginx_upstream_hash-0.3.tar
#tar -xvf pcre-7.7.tar
#cd nginx-0.6.32 (随后 patch 的运行必须要在该目录下)
#patch -p0 < /usr/local/src/nginx_upstream_hash-0.3/nginx.patch
```



```
#!/configure --add-module=/usr/local/src/nginx_upstream_hash-0.3/  
--with-pcre=/usr/local/src/pcre-7.7/ --prefix=/usr/local/nginx  
--with-http_stub_status_module --without-select_module  
--without-poll_module  
(模块的选择请参考附录)  
#make
```

4.验证安装

安装成功后 /usr/local/nginx 目录下有四个子目录分别是：conf、html、logs、sbin。其中 Nginx 的配置文件存放于 conf/nginx.conf；Nginx 只有一个程序文件位于 sbin 目录下的 nginx 文件。如果你使用默认的配置，则要确保系统的 80 端口没被其他程序占用，然后运行 /usr/local/nginx/sbin/nginx 命令来启动 Nginx。

可以通过以下两种方法测试 Nginx 是否成功启动

1. 打开浏览器访问此机器的 http://IP:端口 (80 可免输入)，如果浏览器出现 Welcome to nginx! 则表示 Nginx 已经安装并运行成功。
2. 输入命令 netstat -nlp 如果能找到 Nginx 的端口那也代表你的 Nginx 成功启动了。我的输出结果是：

```
tcp 0 0 0.0.0.0:82          0.0.0.0:* LISTEN      10209/nginx
```

5.Nginx 的常用参数和控制

Nginx 安装后只有一个程序文件，本身并不提供各种管理程序，它是使用参数和系统信号机制对 Nginx 进程本身进行控制的。

程序运行参数

- c <path_to_config>: 使用指定的配置文件而不是 conf 目录下的 nginx.conf。
 - t: 测试配置文件是否正确，在运行时需要重新加载配置的时候，此命令非常重要，用来检测所修改的配置文件是否有语法错误。
 - v: 显示 nginx 版本号。
 - V: 显示 nginx 的版本号以及编译环境信息以及编译时的参数。
- 如果要测试某个配置文件是否书写正确，可以使用以下命令
- ```
/usr/local/nginx/sbin/nginx -t -c conf/nginx_my.conf
```

### 通过信号量对 Nginx 进行控制

有一个奇怪的现象不知道大家有没有注意到，就是你会发现只有启动 Nginx

的命令，但并没有关闭 Nginx 的命令。原因是 Nginx 是通过信号量来控制 Nginx 的，包括重启 Nginx、停止 Nginx 等。

Nginx 支持下表中的信号：

| 信号名       | 作用描述                              |
|-----------|-----------------------------------|
| TERM, INT | 快速关闭程序，中止当前正在处理的请求                |
| QUIT      | 处理完当前请求后，关闭程序                     |
| HUP       | 重新加载配置，并开启新的工作进程，关闭就的进程，此操作不会中断请求 |
| USR1      | 重新打开日志文件，用于切换日志，例如每天生成一个新的日志文件    |
| USR2      | 平滑升级可执行程序                         |
| WINCH     | 从容关闭工作进程                          |

有两种方式来通过这些信号去控制 Nginx：

- 1) 通过 logs 目录下的 nginx.pid 查看当前运行的 Nginx 的进程 ID，通过 `kill - XXX <pid>` 来控制 Nginx，其中 XXX 可通过 `cat $Nginx_HOME/logs/nginx.pid` 来查看。
- 2) 如果您的系统中只有一个 Nginx 进程，那您也可以通过 `killall` 命令来完成，例如运行 `killall - s HUP nginx` 来让 Nginx 重新加载配置。

## 6. 编写管理 Nginx 的脚步

从上面的叙述可知道，Nginx 是通过信号量来控制的，所以管理起来是比较麻烦的，但我们可以通过书写相应的脚本来达到方便管理。（这样的管理脚本在网上很多，大家也可以在网上查找）

步骤如下：

- 1) `cd /usr/local/nginx`
- 2) `mkdir bin`
- 3) `vi nginx.sh`
- 4) 内容是：

---

```
#!/bin/sh
NGINX_HOME=/usr/local/nginx
NGINX_SBIN=$NGINX_HOME/sbin/nginx
NGINX_CONF=$NGINX_HOME/conf/nginx.conf
NGINX_PID=$NGINX_HOME/logs/nginx.pid
NGINX_MAXFD=65535
```



```
NGINX_NAME="Nginx"

. /etc/rc.d/init.d/functions

if [! -f $NGINX_SBIN]
then
 echo "$NGINX_NAME startup: $NGINX_SBIN not exists! "
 exit
fi

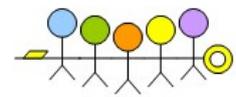
start() {
 ulimit -HSn $NGINX_MAXFD
 $NGINX_SBIN -c $NGINX_CONF
 ret=$?
 if [$ret -eq 0]; then
 action "$Starting $NGINX_NAME: " /bin/true
 else
 action "$Starting $NGINX_NAME: " /bin/false
 fi
}

stop() {
 kill `cat $NGINX_PID`
 ret=$?
 if [$ret -eq 0]; then
 action "$Stopping $NGINX_NAME: " /bin/true
 else
 action "$Stopping $NGINX_NAME: " /bin/false
 fi
}

restart() {
 stop
 start
}

check() {
 $NGINX_SBIN -c $NGINX_CONF -t
}

reload() {
 kill -HUP `cat $NGINX_PID`
}
```



```
relog() {
 kill -USR1 `cat $NGINX_PID`
}

case "$1" in
 start)
 start
 ;;
 stop)
 stop
 ;;
 restart)
 restart
 ;;
 check)
 check
 ;;
 reload)
 reload
 ;;
 relog)
 relog
 ;;
 *)
 echo $"Usage: $0 {start|stop|restart|reload|check|relog}"
 exit 1
esac
```

---

5) `chmod +x nginx.sh`

6) 现在可以通过如下方式来控制 Nginx

|                                                    |              |
|----------------------------------------------------|--------------|
| <code>/usr/local/nginx/bin/nginx.sh start</code>   | 启动           |
| <code>/usr/local/nginx/bin/nginx.sh stop</code>    | 关闭           |
| <code>/usr/local/nginx/bin/nginx.sh restart</code> | 重新启动，即先关闭后启动 |
| <code>/usr/local/nginx/bin/nginx.sh reload</code>  | 重新装载配置文件     |
| <code>/usr/local/nginx/bin/nginx.sh check</code>   | 检查配置文件       |
| <code>/usr/local/nginx/bin/nginx.sh relog</code>   | 重新打开日志文件     |

## 7. 使用注意事项

- 1) 在生产运行中，请注意 Nginx 的日志，定期进行日志文件的归档和截断。  
A 将现有日志文件备份：  
B/usr/local/nginx/bin/nginx.sh relog
- 2) 当要求 reload（重新装载配置文件）时，Nginx 会对配置文件进行检查，如果配置文件有错，那么会继续使用旧的，已装载完毕的配置文件运行。为保证生产系统的稳定运行，修改完配置文件后，必须进行 check 操作，以确保配置文件的正确性。
- 3) 为保证生产系统的稳定运行，Nginx 与 php-cgi 的通讯端口请使用 tcp/ip 方式，而不使用 unix 套接字。虽然 tcp/ip 效率较低，但是相对稳定，而且可以将 php 运行在其他机器上。当 php 应用运行较慢，并发请求多的情况下，使用 unix 套接字容易导致连接失败，从而报告 502 错误（Bad Gateway）。

## Nginx 配置文件详解

接下来，观察一个 Nginx 默认的配置文件的含义

```
#user nobody; ----- 工作进程的属主
#worker_processes 1; ----- 工作进程数，一般与 CUP 的核数相同
#error_log logs/error.log; ----- 这三个是 Nginx 的错误日志
#error_log logs/error.log notice;
#error_log logs/error.log info;
pid logs/nginx.pid; -----Nginx pid 文件存放路径

events {
 #use epoll;-----Nginx 的工作模式，Linux 下性能最好的 event 模式
 worker_connections 1024; -----每个工作进程允许最大的同时连接数
}
-----http, 设定 http 服务器，利用它的反响代理功能，并可以利用它的负载均衡功能
http {
 include mime.types; -----设定 mime 类型
 default_type application/octet-stream;
 -----以下是设定日志格式
 log_format main '$remote_addr - $remote_user [$time_local]
$request '
 # '$status' $body_bytes_sent "$http_referer" '
 # '$http_user_agent' "$http_x_forwarded_for";
```

```
#access_log logs/access.log main; -----日志文件名

sendfile on;
#tcp_nopush on;
#keepalive_timeout 0;
keepalive_timeout 65;
#gzip on; -----是否开启 gzip 功能
-----server 设定虚拟主机
server {
 listen 80;
 server_name localhost;

 #charset koi8-r;

 #access_log logs/host.access.log main;

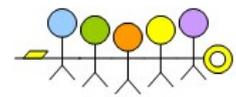
 location / {
 root html;
 index index.html index.htm;
 }

 #error_page 404 /404.html;

 # redirect server error pages to the static page /50x.html
 #
 error_page 500 502 503 504 /50x.html;
 location = /50x.html {
 root html;
 }

 # proxy the PHP scripts to Apache listening on 127.0.0.1:80
 #
 #location ~ /\.php$ {
 # proxy_pass http://127.0.0.1;
 #}

 # pass the PHP scripts to FastCGI server listening on
127.0.0.1:9000
 #
 #location ~ /\.php$ {
 # root html;
 # fastcgi_pass 127.0.0.1:9000;
 # fastcgi_index index.php;
 # fastcgi_param SCRIPT_FILENAME
```



```
/scripts$fastcgi_script_name;
 # include fastcgi_params;
 #}

 # deny access to .htaccess files, if Apache's document root
 # concurs with nginx's one
 #
 #location ~ /\.ht {
 # deny all;
 #}
}
another virtual host using mix of IP-, name-, and port-based
configuration
#
#server {
listen 8000;
listen somename:8080;
server_name somename alias another.alias;

location / {
root html;
index index.html index.htm;
}
#}
HTTPS server
#
#server {
listen 443;
server_name localhost;

ssl on;
ssl_certificate cert.pem;
ssl_certificate_key cert.key;

ssl_session_timeout 5m;

ssl_protocols SSLv2 SSLv3 TLSv1;
#
ssl_ciphers
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
ssl_prefer_server_ciphers on;
location / {
root html;
index index.html index.htm;
}
}
```

```
#}
}
(文件配置内容在逐步更新)
```

从上面 Nginx 默认的配置我们可以知道, 我们可以通过编写相应的模块达到我们想要的效果。

## Nginx 中的全局变量

|                                  |                                 |
|----------------------------------|---------------------------------|
| <code>\$args</code>              | <code>\$remote_port</code>      |
| <code>\$content_length</code>    | <code>\$remote_user</code>      |
| <code>\$content_type</code>      | <code>\$request_filename</code> |
| <code>\$document_root</code>     | <code>\$request_uri</code>      |
| <code>\$document_uri</code>      | <code>\$query_string</code>     |
| <code>\$host</code>              | <code>\$scheme</code>           |
| <code>\$http_user_agent</code>   | <code>\$server_protocol</code>  |
| <code>\$http_cookie</code>       | <code>\$server_addr</code>      |
| <code>\$limit_rate</code>        | <code>\$server_name</code>      |
| <code>\$request_body_file</code> | <code>\$server_port</code>      |
| <code>\$request_method</code>    | <code>\$uri</code>              |
| <code>\$remote_addr</code>       |                                 |

## 通过例子学习

小例子:

1. 禁止访问某个目录下的文件

```
location ~ ^/data {
 deny all;
}
```

2. 禁止多个目录

```
location ~ ^/(cron|templates)/ {
 deny all;
break;
```

```
}
```

### 3. 禁止以/data开头的文件

```
location ~ ^/data {
 deny all;
}
```

### 4. 禁止单个目录

不能禁止.log.txt能请求

```
location /searchword/cron/ {
 deny all;
}
```

### 5. 禁止单个文件

```
location ~ /data/sql/data.sql {
 deny all;
}
```

### 6. 只允许固定的 IP 访问并有密码

```
location / {
 root /opt/htdocs/www;
 allow Address;
 allow Address;
 allow Address;
 deny all;
 auth_basic "CIG_ADMIN";
 auth_basic_user_file htpasswd;
}
```

### 7. 文件和目录不存在的时候重定向:

```
if (!-e $request_filename) {
 proxy_pass http://127.0.0.1;
}
```

(注意: Nginx 不允许 if 嵌套操作)

案例一:

要求:

1) 配置两个虚拟主机, 分别为:

A www.my.com.cn 根连接为/data/html/my, 端口为 80

B www.My\_two.com.cn 根连接为/data/html/my\_two, 端口为 8080)

2) 针对两个域名把 gif, jpg 图片设置过期时间为 1 个月

3)对 html、css、js 进行压缩  
1,2 的实现

```
server {
 listen 80;
 server_name www.my.com.cn;

 charset gb2312;

 access_log /data/my/logs/host.access.log main;

 location / {
 root /data/html/my;
 index index.html index.htm;
 }
 location ~*\.(gif|jpg)$ {
 Expires 30d;
 }
}

server {
 listen 8080;
 server_name www.My_two.com.cn;

 charset gb2312;

 #access_log logs/host.access.log main;

 location / {
 root /data/html/my_two;
 index index.html index.htm;
 }
 location ~*\.(gif|jpg)$ {
 Expires 30d;
 }
}
```

3 的实现

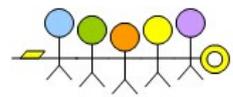
在 http 里添加:

```
gzip on;
gzip_types text/plain text/css text/html text/javascript
```

案例二

利用 Nginx 实现负载均衡, 来自

<http://dianping.blog.51cto.com/427241/95274/>



```
http {
 include 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 off;
 access_log logs/access.log;# 日志文件名
 sendfile on;
 #tcp_nopush on;
 tcp_nodelay on;
 keepalive_timeout 65;
 include gzip.conf;
 # 集群中的所有后台服务器的配置信息
 upstream tomcats {
 server 192.168.0.11:8080 weight=10;
 server 192.168.0.12:8081 weight=10;
 server 192.168.0.13:8080 weight=10;
 server 192.168.0.14:8081 weight=10;
 server 192.168.0.15:8080 weight=10;
 server 192.168.0.16:8081 weight=10;
 }

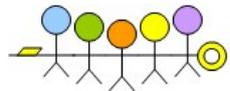
 server {
 listen 80;#HTTP 的端口
 server_name localhost;
 charset utf-8;
 #access_log logs/host.access.log main;
 location ~ ^/NginxStatus/ {
 stub_status on; #Nginx 状态监控配置
 access_log off;
 }

 auth_basic "NginxStatus";
 auth_basic_user_file /usr/local/nginx/conf/htpasswd;

 }

 location ~ ^/(WEB-INF)/ {
 deny all;
 }

 location ~ \.(htm|html|asp|php|gif|jpg|jpeg|png|bmp|ico|rar|css|js|
zip|java|jar|txt|flv|swf|mid|doc|ppt|xls|pdf|txt|mp3|wma)$ {
 root /opt/webapp;
 expires 24h;
 }
}
```



```
 }
 location / {
 proxy_pass [url]http://tomcats;#[/url] 反向代理
 include proxy.conf;
 }
 error_page 404 /html/404.html;
 # redirect server error pages to the static page /50x.html
 #
 error_page 502 503 /html/502.html;
 error_page 500 504 /50x.html;
 location = /50x.html {
 root html;
 }
}
}
```

更具体地书写配置文件，在介绍完 rewrite 后会有更详细的介绍。

## Nginx 监控

要使用 Nginx 的监控功能必须有一个用户和密码，因此我们必需生成一个 htpasswd 文件，这时候我们得用到 Apache 的 htpasswd 工具了。

生成 htpasswd 文件的方法：

```
#htpasswd -c /tmp/htpassswd user
```

输入两次密码就完成了，然后把生成好的 htpasswd 文件拷贝到 proxy 的机器的 /usr/local/nginx/conf 目录下就行了。

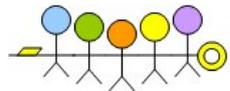
有这个文件后，我们还要配置 conf 文件才能应用这一功能：

在根 {} 下书写以下模块

```
location ~~/NginxStatus{
 stub_status on;
 access_log on;
 auth_basic "NginxStatus";
 Auth_basic_user_file conf/htpasswd;
}
```

这个时候我们可以从浏览器中输入 http://ip/NginxStatus  
输入验证账号后我们就可以看到相关的信息了，例如：

```
Active connection:10
Server accepts handled requests
9309 8982 28890
Reading:1 Writing:3 Waiting:6
```



## Nginx Rewrite 基础

### Rewrite 基本标记

last 相当于Apache里的[L]标记，表示完成rewrite

break 终止匹配，不再匹配后面的规则

redirect 返回 302 临时重定向 地址栏会显示跳转后的地址

permanent 返回 301 永久重定向 地址栏会显示跳转后的地址

### Rewrite 正则表达式匹配

\* ~ 为区分大小写匹配

\* ~\* 为不区分大小写匹配

\* !~和!~\*分别为区分大小写不匹配及不区分大小写不匹配

### Rewrite 条件判断

-f 和!-f 用来判断是否存在文件

-d和!-d用来判断是否存在目录

-e和!-e用来判断是否存在文件或目录

-x和!-x用来判断文件是否可执行

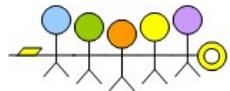
### Rewrite 应用实例

#### 伪地址

```
/123456/xxxx -> /xxxx?id=123456
rewrite ^/(\d+)/(.+)/ /$2?id=$1 last;
```

```
/[0-9]/my.html
rewrite
/(\d+)/my.html /friend/my_list.jsp?uid=$1; break;
```

通过伪地址能提高你网站的安全性，这样暴露给用户的是一个假的地址。



## 域名跳转

### 简单域名跳转

```
server
{
 listen 80;
 server_name sky.com;
 index index.html index.htm index.php;
 root /opt/sky/www;
 rewrite ^/ http://skybin090804.cublog.cn/;
 access_log off;
}
```

### 多域名转向

```
server_name www.sky.com www.sky.net;
 index index.html index.htm index.php;
 root /data/sky;
if ($host ~ "sky\.net") {
 rewrite ^(.*) http://www.sky.com$1 permanent;
}
```

Nginx 相关知识在更新中

## Resin 简介

Resin 是 CAUCHO 公司 (<http://www.caucho.com/>) 的产品，是一个非常流行的支持 Servlets 和 JSP 的引擎，速度非常快。同时，Resin 还支持 JSF、JSTL、EJB Lite、Transaction management、Messaging、Remote services、Distributed sessions、SSL、PHP。总的来说，它是一个速度非常快能力非常强大的 http 服务器。虽然它可以显示动态内容，但是它显示静态内容的能力也非常强，速度直逼 APACHE SERVER。许多站点都是使用该 WEB 服务器构建的。

Resin 也可以和许多其他的 WEB 服务器一起工作，比如 Apache server 和 IIS 等。Resin 支持 Servlets 2.3 标准和 JSP 1.2 标准。熟悉 ASP 和 PHP 的用户可以发现用 Resin 来进行 JSP 编程是件很容易的事情。

Resin 支持负载均衡 (Load balancing)，可以增加 WEB 站点的可靠性。方

法是增加服务器的数量。比如一台 SERVER 的错误率是 1% 的话，那么支持负载平衡的两个 Resin 服务器就可以使错误率降到 0.01%。

现在最新的版本是 Resin 3.1 (Stable)，你可以从 <http://www.caucho.com/download/> 站点上查询 Resin 的最新版本并下载它。

Resin 的特点（翻译于 <http://www.caucho.com>）：

- 1) 快速，可靠性高；
- 2) 能在大部分服务器上部署
- 3) 它是开源的
- 4) 易于安装、编译、管理和监控
- 5) 十年时间的不断的测试和使用的
- 6) 10 个星期作为一个更新周期
- 7) 由 Caucho 开发者直接提供技术支持
- 8) 性能浪费的解决方法

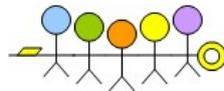
## Resin 安装

### Resin 安装前提

1. 确认系统是否安装了 JDK
2. 确认 JDK 的路径是否已经加到系统变量中（\$JAVA\_HOME）。

### Resin 安装

```
#cd /usr/local (这部是非必要)
#wget xxx/resin-3.0.6.tar.gz 获取 resin 的安装包
#gzip resin-3.0.6.tar.gz
#tar -xvf resin-3.0.6.tar
#cd resin-3.0.6
#./configure
#make && make install
#/usr/local/resin-3.0.6/bin/httpd.sh (启动 resin)
通过 http://ip:8080 访问 /opt/resin-3.0.6/doc/ 验证是否安装成功
```

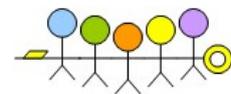


# Resin 配置文件

## 单服务器篇

在\$RESIN\_HOME/conf/sample 下有很多配置模板，可以拿来参考。以下是一个蛋服务器的配置，请更注意红色字体的配置。

```
<resin xmlns="http://caucho.com/ns/resin"
xmlns:resin="http://caucho.com/ns/resin/core">
 <!-- 把 .jar 结尾的文件放在 resin/lib 目录下, resin 会加载 resin/lib 下的所有 .jar
 文件-->
 <class-loader>
 <tree-loader path="{resin.home}/lib"/>
 <tree-loader path="{resin.root}/lib"/>
 </class-loader>
 <!-- - 管理配置
 - Remote management can be enabled by adding jmx-service with
 - with a random string.
 -->
 <management path="{resin.root}/admin">
 </management>
 <!-- -JDK 日志接口的配置. -->
 <log name="" path="stdout:" timestamp="[H:%M:%S.%s]"/>
<!-- --
例如:
<stderr-log path="{resin.home}/log/stderr.log" timestamp="[m-%d %H:%M:%S.%s]
" rollover-size="100mb" rollover-period="1W" rollover-count="20" />
-->
<!-- 日志信息的级别: 'info' 生产环境 'config' 开发环境 'loader' 调试环境 -->
<logger name="com.caucho" level="info"/>
<logger name="com.caucho.java" level="config"/>
<logger name="com.caucho.loader" level="config"/>
<!-- - 环境上下文的检测时间, 对于生产站点, 这个要设置长一点, 例如 600 秒, 10 分
钟 -->
<dependency-check-interval>2s</dependency-check-interval>
(以上是检测 jsp 文件更新重新编译的时间间隔, 按需要提高或降低)
<!-- - 发送邮件通知的 SMTP 服务器 -->
<system-property mail.smtp.host="127.0.0.1"/>
<system-property mail.smtp.port="25"/>
<!-- - 你可以把编译器改成 "javac", "eclipse" 或者 "internal". -->
<javac compiler="internal" args="-source 1.5"/>
(以上配置 java 编译器路径, 必须正确)
<!--
```



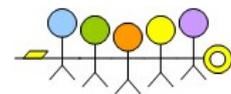
例如

```
<javac compiler="/usr/java/jdk1.6.0_06/bin/javac" args="" />
 -->
 <!-- Security providers.
 - <security-provider>
 - com.sun.net.ssl.internal.ssl.Provider
 - </security-provider>
 -->
 <!-- 去掉注释，如果你使用 resin 提供的 xml 应用
 -
 - <system-property javax.xml.parsers.DocumentBuilderFactory
 - ="com.caucho.xml.parsers.XmlDocumentBuilderFactory"/>
 - <system-property javax.xml.parsers.SAXParserFactory
 - ="com.caucho.xml.parsers.XmlSAXParserFactory"/>
 -->
 <cluster id="app-tier">
 <!-- 设置集群上下文的根，相对于 server.root -->
 <root-directory>.</root-directory>
 <server-default>
 <!-- HTTP 服务的端口-->
 <http address="*" port="8080"/>
 (以上配置端口，不要与已在用的端口冲突，否则 resin 会启动失败)
 <!--
 - SSL 端口配置:
 -
 - <http address="*" port="8443">
 - <openssl>
 - <certificate-file>keys/gryffindor.crt</certificate-file>
 - <certificate-key-file>keys/gryffindor.key</certificate-key-file>
 - <password>test123</password>
 - </openssl>
 - </http>
 -->
 <!-- - JVM 参数设置 -->

 <jvm-arg>-Xms512m</jvm-arg>
 (jvm最小内存，也是启动resin后的默认内存分配值)
 <jvm-arg>-Xmx512m</jvm-arg> <!-- make ms=mx to reduce GC
times -->
```

(jvm 最大内存，当内存使用超过 Xms 分配的值之后会自动向这个最大值提升，一般配置成最大最小值相等，理论上能够降低 GC 垃圾收集的时间，可按实际进行配置)

<jvm-arg>-Xmn86m</jvm-arg> (内存分配增量，当内存需求超过Xms值之后进行第一次分配请求的内存值，一般为Xmx的1/3-1/4；开始时候可以先屏蔽，当应用出现OutOfMemory的时候再打开也可以)



```
<jvm-arg>-XX:MaxNewSize=256m</jvm-arg>
 <jvm-arg>-XX:PermSize=128m</jvm-arg>
 <jvm-arg>-XX:MaxPermSize=256m</jvm-arg>
```

(以上三项是为了减少 **OutOfMemory** 而配置的，是每个 **java** 编译执行的时候最多能一次申请 **jvm** 内存空间的值，以上默认配置基本够用，但依然出 **OutOfMemory** 的时候可以适当调大，但不能超越 **Xmx** 的值；开始时候可以先屏蔽，当应用出现 **OutOfMemory** 的时候再打开也可以)

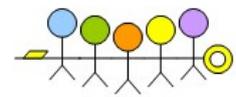
```
<jvm-arg>-Xss256k</jvm-arg> <!-- jvm Stack config -->
 <jvm-arg>-Djava.awt.headless=true</jvm-arg> <(允许使用验证码)>
 <jvm-arg>-Djava.net.preferIPv4Stack=true</jvm-arg> <!-- disable IPv6 -->
 <jvm-arg>-Doracle.jdbc.V8Compatible=true</jvm-arg>
```

(针对 **oracle10** 的兼容配置)

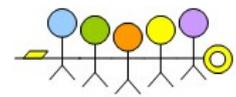
```
<watchdog-arg>-Dcom.sun.management.jmxremote</watchdog-arg>
<!-- 强制 resin 强制重起时的最小空闲内存 -->
<memory-free-min>2M</memory-free-min>
<!-- 最大线程数量. -->
<thread-max>256</thread-max>
<!-- 套接字等待时间 -->
<socket-timeout>65s</socket-timeout>
<!-- 配置 keepalive -->
<keepalive-max>128</keepalive-max>
<keepalive-timeout>15s</keepalive-timeout>
<!-- - 如果使用的是 UNIX, 这里是启动的帐号和用户组.
- <user-name>resin</user-name>
- <group-name>resin</group-name>
-->
</server-default>
<!-- 定义群集服务器 -->
<server id="" address="127.0.0.1" port="6800"/>
```

(以上为 **resin** 的 **server id**，跟启动脚本的 **server name** 匹配，这样启动才会正确，端口要看看当前有没被使用，如果已被使用，则需要改成其他可用的)

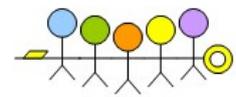
```
<!-- Configures the persistent store for single-server or clustered 配置
独立服务器或者群集的持久化存储，专业版的功能 -->
<resin:if test="{resin.isProfessional()}">
 <persistent-store type="cluster">
 <init path="session"/>
 </persistent-store>
</resin:if>
<!-- 为了安全，你可以为 SSL 会话(SSL sessions)定义一个不同的 cookie.
- <ssl-session-cookie>SSL_JSESSIONID</ssl-session-cookie>
-->
<!-- 缓存启用 (专业版的功能) -->
<resin:if test="{isResinProfessional}">
 <cache path="cache" memory-size="64M">
```



```
<!-- Vary header rewriting for IE -->
<rewrite-vary-as-private/>
</cache>
</resin:if>
<!-- 启用周期性的服务器状态检查和死锁检查，所有的服务器可以添加 <url> 来检查。 -->
<resin:if test="{isResinProfessional}">
 <ping>
 <!-- <url>http://localhost:8080/test-ping.jsp</url> -->
 </ping>
</resin:if>
<!-- 包含 web 应用的默认行为 -->
<resin:import path="{resin.home}/conf/app-default.xml"/>
<!-- 每一个 web 应用的默认参数 -->
<web-app-default>
 <!-- 扩展库的公共 jar 文件，扩展是安全的即使没有类装载器知道的 jars，装载的类将为每个应用分别装载，也就是这些类都是不同的 -->
 <class-loader>
 <tree-loader path="{server.root}/ext-webapp"/>
 </class-loader>
 <!-- 设置缓存页、静态也的延时值 -->
 <cache-mapping url-pattern="/" expires="5s"/>
 <cache-mapping url-pattern="*.gif" expires="60s"/>
 <cache-mapping url-pattern="*.jpg" expires="60s"/>
 <cache-mapping url-pattern="*.png" expires="60s"/>
 <!-- 启用 EL 表达式 -->
 <allow-servlet-el/>
 <!-- 安全原因，默认禁用了会话的 URLs -->
 <session-config>
 <enable-url-rewriting>>false</enable-url-rewriting>
 </session-config>
 <!-- 安全原因，在 cookies 中设置 HttpOnly 标志 -->
 - <cookie-http-only/>
 -->
 <!-- 一些 JSP 包有不正确的 .tld 文件。可以把 validate-taglib-schema 设置成 false，可能继续正常工作 -->
 - Some JSP packages have incorrect .tld files. It's possible to set
 validate-taglib-schema to false to work around these packages.
 -->
 <jsp>
 <validate-taglib-schema>>true</validate-taglib-schema>
 <fast-jstl>>true</fast-jstl>
 <fast-jsf>>true</fast-jsf>
 </jsp>
```



```
</web-app-default>
<!-- 简单的数据池配置
 - The JDBC name is java:comp/env/jdbc/test
 <database>
 <jndi-name>jdbc/mysql</jndi-name>
 <driver type="org.gjt.mm.mysql.Driver">
 <url>jdbc:mysql://localhost:3306/test</url>
 <user></user>
 <password></password>
 </driver>
 <prepared-statement-cache-size>8</prepared-statement-cache-size>
 <max-connections>20</max-connections>
 <max-idle-time>30s</max-idle-time>
 </database>
-->
<!-- 定义所有虚拟主机的默认配置 -->
<host-default>
 <!-- 如果和别的 web 服务器整合, 这个可以被去掉, 因为 web 服务器也可以记录这
 些信息。 -->
 <access-log path="logs/access.log" format='%h %l %u %t "%r" %s %b
 "%{Referer}i" "%{User-Agent}i"' rollover-period="1W"/>
 <!-- war 文件的布置目录 -->
 <web-app-deploy path="webapps"/>
 <!-- ear 文件的布置目录 -->
 <ear-deploy path="deploy">
 <ear-default>
 <ejb-server>
 <config-directory>WEB-INF</config-directory>
 <data-source>jdbc/test</data-source>
 </ejb-server>
 </ear-default>
 </ear-deploy>
 <!-- rar 文件的布置目录 -->
 <resource-deploy path="deploy"/>
</host-default>
<!-- 虚拟主机的布置目录 -->
<host-deploy path="hosts">
 <host-default>
 <resin:import path="host.xml" optional="true"/>
 </host-default>
</host-deploy>
<!-- 默认的虚拟主机配置 -->
<host id="" root-directory=".">
 <!-- 配置默认的应用 webapp's ROOT -->
```



```
<web-app id="/" root-directory="webapps/ROOT"/>
<web-app id="/resin-admin" root-directory="{resin.home}/php/admin">
 <!-- - 管理应用程序 /resin-admin
 - password is the md5 hash of the password. md5 码的密码。
 - localhost is true to limit access to the localhost. localhost 设置
成 true, 这样只有 localhost 才能访问
 -->
 <prologue>
 <resin:set var="resin_admin_user" value=""/>
 <resin:set var="resin_admin_password" value=""/>
 <resin:set var="resin_admin_external" value="false"/>
 </prologue>
</web-app>
</host>
</cluster>
<!-- - Configuration for the web-tier/load-balancer -->
<resin:if test="{resin.isProfessional()}">
 <cluster id="web-tier">
 <server-default>
 <!-- The http port -->
 <http address="*" port="9080"/>
 </server-default>
 <server id="web-a" address="127.0.0.1" port="6700"/>
 <cache path="cache" memory-size="64M"/>
 <host id="">
 <web-app id="/">
 <rewrite-dispatch>
 <load-balance regexp="" cluster="app-tier"/>
 </rewrite-dispatch>
 </web-app>
 </host>
 </cluster>
</resin:if>
</resin>
```

## Resin 中 JVM 的分析

### JVM 虚拟内存

JVM 虚拟内存与系统内存的关系图：



- Xms 初始化jvm的时候从系统内存划分的heap内存数量
- Xmx jvm最大能使用的heap内存数量
- Xmn (-XX:NewSize/ -XX:MaxNewSize) 年轻代占heap的内存数量
- XX:PermSize / -XX:MaxPermSize 持久代占用的系统内存数量
- Xss 单个线程堆栈占的内存量，跟线程数相关

整个JVM的工作流程是：新的线程在Eden生成；Eden满，把有经常访问的资源放在S01中，对Eden进行空间回收；当S01满，将常使用的资源放到S02，对S01的空间进行回收；当S02满，将经常访问的资源放到Old，对S02进行空间回收；按一定的规则对Old的资源进行处理。（为了容易记忆，我们可以对S01、S02、Old起个其他的名字，分别是幸存者1区，幸存者2区和养老区）

简单概括为：Heap设定与垃圾回收Java Heap分为3个区，New，Old和Permanent。New保存刚实例化的对象。当该区被填满时，GC会将对象移到Old区。Permanent区则负责保存反射对象。（Perm区在过去持久化没有使用的年代默认的配置已经足够使用，但现在要注意该区的情况，有可能OOM--out of menery 就是由于它而发生）

从上面的JVM工作机制，我们可以看出对JVM的优化主要在于：JVM--垃圾收集。

注意：Heap大小并不决定进程的内存使用量。进程的内存使用量要大于-Xmx定义的值，因为Java为其他任务分配内存，例如每个线程的Stack等。



## JVM--垃圾收集

JVM在运行过程中，会产生大量的对象和数据，定期丢弃过期和无效的对象和数据，以清理出可用内存给新生成的对象和数据使用，这个过程就是垃圾收集。JVM有2个GC线程。第一个线程负责回收Heap的New区。第二个线程在Heap不足时，遍历Heap，将New区升级为Old区。进行垃圾收集的时候，是需要整个jvm用户线程停止下来进行收集，所采用的垃圾收集器和回收算法会影响这个垃圾收集效率。减少垃圾收集的频率和时间，以令java进程提供更多的时间为用户提供服务，这个是jvm优化的核心。收集算法包括：复制(Copying)、标记清除(mark-sweep)、标记整理(mark-sweep-compact)。JDK的垃圾收集器包括：并发收集器(响应时间优先)、并行收集器(吞吐量优先)、串行收集器(古老)、增量收集器(单cpu使用/已停止维护)。

## JVM 垃圾收集算法

### 复制(Copying)

将堆内分成两个相同空间，从根(ThreadLocal的对象，静态对象)开始访问每一个关联的活跃对象，将空间A的活跃对象全部复制到空间B，然后一次性回收整个空间A。因为只访问活跃对象，将所有活动对象复制走之后就清空整个空间，不用去访问死对象，所以遍历空间的成本较小，但需要巨大的复制成本和较多的内存。

### 标记清除(mark-sweep)

收集器先从根开始访问所有活跃对象，标记为活跃对象。然后再遍历一次整个内存区域，把所有没有标记活跃的对象进行回收处理。该算法遍历整个空间的成本较大暂停时间随空间大小线性增大，而且整理后堆里的碎片很多。

### 标记整理(mark-sweep-compact)

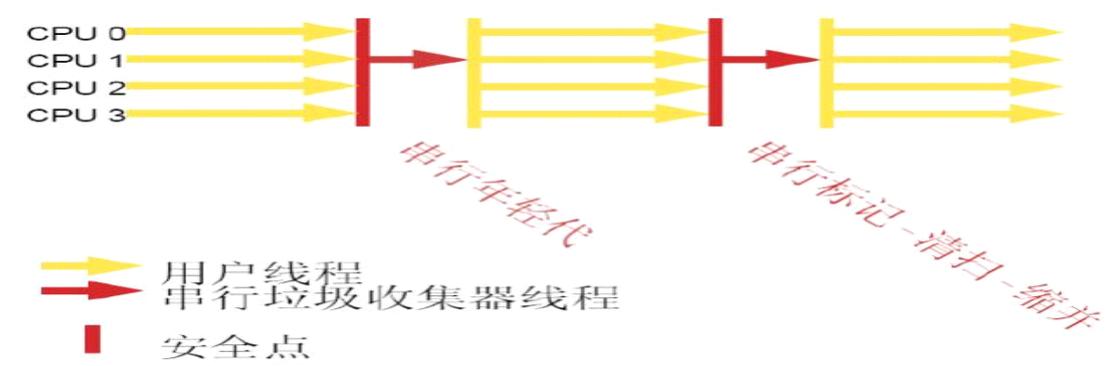
综合了上述两者的做法和优点，先标记活跃对象，然后将其合并成较大的内存块。

## JDK 的四种垃圾收集器

### 串行收集器(Serial Collector)

-XX:+UseSerialGC: 策略为

年轻代串行复制，  
年老代串行标记整理。



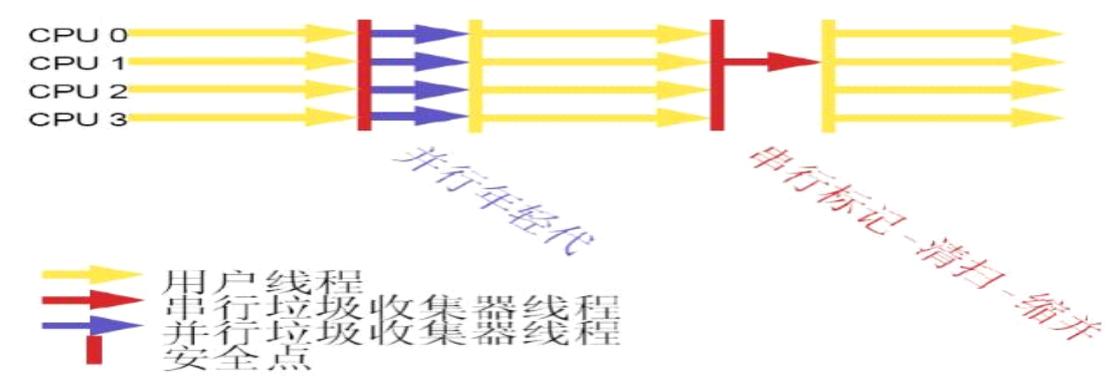
### 吞吐量优先的并行收集器 (Throughput Collector)

XX:+UseParallelGC: 这是JDK5 -server的默认值。策略为:

New: 暂停应用程序, 多个垃圾收集线程并行地复制收集, 线程数默认为CPU个数, CPU很多时, 可用XX:ParallelGCThreads= 设定线程数。

Old: 暂停应用程序, 与串行收集器一样, 单垃圾收集线程标记整理。

如上可知该收集器需要 2+的 CPU 时才会优于串行收集器, 适用于后台处理, 科学计算。



### 暂停时间优先的并发收集器 (Concurrent Low Pause Collector-CMS)

-XX:+UseConcMarkSweepGC , 策略为:

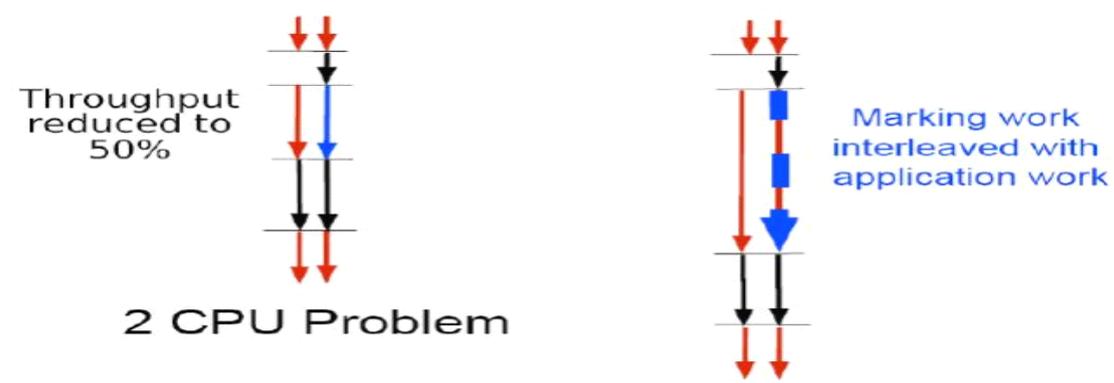
New: 同样是暂停应用程序, 多个垃圾收集线程并行的复制收集。

Old: 则只有两次短暂暂停, 其他时间应用程序与收集线程并发的清除。



## 增量并发收集器 (Incremental Concurrent-Mark-Sweep/i-CMS)

虽然 CMS 收集算法在最为耗时的内存区域遍历时采用多线程并发操作，但对于服务器 CPU 资源不够的情况下，其实对性能是没有提升的，反而会导致系统吞吐量的下降，为了尽量避免这种情况的出现，就有了增量 CMS 收集算法，就是在并发标记、清理的时候让 GC 线程、用户线程交叉运行，尽量减少 GC 线程的全程独占式执行



## 监控 JVM 的工具和使用

Jdk1.5以上版本自带的工具：

jinfo：可以输出并修改运行时的java 进程的opts。

jps：与unix上的ps类似，用来显示本地的java进程，可以查看本地运行着几个java程序，并显示他们的进程号。

jstat：一个极强的监视VM内存工具。可以用来监视VM内存内的各种堆和非堆的

大小及其内存使用量。

jstack: 主要用于线程死锁的监控。(输出thread dump)

jmap:打印出某个java进程(使用pid)内存内的,所有'对象'的情况(如:产生那些对象,及其数量)。

jhat: 主要用于分析jmap产生的dump并提供web页面查看分析结果。

jconsole:一个java GUI监视工具,可以以图表化的形式显示各种数据。并可通过远程连接监视远程的服务器VM。

jstat 是最常用的命令

`${JAVA_HOME}/bin/jstat -h` 可以看到帮助

`jstat [Options] vmid [interval] [count]`

Options — 选项,我们一般使用 `-gcutil` 查看gc情况

vmid — VM的进程号,即当前运行的java进程号

interval - 间隔时间,单位为秒或者毫秒

count — 打印次数,如果缺省则打印无数次

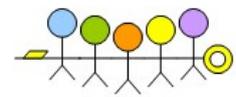
常用参数: `jstat <-gc|-gcutil> <时间间隔 ms> <执行次数> <-h 第几行显示标题栏>` 查看各代的垃圾收集情况

## 操作实例

先使用`#netstat -nlp` 找出使用 jvm 的实例;

Gc 内存大小显示 (k) gcutil 百分比显示

```
/usr/java/jdk1.6.0_06/bin/jstat -gc 2926 3 5
SOC S1C S0U S1U EC EU OC OU PC PU
YGC YGCT FGC FGCT GCT
64.0 64.0 0.5 0.0 896.0 635.4 7196.0 5905.5 12288.0
7308.9 178 0.132 1 0.053 0.185
64.0 64.0 0.5 0.0 896.0 635.4 7196.0 5905.5 12288.0
7308.9 178 0.132 1 0.053 0.185
64.0 64.0 0.5 0.0 896.0 635.4 7196.0 5905.5 12288.0
7308.9 178 0.132 1 0.053 0.185
64.0 64.0 0.5 0.0 896.0 635.4 7196.0 5905.5 12288.0
7308.9 178 0.132 1 0.053 0.185
64.0 64.0 0.5 0.0 896.0 635.4 7196.0 5905.5 12288.0
7308.9 178 0.132 1 0.053 0.185
```



```
/usr/java/jdk1.6.0_06/bin/jstat -gcutil 2926 3 5
S0 S1 E O P YGC YGCT FGC FGCT GCT
0.82 0.00 79.07 82.07 59.48 178 0.132 1 0.053 0.185
0.82 0.00 79.07 82.07 59.48 178 0.132 1 0.053 0.185
0.82 0.00 79.07 82.07 59.48 178 0.132 1 0.053 0.185
0.82 0.00 79.07 82.07 59.48 178 0.132 1 0.053 0.185
0.82 0.00 79.07 82.07 59.48 178 0.132 1 0.053 0.185
```

参数含义如下:

- S0C — Heap上的 Survivor space 0 区共有的大小
- S1C — Heap上的 Survivor space 1 区共有的大小
- S0U — Heap上的 Survivor space 0 区已使用的大小
- S1U — Heap上的 Survivor space 1 区已使用的大小
- EC — Heap上的 Eden space 区共有的大小
- EU — Heap上的 Eden space 区已使用的大小
- OC — Heap上的 Old space 区共有的大小
- OU — Heap上的 Old space 区已使用的大小
- PC — Perm space 区共有的大小
- PU — Perm space 区已使用的大小
- S0 — Heap上的 Survivor space 0 区已使用空间的百分比
- S1 — Heap上的 Survivor space 1 区已使用空间的百分比
- E — Heap上的 Eden space 区已使用空间的百分比
- O — Heap上的 Old space 区已使用空间的百分比
- P — Perm space 区已使用空间的百分比
- YGC — 从应用程序启动到采样时发生 New GC 的次数
- YGCT - 从应用程序启动到采样时 New GC 所用的时间(单位秒)
- FGC — 从应用程序启动到采样时发生 Full GC 的次数
- FGCT - 从应用程序启动到采样时 Full GC 所用的时间(单位秒)
- GCT — 从应用程序启动到采样时用于垃圾回收的总时间(单位秒)

分析:

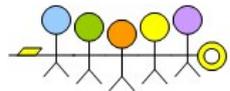
各个的使用率, 始终停留在相同的百分比左右, 说明常驻内存没有突变, 比较正常。

如果young gc和full gc能够正常发生, 而且都能有效回收内存, 常驻内存区变化不明显, 则说明java内存释放情况正常, 垃圾回收及时, java内存泄露的几率就会大大降低。但也不能说明一定没有内存泄露。

这只是个简单的举例, 如果要详细进行分析, 就要搭配不同的参数, 并进行长期的观察这样才能进行判断和调优。

Jdk自带的gc记录参数:

-XX:+PrintGC



输出形式: [GC 118250K->113543K(130112K), 0.0094143 secs]  
          [Full GC 121376K->10414K(130112K), 0.0650971 secs]  
-XX:+PrintGCDetails  
输出形式: [GC [DefNew: 8614K->781K(9088K), 0.0123035 secs]  
118250K->113543K(130112K), 0.0124633 secs]  
          [GC [DefNew: 8614K->8614K(9088K), 0.0000665 secs][Tenured:  
112761K->10414K(121024K), 0.0433488 secs] 121376K->10414K(130112K), 0.0436268  
secs]  
  
-XX:+PrintGCTimeStamps可与上面两个混合使用  
输出形式: 11.851: [GC 98328K->93620K(130112K), 0.0082960 secs]  
  
-XX:+PrintGCApplicationConcurrentTime:打印每次垃圾回收前, 程序未中断的执行时间。  
可与上面混合使用  
输出形式: Application time: 0.5291524 seconds  
  
-XX:+PrintGCApplicationStoppedTime: 打印垃圾回收期间程序暂停的时间。可与上面混合  
使用  
输出形式: Total time for which application threads were stopped: 0.0468229 seconds  
  
-XX:PrintHeapAtGC:打印GC前后的详细堆栈信息  
-Xloggc:filename:与上面几个配合使用, 把相关日志信息记录到文件以便分析。

这些参数都可以配置在 Resin 3.1 的配置文件里面。

## 日常遇到的 OOM 和频繁发生 GC 情况分析

java.lang.OutOfMemoryError: Java heap space  
heap空间不足, 可能是-Xmx配得过大, 或者系统内存不足或泄漏

java.lang.OutOfMemoryError: PermGen space  
持久代内存不足, 存在大量系统类被加载或jpa等架构频繁使用, 需要增加Perm  
的内存配置

java.lang.OutOfMemoryError:unable to create native thread  
空闲内存不足以建立新的线程, 减少max-threads 的配置, 增加空闲内存数量

为什么一些程序频繁发生GC? 有如下原因:

1. 程序内调用了System.gc()或Runtime.gc()。
2. 一些中间件软件调用自己的GC方法, 此时需要设置参数禁止这些GC。
3. Java的Heap太小, 一般默认的Heap值都很小。
4. 频繁实例化对象, Release对象。此时尽量保存并重用对象, 例如使用StringBuffer()和String()。



5. 如果你发现每次GC后，Heap的剩余空间会是总空间的50%，这表示你的Heap处于健康状态。

在 Resin 中对 JVM 的优化能很好的提高 Resin 的性能，通过这些基本知识使你对 Resin 中 JVM 的优化更有把握。

参考文章：

<http://blog.csdn.net/tyrone1979/archive/2006/09/25/1274458.aspx>

<http://www.caucho.com/resin-3.0/performance/jvm-tuning.xtp#garbage-collection>

## Resin 中除 JVM 外的优化

在实际的生产环境中，以下三个东西经常用到，所以在这里提一下，也算是简单的优化。

```
<thread-max>512</thread-max>
```

最大线程数影响resin的系统负载能力以及java进程的内存占用

```
<keepalive-max>128</keepalive-max>
```

keepalive 的最大数量，对网络性能有影响

js/html/css/jpg/gif 等静态文件由nginx提供服务，剩下的由nginx以upstream方式代理到后端resin处理，以减少resin提供这些静态文件访问的性能问题。

Resin 及 jvm 优化，是一项基于提供服务的应用上进行一段相对长时间的测试进行，由于每个项目都有其自身特点，只有根据这些特点来进行优化，才能更好的把该项目配置得更好，不可能硬套到其它项目上。

参考文章：

<http://www.opendigest.org/article.php/450>

欢迎大家增加这方面你的经验



# Memcached 介绍

Memcache是一个高性能的分布式的内存对象缓存系统,通过在内存里维护一个统一的巨大的hash表,它能够用来存储各种格式的数据,包括图像、视频、文件以及数据库检索的结果等。Memcache是danga.com的一个项目,最早是为LiveJournal 服务的,最初为了加速 LiveJournal 访问速度而开发的,后来被很多大型的网站采用。目前全世界不少人使用这个缓存项目来构建自己大负载的网站,来分担数据库的压力。起初作者编写它可能是为了提高动态网页应用,为了减轻数据库检索的压力,来做的这个缓存系统。它的缓存是一种分布式的,也就是可以允许不同主机上的多个用户同时访问这个缓存系统,这种方法不仅解决了共享内存只能是单机的弊端,同时也解决了数据库检索的压力,最大的优点是提高了访问获取数据的速度!基于memcache作者对分布式cache的理解和解决方案。memcache完全可以用到其他地方比如分布式数据库,分布式计算等领域。本介绍转摘自『IT学习者』<http://www.itlearner.com/article/2009/4327.Shtml>

小知识: LiveJournal 团队开发了包含 Memcached、MogileFS、Perlbai 等不错的开源产品。

英文介绍:

## What is Memcached?

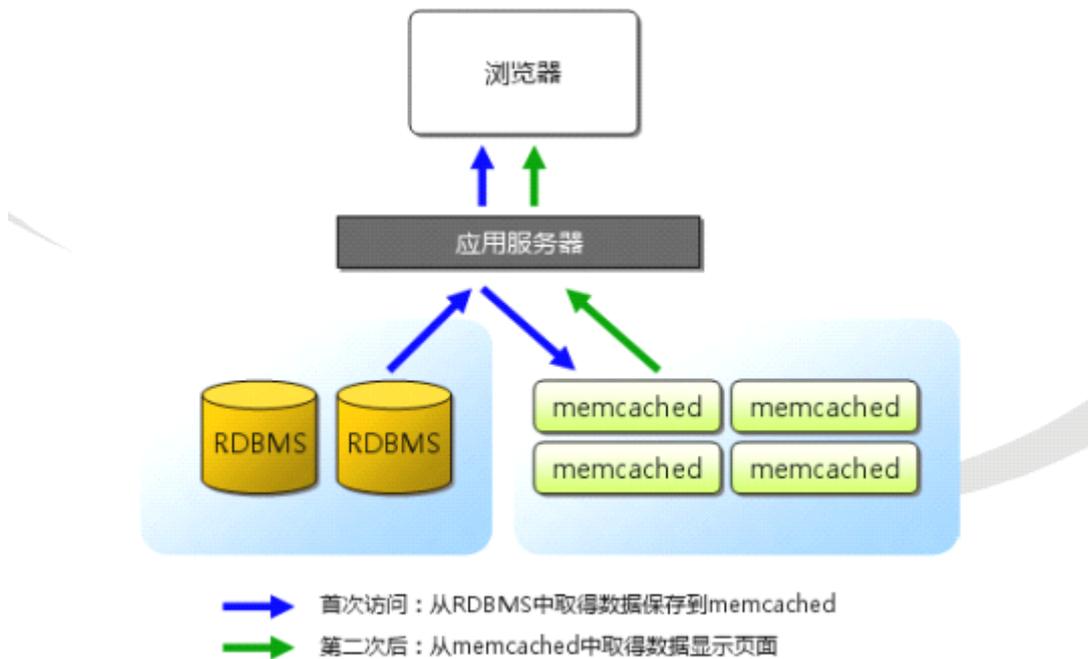
Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

Memcached is simple yet powerful. Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. Its [API](#) is available for most popular languages.

本介绍来自: <http://memcached.org/>

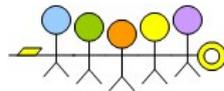
## Memcached 运行图:



此图来自：<http://www.bookfm.com/courseware/coursewaredetail.html?cid=100250>

## Memcached 特点

- 1) 基于 C/S 架构，协议简单
- 2) 基于 Libevent 事件处理 (libevent 是一套跨平台的事件处理接口的封装)
- 3) 自主的内存存储处理



# MemCached 安装

## 安装前提

Memcached 依赖 Libevent, 因此必须先编译安装 Libevent。两者都需要安装在 2.6 以上内核版本的 GNU/Linux 系统中。编译前, 请先确认 gcc、make、patch 等编译工具是否已安装, 并可正常使用。

## 安装前资源的准备

软件清单列表:

libevent	libevent-1.4.8-stable.tar.gz
memcached	memcached-1.2.6.tar.gz

把安装软件上传到服务器目录 (本实例放在: /usr/local/src)

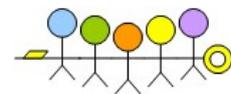
## 安装步骤

```
#cd /usr/local/src
#tar -xf libevent-1.4.8-stable.tar.gz
#tar -xf memcached-1.2.6.tar.gz
#cd libevent-1.4.8-stable
#./configure && make && make install
#cd ../memcached-1.2.6
#./configure && make && make install
```

编译完成后, memcached 被安装在 /usr/local/bin/memcached。

Memcached 常用命令行参数如下:

	说明	备注
-l	监听的地址	memcached 无身份验证功能, 严禁在无防护状态下, 直接监听外网端口!!! 默认 11211
-p	监听的端口	
-d	以 daemon 形式运行	一般皆需增加此参数
-u	以何用户身份运行	一般选 nobody 等低权用户
-m	最大可用内存	以兆为单位
-c	最大的同时并发数	1024 默认
-f	增长因子	参见资料中关于内存使用的章节



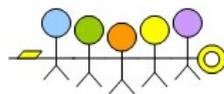
-P	PID 文件	
----	--------	--

这里只列出了我们常用的一些参数。

更详细的参数如下：

```
memcached -h
memcached 1.2.8
-p <num> TCP port number to listen on (default: 11211)
-U <num> UDP port number to listen on (default: 11211, 0 is off)
-s <file> unix socket path to listen on (disables network support)
-a <mask> access mask for unix socket, in octal (default 0700)
-l <ip_addr> interface to listen on, default is INDRR_ANY
-d run as a daemon
-r maximize core file limit
-u <username> assume identity of <username> (only when run as root)
-m <num> max memory to use for items in megabytes, default is 64 MB
-M return error on memory exhausted (rather than removing items)
-c <num> max simultaneous connections, default is 1024
-k lock down all paged memory. Note that there is a
 limit on how much memory you may lock. Trying to
 allocate more than that would fail, so be sure you
 set the limit correctly for the user you started
 the daemon with (not for -u <username> user;
 under sh this is done with 'ulimit -S -l NUM_KB').
-v verbose (print errors/warnings while in event loop)
-vv very verbose (also print client commands/reponses)
-h print this help and exit
-i print memcached and libevent license
-P <file> save PID in <file>, only used with -d option
-f <factor> chunk size growth factor, default 1.25
-n <bytes> minimum space allocated for key+value+flags, default 48
-R Maximum number of requests per event
 limits the number of requests process for a given con nection
 to prevent starvation. default 20
-b Set the backlog queue limit (default 1024)
```

启动:./memcached -d -m 300 -p 11211 -u root -c 4096



## 安装中可能出现的问题

启动memcache时有可能出现以下错误:

```
memcached: error while loading shared libraries: libevent-1.4.so.2:
cannot open shared object file: No such file or directory
```

系统无法定位libevent库, 此时可以用如下方法解决问题:

```
ln -s /usr/local/lib/libevent-1.4.so.2 /lib/libevent-1.4.so.2
```

如果是64位操作系统, 需要指向/lib64目录

```
ln -s /usr/local/lib/libevent-1.4.so.2 /lib64/libevent-1.4.so.2
```

再次启动 memcache, OK。

## 正在更新中

## 附录

### Nginx 编译模块名称解析

- prefix=PATH 设定安装目录
- sbin-path=PATH 设定程序文件目录
- conf-path=PATH 设定配置文件(nginx.conf)目录
- error-log-path=PATH 设定错误日志目录
- pid-path=PATH 设定 pid 文件(nginx.pid)目录
- lock-path=PATH 设定 lock 文件(nginx.lock)目录
- user=USER 设定程序运行的用户环境(www)
- group=GROUP 设定程序运行的组环境(www)
- builddir=DIR 设定程序编译目录
- with-rtsig\_module 允许 rtsig 模块
- with-select\_module 允许 select 模块(一种轮询模式,不推荐用在高载环境)
- without-select\_module 不使用 select 模块
- with-poll\_module 允许 poll 模块(一种轮询模式,不推荐用在高载环境)
- without-poll\_module 不使用 poll 模块
- with-http\_ssl\_module 允许 ngx\_http\_ssl\_module 模块(Apache 对应:mod\_ssl)
- with-http\_realip\_module 允许 ngx\_http\_realip\_module 模块(mod\_rpaf)
- with-http\_addition\_module 允许 ngx\_http\_addition\_module 模块(mod\_layout)
- with-http\_xslt\_module 允许 ngx\_http\_xslt\_module 模块
- with-http\_sub\_module 允许 ngx\_http\_sub\_module 模块
- with-http\_dav\_module 允许 ngx\_http\_dav\_module 模块(mod\_dav)
- with-http\_flv\_module 允许 ngx\_http\_flv\_module 模块(mod\_flvx)
- with-http\_gzip\_static\_module 允许 ngx\_http\_gzip\_static\_module 模块(mod\_dflate)



- with-http\_random\_index\_module 允许 ngx\_http\_random\_index\_module 模块(mod\_autoindex)
- with-http\_stub\_status\_module 允许 ngx\_http\_stub\_status\_module 模块(mod\_status)
- without-http\_charset\_module 不使用 ngx\_http\_charset\_module 模块
- without-http\_gzip\_module 不使用 ngx\_http\_gzip\_module 模块
- without-http\_ssi\_module 不使用 ngx\_http\_ssi\_module 模块
- without-http\_userid\_module 不使用 ngx\_http\_userid\_module 模块
- without-http\_access\_module 不使用 ngx\_http\_access\_module 模块
- without-http\_auth\_basic\_module 不使用 ngx\_http\_auth\_basic\_module 模块
- without-http\_autoindex\_module 不使用 ngx\_http\_autoindex\_module 模块
- without-http\_geo\_module 不使用 ngx\_http\_geo\_module 模块
- without-http\_map\_module 不使用 ngx\_http\_map\_module 模块
- without-http\_referer\_module 不使用 ngx\_http\_referer\_module 模块
- without-http\_rewrite\_module 不使用 ngx\_http\_rewrite\_module 模块
- without-http\_proxy\_module 不使用 ngx\_http\_proxy\_module 模块
- without-http\_fastcgi\_module 不使用 ngx\_http\_fastcgi\_module 模块
- without-http\_memcached\_module 不使用 ngx\_http\_memcached\_module 模块
- without-http\_limit\_zone\_module 不使用 ngx\_http\_limit\_zone\_module 模块
- without-http\_empty\_gif\_module 不使用 ngx\_http\_empty\_gif\_module 模块
- without-http\_browser\_module 不使用 ngx\_http\_browser\_module 模块
- without-http\_upstream\_ip\_hash\_module  
不使用 ngx\_http\_upstream\_ip\_hash\_module 模块
- with-http\_perl\_module 允许 ngx\_http\_perl\_module 模块
- with-perl\_modules\_path=PATH 设置 perl 模块路径
- with-perl=PATH 设置 perl 库文件路径
- http-log-path=PATH 设置 access log 文件路径
- http-client-body-temp-path=PATH 设置客户端请求临时文件路径
- http-proxy-temp-path=PATH 设置 http proxy 临时文件路径
- http-fastcgi-temp-path=PATH 设置 http fastcgi 临时文件路径
- without-http 不使用 HTTP server 功能
- with-mail 允许 POP3/IMAP4/SMTP 代理模块
- with-mail\_ssl\_module 允许 ngx\_mail\_ssl\_module 模块
- without-mail\_pop3\_module 不允许 ngx\_mail\_pop3\_module 模块
- without-mail\_imap\_module 不允许 ngx\_mail\_imap\_module 模块
- without-mail\_smtp\_module 不允许 ngx\_mail\_smtp\_module 模块
- with-google\_perftools\_module 允许 ngx\_google\_perftools\_module 模块(调试用)
- with-cpp\_test\_module 允许 ngx\_cpp\_test\_module 模块
- add-module=PATH 允许使用外部模块,以及路径
- with-cc=PATH 设置 C 编译器路径
- with-cpp=PATH 设置 C 预处理路径
- with-cc-opt=OPTIONS 设置 C 编译器参数
- with-ld-opt=OPTIONS 设置连接文件参数
- with-cpu-opt=CPU 为指定 CPU 优化,可选参数有:  
pentium, pentiumpro, pentium3, pentium4,  
athlon, opteron, sparc32, sparc64, ppc64

- without-pcre 不使用 pcre 库文件
- with-pcre=DIR 设定 PCRE 库路径
- with-pcre-opt=OPTIONS 设置 PCRE 运行参数
- with-md5=DIR 设定 md5 库文件路径
- with-md5-opt=OPTIONS 设置 md5 运行参数
- with-md5-asm 使用 md5 源文件编译
- with-sha1=DIR 设定 sha1 库文件路径
- with-sha1-opt=OPTIONS 设置 sha1 运行参数
- with-sha1-asm 使用 sha1 源文件编译
- with-zlib=DIR 设定 zlib 库文件路径
- with-zlib-opt=OPTIONS 设置 zlib 运行参数
- with-zlib-asm=CPU 使 zlib 对特定的 CPU 进行优化,可选参数:  
pentium, pentiumpro
- with-openssl=DIR 设定 OpenSSL 库文件路径
- with-openssl-opt=OPTIONS 设置 OpenSSL 运行参数
- with-debug 允许调试日志

介绍一个 Nginx 外部模块:

#### ngx\_http\_accesskey\_module

封堵下载软件的好东东,只有远程 IP 地址符合加密字串的才被允许访问.示例如下:

```
location /download {
accesskey on;
accesskey_hashmethod md5;
accesskey_arg "key";
accesskey_signature "mypass$remote_addr";
}
```

模块地址:

<http://wiki.codemongers.com/NginxHttpAccessKeyModule>

其它更多外部模块请访问:

<http://wiki.codemongers.com/NginxModules>

在 OpenBSD 环境下安装 Nginx 需要注意下,在./configure 之后还需要修改一下源文件,否则后面 make 过不去:

```
vi +74 src/os/unix/nginx_posix_config.h
```

将这行的 malloc.h 改成 stdlib.h,保存退出.

然后再执行 make && make install 即可!< begin ajax random post data >