Varnish Documentation

Version：varnish-2.1.3 released
http://varnish-cache.org/docs/2.1/
文档版本 varnish-cn.0.9-bate

译者：andy.feng
网名：fh.cn
Email：lr@isadba.com

目录

关于本手册

本手册由 andy.feng 于 2010/6 月初开始翻译，使用的是最新版本的官方文档和最新的软件包为基准，由于 2.0.6 版本后，varnish 做了很大改动，包括一些设计理念，VCL 配置文件语法等，和以前的版本出现了一些不可兼容性。而且 varnish 团队也保证以后不会出现这种大规模的改动。

感谢：我要感谢我的女朋友 mandy.gu，对我工作的支持和谅解（因为花费了很多陪她的时间）。还要感谢公司老大，boyod.yang 的支持，以及朋友刘东林，彬帅。感谢松涛为 PDF 做的封面，以及“varnish 中文项目”QQ 群和很多 IT 朋友的帮助与支持，最后还要感谢 51cto.com 的管理员们的支持和推广。

申明：这是我第一次翻译英文技术文档，一个是官方手册本来有一些病句和错误的单词，以及错误的代码。另外就是一些术语我也没法理解，所以翻译有误，或者不能看懂的情况下情参照官方文档。当然也欢迎你对此文档提出修改意见。我会整理后发，发布修订版，相信有您的参与，varnish 会给更多中国 IT 人带来福音。

版本规定：

version-0.8　　翻译首搞（已完成）

version-0.9　　译者校对（已完成）

version-1.0-beta　　此版本为当前文档版本，文档还不完善，请大家把发现的问题发送到

指定 bug 收集处，我会整理发布 release 版本。
本文章指定问题反馈和拍砖地址：http://linuxguest.blog.51cto.com/195664/354889
version-1.0-release　　网友反馈，更新到文档中（进行中）

Andy.feng 2010/9/2

欢迎阅读 varnish's 中文文档！

varnish 是一款先进的网站加速器，他的任务是代理并缓存他后面 web 服务器的内容。加速访问您的 web 站点。我们建议您先阅读 “varnish 安装”。如果您的 varnish 已经成功启动，您可以去阅读“varnish 指南”。

正文：

一、Varnish 安装

●安装 varnish

它是一个开源软件，你可以选择安全二进制包，或者从源码定制编译安装。安装二进制包还是源码包是个人习惯。如果您不知道选择哪种方式，请阅读整个文档，然后选择一种您喜欢的方式。

通过源码包还是通过二进制包安装？

在相关系统上，可以使用系统自带的包管理器来安装，常见的用例：

FreeBSD

源码安装：

```
cd /usr/ports/varnish && make install clean
```

二进制安装：

```
pkg_add –r varnish
```

CentOS/RedHat

我们尽力维护最近的版本提供 RPMS（EL4&EL5） on SourceForge。Varnish 被包含在 EPEL 资源库，不幸的是 varnish2.0.6→varnish2.1.X 有语法改变。这样我们就不能更新 EPEL 中的 varnish 版本，EPEL 中最后的版本是 2.0.6.

DEBIAN/UBUNTU

varnish 已经发布了 DEBINA 和 UBUNTU 的包，只需要使用一下命令就可以安装，注意：这样安装可能不是最新的版本。

apt-get install varnish

OTHER SYSTEMS

您最好使用源码安装，参照“源码编译安装“

如果您已经完成了安装，您可以阅读“varnish 指南“，“varnish 指南”比安装更加有趣。

●通过源码包编译安装

如果没有您系统适用的二进制包，或其他原因您想要通过源码编译，请参考以下步骤：

首先需要使用 svn 命令下载源码。如果您没有这个命令，您需要先在您的系统上安装 subversion 软件，笔者使用的是二进制安装的 subversion。

下载当前的 2.1 分支版本

svn co http://varnish-cache.org/svn/branches/2.1

下载当前的开发源码

svn co http://varnish-cache.org/svn/trunk

DEBIN/UBUNTU 系统环境下的依耐关系

要在 DEBIN/UBUNTU 系统下成功编译安装 varnish，需要先安装以下软件包：

〇autotools-dev
〇automake1.9
〇libtool
〇autoconf
〇libncurses-dev
〇xsltproc
〇groff-base
〇libpcre3-dev

使用以下命令安装上面所有的包

sudo apt-get install autotools-dev automake1.9 libtool autoconf libncurses-dev xsltproc groff-base libpcre3-dev

RedHat/CentOS 系统环境下的依耐关系

如果您是 RedHat/CentOS 系统想安装 varnish，您需要安装以下软件包：

〇automake
〇autoconf
〇libtool
〇ncurses-devel
〇libxslt
〇groff
〇pcre-devel
〇pkgconfig

以下是在配置好 yum 包管理器的情况下运行

```
yum install automake autoconf libtool ncurses-devel libxslt groff pcre-devel pkgconfig
```

配置和编译

下一步，配置，配置时会检查软件的依耐关系是否满足。

```
cd varnish-cache
sh autogen.sh
sh configure
make
```

这里的 configure 命令可以使用一些参数，但是大多数情况下和上面一样不使用参数。您可以忘记这一步，因为几乎所有的参数都可以在 varnish 运行的时候添加。

在您安装 varnish 前，您可以运行自动测试，然后去喝杯茶，因为它会花费您几分钟。

```
cd bin/varnishtest && ./varnishtest tests/*.vtc
```

如果您发现有一两个失败的时候，请不用担心，某些测试项目对时间比较敏感（您可以告诉我们，我们来解决这个问题）。如果出现很多错误，或者 b00000.vtc 测试也失败的话，应该是发生了某些可怕的错误，如果您不解决这个错误的话，接下来将一事无成。

INSTALLING

最后的考验，您是否有一颗勇敢的心：

```
make install
```

varnish 将被安装到/usr/local 目录，varnishd 可执行文件将被安装到/usr/local/sbin/，默认配置文件被安装到/usr/local/etc/varnish/default.vcl。

●获得支持

关于直接获取 varnish 团队的支持，我们会在时间允许的情况下尽量多的帮助大家，并试图尽可能的简化这一过程。

但是请在联系我们前花一点时间，整理您的想法和明白表达您的问题，如果您只告诉我们“我的 varnish 不能工作了”，而没有进一步的信息，这将是毫无意义的。

IRC CHANNEL

最直接的获得我们帮助的方法就是加入我们的 IRC 通道。

#varnish on server irc.linpro.no

含义:时区是欧洲+美国

如果您要发表您的 VCL 或者相关文档，可以使用 http://gist.github.com/

MAILING LISTS

打开或关闭邮件列表请访问 MailMan http://lists.varnish-cache.org/mailman/listinfo

COMMERCIAL SUPPORT

商业支持，请联系 sales@varnish-software.com

●提交 BUG

Varnish 的 debug 工作就像一个棘手的禽兽，有可能一些数据的拥挤，导致成千上

万的线程核心转储。

如果您发现一个 bug，花一点时间收集 bug 的相关信息是十分必要且必须的，我们可以通过您收集的信息来修复这个 bug。

二、varnish 指南

Varnish 是一个 web 加速器，被安装在 web 应用程序前面，缓存 web 应用程序，并响应用户请求，varnish 让您的 web 应用程序运行的更快，并且 varnish 灵活好用。

这个指南没有覆盖所有的 varnish 函数和功能，它可以让您对 varnish 有个全面的认识并且掌握如何运用它。

我们假设您有一个 web 服务器和 web 应用程序正在运行，而且您想使用 varnish 给您的 web 运行程序加速。

此外我们还假设您已经阅读了“varnish 安装”并且按照默认配置安装了 varnish。

●Backend servers(后端服务器)

varnish 有一个概念叫做“后端服务器”或者叫“原点服务器”，一个后端服务器将提供 varnish 加速的内容。

我们的第一个任务就是告诉 varnish 在哪里可以找到他要的内容。使用您喜欢的文本编辑器打开 varnish 默认的配置文件，如果您是源码安装的配置文件可能在 /usr/local/etc/varnish/default.vcl，如果您是二进制包安装的配置文件可能在 /etc/varnish/default.vcl.

配置文件的最顶端如下：

```
# backend default {
#       .host = "127.0.0.1";
#       .port = "8080";
# }
```

我们取消前面的注释，并且把 8080 端口改成 80 端口。如下

```
backend default {
    .host = "127.0.0.1";
    .port = "80";
}
```

现在，这块配置定义了一个 varnish 默认访问的后端服务器，当 varnish 需要从后端服务器获取内容的时候，它就会访问自己（127.0.0.1）的 80 端口。

Varnish 可以定义多个后端服务器而且您可以通过定义多个后端服务器达到负载均衡的目的。

现在我们完成了基本的 varnish 配置，我们可以在 8080 端口上启动 varnish，并做一些基本的测试。

●Starting Varnish（启动 varnish）

假设 varnishd 在您的环境变量中，您可能需要运行 pkill varnishd 来确定 varnish 没有运行。然后使用 root 执行下面的命令。

varnishd -f /usr/local/etc/varnish/default.vcl -s malloc,1G -T 127.0.0.1:2000 -a 0.0.0.0:8080

我添加了一些选项，现在来详细分析他们：

-f /usr/local/etc/varnish/default.vcl

这个 –f 选项指定 varnishd 使用哪个配置文件。

-s malloc，1G

这个 –s 选项用来确定 varnish 使用的存储类型和存储容量，我使用的是 malloc 类型（malloc 是一个 C 函数，用于分配内存空间）， 1G 定义多少内存被 malloced，1G = 1gigabyte。

-T 127.0.0.1：2000

Varnish 有一个基于文本的管理接口，启动它的话可以在不停止 varnish 的情况下来管理 varnish。您可以指定管理软件监听哪个接口。当然您不能让全世界的人都能访问您的 varnish 管理接口，因为他们可以很轻松的通过访问 varnish 管理接口来获得您的 root 访问权限。我推荐只让它监听本机端口。如果您的系统里有您不完全信任的用户，您可以通过防火墙规则来限制他访问 varnish 的管理端口。

-a 0.0.0.0：8080

这一句的意思是制定 varnish 监听所有 IP 发给 8080 端口的 http 请求，如果在生产环境下，您应该让 varnish 监听 80，这也是默认的。

现在您的 varnish 已经在运行了，现在我们来验证它是否工作正常，在留言其中输入 http://192.168.2.2:8080/ ，您应该可以看见您的 web 程序在这里运行。

使用 varnish 后，web 应用程序是否加速，取决于一些原因。如果您的程序的每个会话都使用 cookies 或者您每个程序都需要三次握手认证，这样 varnish 就不能缓存更多的数据，我们暂时忽略这个问题，等到“提高缓存命中率”这节的时候我们再继续讨论这个问题。

想要知道 varnish 对您的网站做了什么，请查看 logs。

●Logging in varnish（记录数据）

Varnish 一个真正的特点就是它如何记录数据的。使用内存段代替普通的日志文件，当内存段使用完以后，又从头开始，覆盖最旧的记录。这样就可以非常快的记录数据，，并且不需要磁盘空间。

缺点就是您没有把数据写到磁盘上，可能会消失。（varnish 也支持将数据写到硬盘的文件上，看您如何选择）

Varnishlog 这个程序可以查看 varnish 记录了哪些数据。Varnishlog 给您生成原始的日志，包括所有的事件。我一会给您演示。

在运行了 varnish 的终端窗口上，运行 varnishlog 这个命令。

您可以看见如下显示

```
0 CLI            - Rd ping
0 CLI            - Wr 200 PONG 1277172542 1.0
```

这是检查 varnish 的主进程是否正常，如果看见这就说明一切 OK

现在您去浏览器通过 varnish 重新访问您的 web 程序，您将看到如下信息：

```
11 SessionOpen   c 127.0.0.1 58912 0.0.0.0:8080
11 ReqStart      c 127.0.0.1 58912 595005213
11 RxRequest     c GET
11 RxURL         c /
11 RxProtocol    c HTTP/1.1
11 RxHeader      c Host: localhost:8080
11 RxHeader      c Connection: keep-alive
```

第一列是任意的数，它用来定义请求，相同的号码代表相同的 HTTP 传输。

第二列是信息标记，所有的日志都带有一个标记（tag），标记对应相对的操作。Rx 表示 varnish 收到数据，Tx 表示 varnish 发送数据。

第三列代表数据是从哪里传出或者传入的，是从 c（client）还是 b（backend）。

第四列是被记录的数据。

现在，您可以过滤掉相当多的 varnishlog，这些基本的选项，您需要知道：

-b　　\\只显示 varnish 和 backend server 之间的日志，当您想要优化命中率的时候可以使用这个参数。

-c　　\\和-b 差不多，不过它代表的是 varnish 和 client 端的通信。

-i tag　　\\只显示某个 tag，比如“varnishlog –i SessionOpen”将只显示新会话，注意，这个地方的 tag 名字是不区分大小写的。

-I　　\\通过正则表达式过滤数据，比如“varnishlog -c -i RxHeader -I Cookie”将显示所有接到到来自客户端的包含 Cookie 单词的头信息。

-o　　\\聚合日志请求 ID

如果 varnish 一切运行 OK，我们就可以把它调整到 80 端口上。

●PutVarnish on port 80(使 varnish 工作在 80 端口上)

如果您的程序正常运行，没有问题，我们就可以把 varnish 调整到 80 端口运行。先关闭 vernish

```
pkill varnishd
```

然后停止您的 web 服务器，修改 web 服务器配置，把 web 服务器修改成监听 8080 端口，然后修改 varnish 的 default.vcl 和改变默认的后端服务器端口为 8080.

先启动您的 web 服务器，然后在启动 varnish：

```
varnishd    -f /usr/local/etc/varnish/default.vcl    -s malloc,1G -T 127.0.0.1:2000
```

我们取消了-a 选项，这样 varnish 将监控默认端口，启动后，检查您的 web 程序是否正常。

●Varnish Configuration Language - VCL（varnish 配置语言-VCL）

Varnish 有一个很棒的配置系统，大部分其他的系统使用配置指令，让您打开或者关闭一些开关。Varnish 使用区域配置语言，这种语言叫做“VCL”（varnish configuration language），

在执行 vcl 时，varnish 就把 VCL 转换成二进制代码。

VCL 文件被分为多个子程序，不同的子程序在不同的时间里执行，比如一个子程序在接到请求时执行，另一个子程序在接收到后端服务器传送的文件时执行。

varnish 将在不同阶段执行它的子程序代码，因为它的代码是一行一行执行的，不存在优先级问题。随时可以调用这个子程序中的功能并且当他执行完成后就退出。

如果到最后您也没有调用您的子进程中的功能，varnish 将执行一些内建的 VCL 代码，这些代码就是 default.vcl 中被注释的代码。

99%的几率您需要改变 vcl_recv 和 vcl_fetch 这两个子进程。

vcl_recv

vcl_recv（当然，我们在字符集上有点不足，应为它是 unix）在请求的开始被调用，在接收、解析后，决定是否响应请求，怎么响应，使用哪个后台服务器。

在 vcl_recv 中，您可以修改请求，比如您可以修改 cookies，添加或者删除请求的头信息。

注意 vcl_recv 中只有请求的目标,req is available。

vcl_fetch

vcl_fetch 在一个文件成功从后台获取后被调用，通常他的任务就是改变 response headers，触发 ESI 进程，在请求失败的时候轮询其他服务器。

在 vcl_fetch 中一样的包含请求的 object，req，available，他们通常是 backend response，beresp。beresp 将会包含后端服务器的 HTTP 的头信息

actions

主要有以下动作

pass \\当一个请求被 pass 后，这个请求将通过 varnish 转发到后端服务器，但是它不会被缓存。pass 可以放在 vcl_recv 和 vcl_fetch 中。

lookup \\当一个请求在 vcl_recv 中被 lookup 后，varnish 将从缓存中提取数据，如果缓存中没有数据，将被设置为 pass，不能在 vcl_fetch 中设置 lookup。

pipe \\pipe 和 pass 相似，都要访问后端服务器，不过当进入 pipe 模式后，在此连接未关闭前，后续的所有请求都发到后端服务器（这句是我自己理解后简化的，有能力的朋友可以看看官方文档，给我提修改建议）。

deliver \\请求的目标被缓存，然后发送给客户端

esi \\ESI-process the fetched document（我理解的就是 vcl 中包换一段 html 代码）

Requests，Responses and objects

在 VCL 中，有 3 个重要的数据结构

request 从客户端进来

responses 从后端服务器过来

object 存储在 cache 中

在 VCL 中，你需要知道以下结构

req \\请求目标，当 varnish 接收到一个请求，这时 req object 就被创建了，你在 vcl_recv 中的大部分工作，都是在 req object 上展开的。

beresp \\后端服务器返回的目标，它包含返回的头信息，你在 vcl_fetch 中的大部分工作都是在 beresp object 上开展的。

obj \\被 cache 的目标，只读的目标被保存于内存中，obj.ttl 的值可修改，其他的只能读。

Operaors

VCL 支持以下运算符，请阅读下面的例子：

```
=               \\赋值运算符
==              \\对比
~               \\匹配，在 ACL 中和正则表达式中都可以用
!               \\否定
&&              \\逻辑与
||              \\逻辑或
```

EXAMPLE 1 – manipulation headers

我们想要取消我们服务器上/images 目录下的所有缓存：

```
sub vcl_recv {
      if (req.url ~ "^/images") {
            unset req.http.cookie;
            }
}
```

现在，当这个请求在操作后端服务器时，将不会有 cookie 头，这里有趣的行是 if-statement，它匹配 URL，如果匹配这个操作，那么头信息中的 cookie 就会被删除。

EXAMPLE 2 – manipulation beresp

从后端服务器返回对象的值满足一些标准，我们就修改它的 TTL 值：

```
sub vcl_fetch {
    if (beresp.url ~ "\.(png|gif|jpg)$") {
       unset beresp.http.set-cookie;
       beresp.ttl = 3600;
   }
}
```

EXAMPLE3-ACLs

你创建一个 VCL 关键字的访问控制列表。你可以配置客户端的 IP 地址

```
# Who is allowed to purge....
acl local {
      "localhost";
      "192.168.1.0"/24; /* and everyone on the local network */
      ! "192.168.1.23"; /* except for the dialin router */
}

sub vcl_recv {
   if (req.request == "PURGE") {
      if (client.ip ~ local) {
           return(lookup);
      }
   }
}

sub vcl_hit {
    if (req.request == "PURGE") {
```

```
        set obj.ttl = 0s;
        error 200 "Purged.";
    }
}

sub vcl_miss {
  if (req.request == "PURGE") {
    error 404 "Not in cache.";
  }
}
```

●Statistics（统计 varnish 相关数据）

现在您的 varnish 已经正常运行，我们来看一下 varnish 在做什么，这里有些工具可以帮助您做到。

Varnishtop

Varnishtop 工具读取共享内存的日志,然后连续不断的显示和更新大部分普通日志。

适当的过滤使用 –I，-i，-X 和-x 选项，它可以按照您的要求显示请求的内容，客户端，浏览器等其他日志里的信息。

varnishtop -i rxurl　　\\您可以看到客户端请求的 url 次数。

Varnishtop -i txurl　　\\您可以看到请求后端服务器的 url 次数。

Varnishtop -i Rxheader –I Accept-Encoding \\可以看见接收到的头信息中有有多少次包含 Accept-Encoding。

Varnishhist

Varnishhist 工具读取 varnishd 的共享内存段日志，生成一个连续更新的柱状图，显示最后 N 个请求的处理情况。这个 N 的值是终端的纵坐标的高度，横坐标代表的是对数，如果缓存命中就标记“|”，如果缓存没有命中就标记上“#”符号。

Varnishsizes

Varnishsizes 和 varnishhist 相似，除了 varnishsizes 现实了对象的大小，取消了完成请求的时间。这样可以大概的观察您的服务对象有多大。

Varnishstat

Varnish 有很多计数器，我们计数丢失率，命中率，存储信息，创建线程，删除对象等，几乎所有的操作。Varnishstat 将存储这些数值，在优化 varnish 的时候使用这个命令。

有一个程序可以定期轮询 varnishstat 的数据并生成好看的图表。这个项目叫做 Munin。Munin 可以在 http://munin-monitoring.org/找到。在 varnish 的源码中有 munin 插件。

●Achiveving a high hitrate（提高缓存命中率）

现在 varnish 已经正常运行了,您可以通过 varnish 访问到您的 web 应用程序。如果您的 web 程序在设计时候没有考虑到加速器的架构，那么您可能有必要修改您的应用程序或者 varnish 配置文件，来提高 varnish 的命中率。

既然这样，您就需要一个工具用来观察您和 web 服务器之间 HTTP 头信息。服务器

端您可以轻松的使用 varnish 的工具，比如 varnishlog 和 varnishtop，但是客户端的工具需要您自己去准备，下面是我经常使用的工具。

Varnistop

您可以使用 varnishtop 确定哪些 URL 经常命中后端。Varnishtop –i txurl 就是一个基本的命令。您可以通过阅读“Statistics”了解其他示例。

Varnishlog

当您需要鉴定哪个 URL 被频繁的发送到后端服务器，您可以通过 varnishlog 对请求做一个全面的分析。varnishlog –c –o /foo/bar 这个命令将告诉您所有（-o）包含”/football/bar”字段来自客户端（-c）的请求。

Lwp-request

Lwp-request 是 www 库的一部分，使用 perl 语言编写。它是一个真正的基本程序，它可以执行 HTTP 请求，并给您返回结果。我主要使用两个程序，GET 和 HEAD。

Vg.no 是第一个使用 varnish 的站点，他们使用 varnish 相当完整，所以我们来看看他们的 HTTP 头文件。我们使用 GET 请求他们的主页：

```
$ GET -H 'Host: www.vg.no' -Used http://vg.no/
GET http://vg.no/
Host: www.vg.no
User-Agent: lwp-request/5.834 libwww-perl/5.834

200 OK
Cache-Control: must-revalidate
Refresh: 600
Title: VG Nett - Forsiden - VG Nett
X-Age: 463
X-Cache: HIT
X-Rick-Would-Never: Let you down
X-VG-Jobb: http://www.finn.no/finn/job/fulltime/result?keyword=vg+multimedia
Merk:HeaderNinja
X-VG-Korken: http://www.youtube.com/watch?v=Fcj8CnD5188
X-VG-WebCache: joanie
X-VG-WebServer: leon
```

OK，我们来分析它做了什么。GET 通过发送 HTTP 0.9 的请求，它没有主机头，所以我需要添加一个主机头使用-H 选项，-U 打印请求的头，-s 打印返回状态，-e 答应返回状态的头，-d 丢弃当前的连接。我们正真关心的不是连接，而是头文件。

如您所见 VG 的头文件中有相当多的信息，比如 X-RICK-WOULD-NEVER 是 vg.no 定制的信息，他们有几分奇怪的幽默感。其他的内容，比如 X-VG-WEBCACHE 是用来调试错误的。

核对一个站点是否使用 cookies，可以使用下面的命令：

```
GET -Used http://example.com/ |grep ^Set-Cookie
```

Live HTTP Headers

这是一个 firefox 的插件，live HTTP headers 可以查看您发送的和接收的 http 头。软件在 https://addons.mozilla.org/en-US/firefox/addon/3829/下载。或者 google “Live HTTP headers”。

The Role of HTTP headers

Varnish 认为自己是真正的 web 服务器，因为它属于您控制。IETF 没有真正定义 surrogate origin cache 角色的含义，（The role of *surrogate origin cache* is not really well defined by the IETF so RFC 2616 doesn't always tell us what we should do.不知如何翻译）

Cache-Control

Cache-control 指示缓存如何处理内容，varnish 关心 max-age 参数，并使用这个参数计算每个对象的 TTL 值。

“cache-control：nocache” 这个参数已经被忽略，不过您可以很容易的使它生效。

在头信息中控制 cache-control 的 max-age，您可以参照下面，varnish 软件管理服务器的例子：

```
$ GET -Used http://www.varnish-software.com/|grep ^Cache-Control
Cache-Control: public, max-age=600
```

Age

Varnish 添加了一个 age 头信息，用来指示对象已经被保存在 varnish 中多长时间了。您可以在 varnish 中找到 Age 信息：

```
varnishlog -i TxHeader -I ^Age
```

Overriding the time-to-live（ttl）

有时候后端服务器会当掉，也许是您的配置问题，很容易修复。不过更简单的方法是修改您的 ttl，能在某种程度上修复难处理的后端。

您需要在 VCL 中使用 beresp.ttl 定义您需要修改的对象的 TTL：

```
sub vcl_fetch {
      if (req.url ~ "^/legacy_broken_cms/") {
            set beresp.ttl = 5d;
      }
}
```

Cookies

现在 Varnish 接收到后端服务器返回的头信息中有 Set-Cookie 信息的话，将不缓存。所以当客户端发送一个 Cookie 头的话，varnish 将直接忽略缓存，发送到后端服务器。

这样的话有点过度的保守，很多站点使用 Google Analytics（GA）来分析他们的流量。GA 设置一个 cookie 跟踪您，这个 cookie 是客户端上的一个 java 脚本，因此他们对服务器不感兴趣。

对于一个 web 站点来说，忽略一般 cookies 是有道理的，除非您是访问一些关键部分。这个 VCL 的 vcl_recv 片段将忽略 cookies，除非您正在访问/admin/：

```
if ( !( req.url ~ “^/admin/” ) ) {
   unset req.http.Cookie;
}
```

很简单，不管您需要做多么复杂的事情，比如您要删除一个 cookies，这个事情很困难，varnish 也没有相应的工具来处理，但是我们可以使用正则表达式来完成这个工作，如果您熟悉正则表达式，您将明白接下来的工作，如果您不会我建议您找找相关资料学习一下。

我们来看看 varnish 软件是怎么工作的，我们使用一些 GA 和一些相似的工具产生 cookies。所有的 cookies 使用 jsp 语言。Varnish 和企业网站不需要这些 cookies，而 varnish 会因为这些 cookies 而降低命中率，我们将放弃这些多余的 cookies，使用 VCL。

下面的 VCL 将会丢弃所有被匹配的 cookies。

```
// Remove has_js and Google Analytics __* cookies.
set req.http.Cookie = regsuball(req.http.Cookie, "(^|;\s*)(_[_a-z]+|has_js)=[^;]*", "");
// Remove a ";" prefix, if present.
set req.http.Cookie = regsuball(req.http.Cookie, "^;\s*", "");
```

下面的例子将删除所有名字叫 COOKIE1 和 COOKIE2 的 cookies：

```
sub vcl_recv {
   if (req.http.Cookie) {
      set req.http.Cookie = ";" req.http.Cookie;
      set req.http.Cookie = regsuball(req.http.Cookie, "; +", ";");
      set req.http.Cookie = regsuball(req.http.Cookie, ";(COOKIE1|COOKIE2)=", "; \1=");
      set req.http.Cookie = regsuball(req.http.Cookie, ";[^ ][^;]*", "");
      set req.http.Cookie = regsuball(req.http.Cookie, "^[; ]+|[; ]+$", "");

      if (req.http.Cookie == "") {
            remove req.http.Cookie;
      }
}
```

这个例子是来自 varnish wiki 的。

Vary

各式各样的头被发送到 web server，他们让 HTTP 目标多样化。Accept-Encoding 头就有这种感觉，当一个服务器分发一个“Vary：Accept-Encoding”给 varnish。Varnish 需要 cache 来自客户端的每个不同的 Accept-Encoding。如果客户端只接收 gzip 编码，varnish 不对其他编码服务，那么就可以缩减编码量。

问题就是这样的，Accept-Encoding 字段包含很多编码方式，下面是不同浏览器发送的：

```
Accept-Encodign: gzip,deflate
```

另一个浏览器发送的：

```
Accept-Encoding:: deflate, gzip
```

Varnish 可以使两个不同的 accept-enconding 头标准化，这样就可以尽量减少变体。下面的 VCL 代码可以是 accept-encoding 头标准化：

```
if (req.http.Accept-Encoding) {
      if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)$") {
            # No point in compressing these
            remove req.http.Accept-Encoding;
      } elsif (req.http.Accept-Encoding ~ "gzip") {
            set req.http.Accept-Encoding = "gzip";
      } elsif (req.http.Accept-Encoding ~ "deflate") {
            set req.http.Accept-Encoding = "deflate";
      } else {
            # unkown algorithm
            remove req.http.Accept-Encoding;
      }
}
```

这段代码设置客客户端发送的 accept-encoding 头只有 gzip 和 default 两种编码，gzip 优先。

Pitfall – Vary：User-Agent

一些应用或者一些应用服务器发送不同 user-agent 头信息，这让 varnish 为每个单独的用户保存一个单独的信息，这样的信息很多。一个版本相同的浏览器在不同的操作系统上也会产生最少 10 种不同的 user-agent 头信息。如果您不打算修改 user-agent，让他们标准化，您的命中率将受到严重的打击，使用上面的代码做模板。

Pragma

http 1.0 服务器可能会发送“Pragma：nocache”。 Varnish 忽略这个头，您可以轻松的使用 VCL 来完成这个任务：

In vcl_fetch：

```
if (beresp.http.Pragma ~ "nocache") {
    pass;
}
```

Authorization

如果 varnish 收到一个认证请求的头，他将 pass 这个请求，如果您不打算对这个头做任何操作的话。

Normalizing your namespace

有些站点访问的主机名有很多，比如 http://www.varnish-software.com，http://varnish-software.com，http://varnishsoftware.com 所有的地址都对应相同的一个站点。但是 varnish 不知道，varnish 会缓存每个地址的每个页面。您可以减少这种情况，通过修改 web 配置文件或者通过以下 VCL：

```
if (req.http.host ~ "^(www.)?varnish-?software.com") {
  set req.http.host = "varnish-software.com";
}
```

Purging

增加 TTL 值是提高命令率的一个好方法，如果用户访问到的内容是旧的，这样就会对您的商务照成影响。

解决方法就是当有新内容提供的时候通知 varnish。可以通过两种机制 HTTP purging 和 bans。首先，我们来看一个清理的实例：

HTTP purges

HTTP purges 和 HTTP GET 请求相似，除了这是用来 purges 的。事实上您可以在任何您喜欢的时间使用这个方法，但是大多数人使用它 purging。Squid 支持相同的机制，为了让 varnish 支持 purging，您需要在 VCL 中做如下配置：

```
acl purge {
        "localhost";
        "192.168.55.0/24";
}

sub vcl_recv {
        # allow PURGE from localhost and 192.168.55...

        if (req.request == "PURGE") {
```

```
                if (!client.ip ~ purge) {
                        error 405 "Not allowed.";
                }
                return (lookup);
        }
}

sub vcl_hit {
        if (req.request == "PURGE") {
                # Note that setting ttl to 0 is magical.
                # the object is zapped from cache.
                set obj.ttl = 0s;
                error 200 "Purged.";
        }
}

sub vcl_miss {
        if (req.request == "PURGE") {

                error 404 "Not in cache.";
        }
}
```

您可以看见，使用了新的 VCL 子程序，vcl_hit 和 vcl_miss。当您调用 lookup 时将在缓存中查找目标，结果只会是 miss 或者 hit，然后对应的子程序就会被调用，如果 vcl_hit 的目标存储在缓存中，并且可用，我们可以修改 TTL 值。

所以对于 vg.no 的无效首页，他们使用 varnish 做如下处理：

```
PURGE / HTTP/1.0
Host: vg.no
```

如果 varnish 想要丢弃主页，如是很多相同 URL 的变体在 cache 中，只有匹配的变体才会被清除。清除一个相同页面的 gzip 变体可以使用下面命令：

```
PURGE / HTTP/1.0
Host: vg.no
Accept-Encoding: gzip
```

Bans

这是另外一种清空无效内容的方法，bans。您可以认为 bans 是一种过滤方法，您可以禁止某些存在 cache 中存在的数据。您可以基于我们拥有的元数据来禁止。

Varnish 内置的 CLI 接口就是支持 bans 的。禁止 vg 网站上的所有 png 目标代码如下：

```
purge req.http.host == "vg.no" && req.http.url ~ "\.png$"
```

是不是很强大？

在没有被 bans 命中之前的 cache，是能够提供服务的。一个目标只被最新的 bans 检查。如果您有很多长 TTL 的目标在缓存中，您需要知道执行很多的 Bans 对性能照成的影响。

您也可以在 varnish 中添加 bans，这样做需要一点 VCL：

```
sub vcl_recv {
        if (req.request == "BAN") {
                # Same ACL check as above:
                if (!client.ip ~ purge) {
                        error 405 "Not allowed.";
                }
                purge("req.http.host == " req.http.host
                        "&& req.url == " req.url);

                # Throw a synthetic page so the
                # request wont go to the backend.
                error 200 "Ban added"
        }
}
```

这是一个实用 varnish 的 VCL 处理 ban 的方法。添加一个 ban 在 URL 上，包含它的主机部分。

●advanced backend configuration（后端服务器高级配置）

在某些时刻您需要 varnish 从多台服务器上缓存数据。您可能想要 varnish 映射所有的 URL 到一个单独的主机或者不到这个主机。这里很多选项。

我们需要引进一个 java 程序进出 php 的 web 站点。假如我们的 java 程序使用的 URL 开始于/JAVA/

我们让它运行在 8000 端口，现在让我们看看默认的 default.vcl：

```
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}
```

我们添加一个新的 backend：

```
backend java {
    .host = "127.0.0.1";
    .port = "8000";
}
```

现在我们需要告诉特殊的 URL 被发送到哪里：

```
sub vcl_recv {
    if (req.url ~ "^/java/") {
        set req.backend = java;
    } else {
        set req.backend = default.
    }
}
```

这真的很简单，让我们停下来并思考一下。正如您所见，可以通过任意的后端来选择您要的数据。您想发送移动设备的请求到不同的后端？没问题

```
if (req.User-agent ~ /mobile/) ....        \\这样做应该就可以成功。
```

●Directors

您可以把多台 backends 聚合成一个组，这些组被叫做 directors。这样可以增强性能和弹力。您可以定义多个 backends 和多个 group 在同一个 directors。

```
backend server1 {
      .host = "192.168.0.10";
}
backend server2{
      .host = "192.168.0.10";
}
```

现在我们创建一个 director：

```
director example_director round-robin {
{
            .backend = server1;
}
# server2
{
            .backend = server2;
}
# foo
}
```

这个 director 是一个循环的 director。它的含义就是 director 使用循环的方式把 backends 分给请求。

但是如果您的一个服务器宕了？varnish 能否指导所有的请求到健康的后端？当然可以，这就是健康检查在起作用了。

●Health checks(健康检查)

让我们设置一个 director 和两个后端，然后加上健康检查：

```
backend server1 {
  .host = "server1.example.com";
  .probe = {
            .url = "/";
            .interval = 5s;
            .timeout = 1 s;
            .window = 5;
            .threshold = 3;
      }
  }
```

```
backend server2 {
    .host = "server2.example.com";
    .probe = {
            .url = "/";
            .interval = 5s;
            .timeout = 1 s;
            .window = 5;
            .threshold = 3;
    }
 }
```

这些新的就是探针，varnish 将检查通过探针检查每个后端服务器是否健康：

url　　\\哪个 url 需要 varnish 请求。

Interval　　\\检查的间隔时间

Timeout　　\\等待多长时间探针超时

Window　　\\varnish 将维持 5 个 sliding window 的结果

Threshold　　\\至少有 3 次.windows 检查是成功的，就宣告 backends 健康

现在我们定义 director：

```
director example_director round-robin {
        {
                .backend = server1;
        }
        # server2
        {
                .backend = server2;
        }

        }
```

您的站点在您需要的时候使用这个 director，varnish 不会发送流量给标志为不健康的主机。如果所有的 backends 都宕掉了，varnish 可以照常服务。参照“Misbehaving servers”获得更多的信息。

●Misbehaving servers（服务器停止运转）

Varnish 的一个关键特色就是它有能力防御 web 和应用服务器宕机。

Grace mode

当几个客户端请求同一个页面的时候，varnish 只发送一个请求到后端服务器，然后让那个其他几个请求挂起等待返回结果，返回结果后，复制请求的结果发送给客户端。

如果您的服务每秒有数千万的点击率，那么这个队列是庞大的，没有用户喜欢等待服务器响应。为了使用过期的 cache 给用户提供服务，我们需要增加他们的 TTL，保存所有 cache 中的内容在 TTL 过期以后 30 分钟内不删除，使用以下 VCL：

```
sub vcl_fetch {
   set beresp.grace = 30m;
```

```
}
```

Varnish 还不会使用过期的目标给用户提供服务，所以我们需要配置以下代码，在 cache 过期后的 15 秒内，使用旧的内容提供服务：

```
sub vcl_recv {
   set req.grace = 15s;
}
```

你会考虑为什么要多保存过去的内容 30 分钟？当然，如果你使用了健康检查，你可以通过健康状态设置保存的时间：

```
if (! req.backend.healthy) {
    set req.grace = 5m;
} else {
    set req.grace = 15s;
}
```

Saint mode

有时候，服务器很古怪，他们发出随机错误，您需要通知 varnish 使用更加优雅的方式处理它，这种方式叫神圣模式（saint mode）。Saint mode 允许您抛弃一个后端服务器或者另一个尝试的后端服务器或者 cache 中服务陈旧的内容。让我们看看 VCL 中如何开启这个功能的：

```
sub vcl_fetch {
   if (beresp.status == 500) {
      set beresp.saintmode = 10s;
      restart;
   }
   set beresp.grace = 5m;
}
```

当我们设置 beresp.saintmode 为 10 秒，varnish 在 10 秒内将不会访问后端服务器的这个 url。如果有一个备用列表，当重新执行此请求时您有其他的后端有能力提供此服务内容，varnish 会尝试请求他们，当您没有可用的后端服务器，varnish 将使用它过期的 cache 提供服务内容。

它真的是一个救生员。

God mode

还未应用。

●advanced topics（重要的话题）

这里指南涵盖所有 varnish 基本的东西。如果您熟读上面的内容，您现在已经可以使用 varnish。

下面是一个简单的概括，没有完全覆盖指南。

More VCL

VCL 是一个比较复杂的，我们已经讨论至今。这里还有很多可用的子程序和一些动作我们没有讨论。关于完整的 VCL 手册，请参见 man page。

Using Inline C to extend Varnish

使用内置的 C 延伸 varnish，如果您在 varnish 使用这种方法要小心，c 语言运行在 varnish 内部，如果您的 c 语言有问题，那么 varnish 可能会宕掉。

首先使用 C 语言记录日志到 syslog：

```
# The include statements must be outside the subroutines.

C{
        #include <syslog.h>
}C

sub vcl_something {
        C{
                syslog(LOG_INFO, "Something happened at VCL line XX.");
        }C
}
```

Edge side Includes

Varnish 可以在 cache 中创建一个 web 页面和其他页面不放在一起，这个片段有个特殊的缓存策略，如果您的网站有一个列表显示您最受欢迎的 5 篇文章。如果您的网站有这个页面，您可以制造一个缓存包括其他所有的页面。使用得当，可以大大提高您的命中率，减少对服务器的负载。ESI 代码如下：

```
<HTML>
<BODY>
The time is: <esi:include src="/cgi-bin/date.cgi"/>
at this very moment.
</BODY>
</HTML>
```

在 vcl_fetch 中使用 esi 关键字：

```
sub vcl_fetch {
    if (req.url == "/test.html") {
        esi;    /* Do ESI processing */
    }
}
```

●Troubleshooting varnish（varnish 排错）

有时候 varnish 会出错，为了使您知道该检查哪里，您可以检查 varnishlog，/var/log/syslog/,var/log/messages 这里可以发现一些信息，知道 varnish 怎么了。

When varnish won't start

有些时候，varnish 不能启动。这里有很多 varnish 不能启动的原因，通常我们可以观看/dev/null 的权限和是否其他软件占用了端口。

使用 debug 模式启动 varnish，然后观看发生了什么：

```
varnishd -f /usr/local/etc/varnish/default.vcl -s malloc,1G -T 127.0.0.1:2000 -a 0.0.0.0:8080 –d
```

提示-d 选项，它将给您更多的信息关于接下来发生了什么。让我们看看如果其他

程序暂用了 varnish 的端口，它将显示什么：

```
# varnishd -n foo -f /usr/local/etc/varnish/default.vcl -s malloc,1G -T 127.0.0.1:2000
-a 0.0.0.0:8080 -d
storage_malloc: max size 1024 MB.
Using old SHMFILE
Platform: Linux,2.6.32-21-generic,i686,-smalloc,-hcritbit
200 193
-----------------------------
Varnish HTTP accelerator CLI.
-----------------------------
Type 'help' for command list.
Type 'quit' to close CLI session.
Type 'start' to launch worker process.
```

现在 varnish 的主程序已经运行，在 debug 模式中，cache 现在还没有启动，现在您在终端中使用“start”命令来让主程序开启 cache 功能

```
start
bind(): Address already in use
300 22
Could not open sockets
```

在这里，我们发现一个问题。Varnish 要使用的端口被 HTTP 使用了。

Varnish is creashing

当 varnish 宕掉的时候。

Varnish gives me guru mediation

首先查找 varnishlog，这里可能会给您一些信息。

Varnish doesn’t cache

请参考“提高命中率”这章。

三、Varnish 参考手册

●VCL（varnish configuration language）

Author: Dag-Erling Smørgrav
Author: Poul-Henning Kamp
Author: Kristian Lyngstøl
Author: Per Buer
Date: 2010-06-02
Version: 1.0
Manual section: 7

DESCRIPTION 描述

VCL 语言是 varnish （HTTP 加速器）的一种限定域语言，目的在于规定请求的处理和内容的缓存策略。

当一个新的配置文件被加载，varnish 管理进程把 vcl 转换成 c 代码，然后编译成动态共享库连接到服务器进程。

SYNTAX 语法

VCL 的语法相当简单，和 c，perl 相似。使用花括号做界定符，使用分号表示声明结束。注释和 C、C++、perl 语法一样，你可以自己选择。

除此之外还类似 c 语法，比如赋值（=）、比较（==）、和一些布尔值（!、&&、||），VCL 支持正则表达式，ACL 匹配使用 ~ 操作。

不同于 C 和 perl 的地方，反斜杠(\)在 VCL 中没有特殊的含义。只是用来匹配 URLs，所以没有反斜线，请大家自由使用正则表达式。

把所有的字符串都连接在一起，并不对他们做任何操作。

分配和介绍设置关键字，VCL 没有用户定义的变量，只能给 backend、请求、内容这些目标的变量赋值，这些内容大部分是手工输入得，而且给这些变量分配值的时候，必须有一个 VCL 兼容的单位

VCL 有 if 测试，但是没有循环。

Backend declarations（声明 backend）

一个 backend 申明创建和初始化一个 backend 目标：

```
backend www {
    .host = "www.example.com";
    .port = "http";
}
```

一个请求可以选着一个 Backend：

```
if (req.http.host ~ "^(www.)?example.com$") {
    set req.backend = www;
}
```

为了避免后端服务器过载，.max_connections 可以设置连接后端服务器得最大限制数。

在 backend 中申明的 timeout 参数可以被覆盖，.connect_timeout 等待连接后端的时间，.first_byte_timeout 等待从 backend 传输过来的第一个字符的时间，.between_bytes_timeout 两个字符的间隔时间。

示例：

```
backend www {
  .host = "www.example.com";
  .port = "http";
  .connect_timeout = 1s;
  .first_byte_timeout = 5s;
  .between_bytes_timeout = 2s;
}
```

DIRECTORS

Directors 基于健康状态和 per-director 算法选择不同的客户端。现在存在随机和循环两种 director。

定义 Directors：

```
director b2 random {
   .retries = 5;
   {
      // We can refer to named backends
      .backend = b1;
      .weight    = 7;
   }
   {
      // Or define them inline
      .backend    = {
         .host = "fs2";
      }
   .weight              = 3;
   }
}
```

The random director

任意的 director 使用.retries.这个参数指定查找可用后端的次数。默认 director 中的所有后端的.retries 相同。

每个后端的选项 .weight ,和发送多少流量到这个后端有关。

THE round-robin director

Round-robin 没有什么选项。

THE DNS director

DNS director 有三种不同的方法应用在后端，比如 random 或者 round-robin 或者使用.list 中的任意一种：

```
director directorname dns {
            .list = {
                        .host_header = "www.example.com";
                        .port = "80";
                        .connection_timeout = 0.4;
                        "192.168.15.0"/24;
                        "192.168.16.128"/25;
            }
            .ttl = 5m;
            .suffix = "internal.example.net";
}
```

这段代码会制定 384 个后端，都使用 80 端口及 0.4s 的连接超时，.list 声明中设置选项必须在 IPS 的前面。

.ttl 定义 DNSlookups 的时间。

上面的示例中添加了“internal.example.net”引入客户端可以在 looking up 之前提供主机头。所有选项可选。

Bckend probes（后端探针）

探测后端，确定他们是否健康。返回的状态使用 req.backend.healthy 核对。.window 我们检查到的最近的 polls 数量。.threshold 多少 polls 成功，我们就认为后端是健康的。.intial 是多少探针确定 varnish 状态正常，默认和 threshold 值一样。

一个后端定义探针示例：

```
backend www {
   .host = "www.example.com";
  .port = "http";
   .probe = {
      .url = "/test.jpg";
      .timeout = 0.3 s;
      .window = 8;
      .threshold = 3;
      .initial = 3;
   }
}
```

他可以指定原始的 http 请求：

```
backend www {
   .host = "www.example.com";
   .port = "http";
   .probe = {
      # NB: \r\n automatically inserted after each string!
      .request =
         "GET / HTTP/1.1"
         "Host: www.foo.bar"
         "Connection: close";
   }
}
```

ACLS

一个 ACL 给一个访问控制列表命名，随后可以通过调用 ACL 名字来匹配对应的客户端地址。

```
acl local {
   "localhost";              // myself
   "192.0.2.0"/24;           // and everyone on the local network
   ! "192.0.2.23";           // except for the dialin router
}
```

如果一个 ACL 中指定一个主机名，varnish 不能解析，他将解析匹配到所有地址。果你使用了一个否定标记（!），那么将拒绝匹配所有主机。

下面是一个匹配的示例：

```
if (client.ip ~ local) {
   pipe;
}
```

GRACE

如果后端需要很长时间来生成一个对象，这里有一个线程堆积的风险。为了避免这种情况，你可以使用 Grace。他可以让 varnish 提供一个存在的版本，然后从后端生成新的目标版本。

下面的 VCL 代码将 varnish 为客户端提供过期目标，所有对象被保持 2 分钟，在他们失效前新的目标会被制造：

```
sub vcl_recv {
   set req.grace = 2m;
}
sub vcl_fetch {
   set beresp.grace = 2m;
}
```

FUNCTIONS

下面这些内置的函数可以使用的：

Regsub（str，regex，sub）

返回一个第一次匹配 regex 表达式的 str 的复制，使用 sub 代替它，sub,0 就是替换所有满足条件的字符串。

Regsuball（str，regex，sub）

替换所有发现的目标

Purge_url（regex）

清除所有 cache 中的匹配 regex 的 URLS 目标。

Subroutines

一个子程序就是一串可读和可用的代码。

```
sub pipe_if_local {
   if (client.ip ~ local) {
      pipe;
   }
}
```

子程序在 VCL 中没有参数，也没有返回值。

调用一个子程序，使用子程序的关键字名字：

call pipe_if_local;

这里有很多子程序和 varnish 的工作流程相关。这些子程序会检查和操作 http 头文件和各种各样的请求。决定哪个哪些请求被使用。

vcl_recv

在请求开始的时候被调用，当一个完整的请求被接受到，并被解析，它的作用就是决定是否给这个请求提供服务，怎么服务，如果服务，哪个后端会被选取

vcl_recv 子程序以下面的关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pass \\转换到 pass 模式，控制权会传递给 vcl_pass。

pipe \\转换到 pipe 模式，控制权会传递给 vcl_pipe。

lookup \\在 cache 中查找请求目标，控制权最终会传递给 vcl_hit 或者 vcl_miss，取决于目标是否在 cache 中。

vcl_pipe

请求进入 pipe 模式的时候被调用，在这个模式，请求会被 passed 到后端服务器，在连接关闭前，无论是这个客户端还是对应的后端服务器的数据，都会进入 pass 模式。

vcl_pipe 子程序以下面的关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pipe \\继续进入 pipe 模式。

vcl_pass

请求进入 pass 模式的时候被调用，在这个模式，请求会被 passed 到后端服务器，

后端服务器的应答会被 passed 给客户端，但是不会被缓存。相同客户端的随后的请求正常处理。

vcl_pass 子程序以下面的关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pass \\继续进入 pass 模式。

vcl_hash

使用 req.hash += req.http.Cookie 或者 HTTP 头文件包含的 cookie 生成 hash 字符串。

vcl_hash 将以下面的关键字结束：

hash\\继续进入 hash 模式

vcl_hit

当一个请求从 cache 中命中需要的内容，vcl_hit 子程序以下面关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pass \\继续进入 pass 模式，控制权转交 vcl_pass 子程序。

deliver \\提交命中的目标给客户端，控制权转交 vcl_deliver 子程序。

vcl_miss

当需要的内容没有在缓存中命中的时候被调用，决定是否尝试到后端服务器查找目标，从哪个后端服务器查找目标？

vcl_miss 子程序以下面的关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pass \\进入 pass 模式，控制权转交给 vcl_pass

fetch \\从后端服务器获得请求目标，控制权转交给 vcl_fetch。

vcl_fetch

目标成功从后端服务器中获取的时候被调用

vcl_fetch 子程序以下面的关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pass \\进入 pass 模式，控制权转交给 vcl_pass

deliver \\可能把找到的目标插入缓存中，然后发送给客户端，控制权转交给 vcl_deliver

esi \\ ESI-process the document which has just been fetched

vcl_deliver

当一个没有被 cached 内容交付给客户端的时候被调用

vcl_deliver 子程序以下面关键字结束：

error code [reason] \\返回规定的代码给客户端，并终止请求。

pass \\进入 pass 模式，控制权转交给 vcl_pass

deliver \\交付目标给客户端

如果这些子程序没有被定义，或者没有完成预定的处理而被终止，控制权将被转交给系统默认的子程序。查看 EXAMPLES 章节查看默认的代码。

Multiple subroutines（更多的子程序）

如果多个子程序被定义成相同的名字，they are concatenated in the order in which the appear in the source。

示例：

```
# in file "main.vcl"
include "backends.vcl";
```

```
include "purge.vcl";

# in file "backends.vcl"
sub vcl_recv {
   if (req.http.host ~ "example.com") {
      set req.backend = foo;
   } elsif (req.http.host ~ "example.org") {
      set req.backend = bar;
   }
}

# in file "purge.vcl"
sub vcl_recv {
   if (client.ip ~ admin_network) {
      if (req.http.Cache-Control ~ "no-cache") {
         purge_url(req.url);
      }
   }
}
```

内置的子程序会暗中的附加在里面。

Variables（变量）

虽然子程序没有参数，子进程必须的信息通过全局变量来处理。

以下变量是可用的变量：

now \\当前时间

下面的变量在 backend 申明中有效：

.host \\一个 backend 的主机名或者 IP 地址

.port \\一个 backend 的服务名字或者端口号

下面的变量在处理请求时有效：

client.ip \\客户端 IP

server.hostname \\server 的主机名

server.identity \\server 的身份，使用-i 参数设置，如果 –i 参数没有传递给 varnishd，server.identity 将给 varnishd 实例设置名字。设置详细的信息使用 –n 参数。

server.ip \\客户端连接上 socket，接收到的 IP 地址

server.port \\客户端连接上 socket，接收到的端口号

req.request \\请求类型，例如”GET”,”HEAD”

req.url \\请求的 URL

req.proto \\客户端使用的 HTTP 的协议版本

req.backend \\使用哪个后端服务器为这个请求提供服务

req.backend.healthy \\后端服务器是否健康

req.http.header \\对应的 HTTP 头

bereq.connect_timeout \\等待后端服务器响应的时间

bereq.first_byte_timeout \\等待接收第一个字节的等待时间，pipe 模式中无

效。

bereq.between_bytes_timeout \\短时间内，两次从后端服务器接收到字节的间隔，pipe 模式无效。

下面这些变量在请求目标被成功的从后端服务器或者缓存中获得后有效

obj.proto \\返回请求目标的 HTTP 版本

obj.status \\服务器返回的 HTTP 状态码

obj.response \\服务器返回的 HTTP 状态信息

obj.cacheable \\如果返回的结果是可以缓存的，而且 HTTP 状态码必须是 200, 03, 300, 301, 302, 404 和 410.

obj.ttl \\目标的剩余生存时间，以秒为单位。

obj.lastuse \\最后一个请求后，过去的时间，以秒为单位。

obj.hits \\大概的 delivered 的次数，如果为 0，表明缓存出错。

下面这些变量在目标 hash key 以后有效

req.hash \\hash key 和缓存中的目标相关，在读出和写入缓存时使用。

下面这些变量在准备回应客户端时使用

resp.proto \\准备响应的 HTTP 协议版本

resp.status \\返回客户端的 HTTP 状态码

resp.response \\返回客户端的 HTTP 状态信息

resp.http.header \\通信的 HTTP 头

使用 SET 关键字，把值分配给变量：

```
sub vcl_recv {
  # Normalize the Host: header
  if (req.http.host ~ "^(www.)?example.com$") {
     set req.http.host = "www.example.com";
     }
 }
```

可以使用 remove 关键字把 HTTP 头彻底的删除：

```
sub vcl_fetch {
  # Don't cache cookies
  remove obj.http.Set-Cookie;
}
```

EXAMPLES（例子）

下面这段代码和默认的配置相同，后端服务器主机名设置为“backend.exampl.com”

```
backend default {
 .host = "backend.example.com";
 .port = "http";
}

sub vcl_recv {
  if (req.http.x-forwarded-for) {
```

```
    set req.http.X-Forwarded-For = req.http.X-Forwarded-For ", " client.ip;
} else {
    set req.http.X-Forwarded-For = client.ip;
}

if (req.request != "GET" &&
req.request != "HEAD" &&
req.request != "PUT" &&
req.request != "POST" &&
req.request != "TRACE" &&
req.request != "OPTIONS" &&
req.request != "DELETE") {
    // Non-RFC2616 or CONNECT which is weird.
    return (pipe);
    }
if (req.request != "GET" && req.request != "HEAD") {
// We only deal with GET and HEAD by default
    return (pass);
    }
if (req.http.Authorization || req.http.Cookie) {
    // Not cacheable by default
    return (pass);
    }
return (lookup);
}

sub vcl_pipe {
    # Note that only the first request to the backend will have
    # X-Forwarded-For set.    If you use X-Forwarded-For and want to
    # have it set for all requests, make sure to have:
    # set req.http.connection = "close";
    # here.    It is not set by default as it might break some broken web
    # applications, like IIS with NTLM authentication.
    return (pipe);
}

sub vcl_pass {
    return (pass);
}

sub vcl_hash {
    set req.hash += req.url;
    if (req.http.host) {
        set req.hash += req.http.host;
```

```
    } else {
        set req.hash += server.ip;
    }
    return (hash);
}

sub vcl_hit {
    if (!obj.cacheable) {
    return (pass);
    }
return (deliver);
}

sub vcl_miss {
    return (fetch);
}

sub vcl_fetch {
if (!beresp.cacheable) {
    return (pass);
}
if (beresp.http.Set-Cookie) {
    return (pass);
    }
return (deliver);
}

sub vcl_deliver {
    return (deliver);
}

sub vcl_error {
set obj.http.Content-Type = "text/html; charset=utf-8";
synthetic {"
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>"} obj.status " " obj.response {"</title>
</head>
<body>
<h1>Error "} obj.status " " obj.response {"</h1>
<p>"} obj.response {"</p>
```

```
<h3>Guru Meditation:</h3>
<p>XID: "} req.xid {"</p>
<hr>
Varnish cache server
</body>
</html>
"};
return (deliver);
}
```

下面的例子显示一个 varnishd 实例支持多个独立的站点，基于请求的 URL 选择使用的后端服务器：

```
backend www {
    .host = "www.example.com";
    .port = "80";
}

backend images {
    .host = "images.example.com";
    .port = "80";
}

sub vcl_recv {
    if (req.http.host ~ "^(www.)?example.com$") {
        set req.http.host = "www.example.com";
        set req.backend = www;
    } elsif (req.http.host ~ "^images.example.com$") {
        set req.backend = images;
    } else {
        error 404 "Unknown virtual host";
    }
}

The following snippet demonstrates how to force a minimum TTL for
all documents.    Note that this is not the same as setting the
default_ttl run-time parameter, as that only affects document for
which the backend did not specify a TTL:::

sub vcl_fetch {
    if (obj.ttl < 120s) {
    set obj.ttl = 120s;
    }
}
```

下面这段代码用来强制缓存带 cookies 的内容：

```
sub vcl_recv {
```

```
    if (req.request == "GET" && req.http.cookie) {
    call(lookup);
    }
}

sub vcl_fetch {
    if (beresp.http.Set-Cookie) {
    deliver;
  }
}
```

下面代码的作用是利用 squid 的 HTTP PURGE 模式清理无法使用的目标。

```
acl purge {
    "localhost";
    "192.0.2.1"/24;
}

sub vcl_recv {
    if (req.request == "PURGE") {
    if (!client.ip ~ purge) {
        error 405 "Not allowed.";
    }
    lookup;
    }
}

sub vcl_hit {
    if (req.request == "PURGE") {
        set obj.ttl = 0s;
        error 200 "Purged.";
    }
}

sub vcl_miss {
    if (req.request == "PURGE") {
    error 404 "Not in cache.";
    }
}
```

SEE ALSO

Varnishd（1）

HISTORY

The VCL language was developed by Poul-Henning Kamp in cooperation with Verdens Gang AS, Linpro AS and Varnish Software. This manual page was written by Dag-Erling Smørgrav and later edited by Poul-Henning Kamp and Per Buer.

COPYRIGHT

这个文档的版权和 varnish 自身的版本一样，请看 LICENCE。

●varnishadm

Control a running varnish instance(控制一个运行的 varnish 实例)

Author: Cecilie Fritzvold

Author: Per Buer

Date: 2010-05-31

Version: 0.2

Manual section: 1

SYNOPSIS（大纲）

Varnishadm [-t timeout] [-S secret_file] –T address:port [command[…]]

DESCRIPTION(描述)

Varnishadm 实用工具建立一个 CLI 的连接。使用 –T 和 –S 参数。

如果给出了要执行的命令，varnishadm 会传输命令和返回运行的结果到标准输出。

如果没有给出要执行的命令，varnish 会给你一个 CLI 接口，可以在 CLI 接口上输入命令和返回结果。

OPTIONS

-t timeout \\等待一个操作完成的时间 单位秒

-S secret_file \\确定一个认证的安全文件

-T address：port \\连接到管理接口的地址和端口

有效的命令和参数被记录在 varnishd（1）手册里。一些大概的命令可以通过 help 命令获得。一些简要的参数可以通过 param.show 命令获得。

EXIT STATUS

如果给予一个命令，如果它执行成功，退出状态为 0，如果不成功，它的退出状态非 0.

EXAMPLES

一些执行 varnishadm 的方式：

```
varnishadm -T localhost:999 -S /var/db/secret vcl.use foo
echo vcl.use foo | varnishadm -T localhost:999 -S /var/db/secret
echo vcl.use foo | ssh vhost varnishadm -T localhost:999 -S /var/db/secret
```

SEE ALSO（参见）

varnishd（1）

HISTORY（历史）

Varnishadm 工具和这个手册都是 Cecilie Fritzvold 编写的。由 per buer 在 2010 年对程序和手册进行了改写和升级。

COPYRIGHT（版权）

这个文档的版权和 varnish 自己的版权一样，请参阅 LINENCE 文档了解细节。

●varnishd

HTTP accelerator daemon（HTTP 加速进程）

Author: Dag-Erling Smørgrav
Author: Stig Sandbeck Mathisen
Author: Per Buer
Date: 2010-05-31
Version: 1.0
Manual section: 1

SYNOPSIS（大纲）

varnishd [-a address[:port]] [-b host[:port]] [-d] [-F] [-f config] [-g group] [-h type[,options]] [-i identity] [-l shmlogsize] [-n name] [-P file] [-p param=value] [-s type[,options]] [-T address[:port]] [-t ttl] [-u user] [-V] [-w min[,max[,timeout]]]

DESCRIPTION（描述）

Varnishd 进程接收客户端的 HTTP 请求，然后把请求发送给后端服务器，缓存后端服务器返回的内容，这样更好的满足以后相同的请求。

OPTIONS（选项）

-a address：[:port][,address]

监听指定的 IP 地址和端口的请求。地址可以是主机名（“localhost”），或者一个 IPV4（“127.0.0.1”），和 IPV6（“[::1]”）,如果地址没有明确指定，varnishd 将监听所有可用的 IPV4 和 IPV6 地址。如果端口没有指定，那么 varnishd 默认监听/etc/services 中的 HTTP 对应的端口。更多的端口和地址使用逗号分隔。

-b host[:port]

指定后端服务器的地址和端口，如果没有指定端口，默认的是 8080.

-C 编译 VCL 代码成 C 语言，然后退出，指定 VCL 文件用-f 参数。

-d 开启 debug 模式。主进程在前段启动，提供一个 CLI 界面用于标准输入和输出，子进程必须通过 CLI 命令启动。如果结束主进程，那么子进程也会结束。

-F 在前端运行。

-f config 使用指定的 VCL 配置文件代替系统默认的。参见 VCL（7）

-g group 指定 varnishd 子进程使用的用户组。

-h type[,options] 指定 hash 算法。

-i identity 指定 varnishd server 的身份。

-l shmlogsize 指定 shmlogfile 的大小，单位可以使用‘k’和‘m’，默认是 80M，如果指定比 8M 小，那是不明智的。

-n name 为这个实例指定一个名字。

-P file 指定 pidfile，用于保存 PID 信息。

-p param=value 设定指定参数的值，查看运行时的阐述列表

-S file 访问管理端口用到的安全认证文件的路径

-s type[,options] 使用指定的存储后端，查看存储列表获得支持的存储类型。可以多次使用此选项指定不同的存储类型。

-T address[:port]

提供一个管理接口的地址和端口，通过管理接口查看管理命令。

-t ttl 指定最小的 TTL 给 cache 中的内容。这是一个捷径设置 default_ttl run-time 选项。

-u user 指定运行 varnishd 子进程的用户。

-V 显示 varnishd 的版本，然后退出。

-w min[,max[,timeout]]

指定线程最小和最大空闲时间。这是一个设置 thread_pool_min 和 thread_pool_max、thread_pool_timeout 的捷径。

如果只有一个值被指定，那么 thread_pool_min 和 thread_pool_max 都是用这个值。Thread_poll_timeout 会失效。

Hash Algorithms（hash 算法）

以下 hash 算法是可用的：

Simple_list 一个简单的 doubly-linked 列表，不推荐生产环境应用。

Classic[,buckets] 一个标准的 hash 表，默认使用这个。

Critbit xxx very nice。

Storage Types（存储类型）

下面的存储类型是可用的：

malloc[,size]

存储目标分配给 malloc（3）。

Size 参数指定最大分配给 varnishd 的内存，size 默认的单位是 bytes。除非你指定以下单位

K \\kibibytes

M \\membibytes

G \\gibibytes

T \\tebibytes

默认没有限制

File[,path[,size[,granularity]]]

把存储目标分配给一个文件，这个是默认的。

Path 参数指定文件名和路径，或者 path 指定一个目录，varnishd 会自己创建文件。默认在/tmp 目录。

Size 参数指定最大分配给 varnishd 的内存，size 默认的单位是 bytes。除非你指定以下单位

K \\kibibytes

M \\membibytes

G \\gibibytes

T \\tebibytes

%使用所属文件系统空闲空间的百分比。

默认是 50%

如果文件已经存在，那么 varnishd 会缩减或者扩大 backing 文件到指定 size。

注意：如果 varnishd 需要创建或者扩大一个文件，之前没有设置好空间的话，可能会产生碎片，这样会影响性能。在分配文件之前使用 dd 命令来创建文件，这样可以尽量减少碎片。

Granularity 参数指定分配的间隔尺寸，默认的单位是字节，你可以自定义单位，除了使用%。

默认的间隔尺寸就是虚拟内存相同。如果有太多的小文件，那么空间就会减少。

Management Interface（管理接口）

如果指定了-T 选项，varnishd 将提供一个命令行的管理接口在指定的地址和接口。下面的命令是可以用的：

help [command]

显示可用命令列表

param.set param value

给参数设定值，参见参数列表

param.show [-l] [param]

显示运行时参数和值的列表

如果使用-l 选项，列表会对每个参数做一些描述。

如果 param 被指定，那么只显示指定的 param。

ping [timestamp]

Ping varnishd 的进程，查看连接是否存活。

purge field operator argument [&& field operator argument [...]]

匹配 purge 表达式的内容，马上失效。

purge.list

显示 purge 列表

所有在 cache 中的请求目标和 purge.list 相比，如果 cache 中的目标比匹配的 purge list 旧，就会考虑清楚这些旧的目标，从后端服务器获取新的代替。

如果一个清洗表达式比 cache 中的目标就，那么就会从 purge list 中删除它们。

purge.url regexp

使缓存中匹配表达式的 URL 马上失效。

quit

关闭 varnish admin 的连接。

start

如果 varnish cache 程序没有运行，在这里可以启动这个进程。

stats

显示简要的统计信息。

所有的数据是服务器启动后开始统计，更好的观察方式是使用 varnishstat 工具。

status

检查 varnish cache 进程的状态。

stop

停止 varnish cache 进程

url.purge regexp

弃用，使用 purge.url 代替。

vcl.discard configname

丢弃指定的 VCL 配置文件。

vcl.inline　　configname vcl

创建一个 vcl 配置文件，configname 是配置文件的名字，vcl 是配置文件的 VCL 代码。

vcl.list

显示当前可用的 vcl 列表，active 表示当前使用的配置文件。

vcl.load　configname filename

创建一个新的配置文件，configname 是配置的名字，filename 是 vcl 配置文件的路径和名字。

vcl.show　config

显示 vcl 配置文件的源代码。

vcl.use　　configname

对所有新的请求指定 vcl 配置文件，已经存在的请求使用抵达时的配置文件。

Run-Time Parameters(运行时的参数)

运行参数有速记标志，避免重复出现相同的文件，该标志的含义：

experimental

对这个参数，我们没有固定的值来说明好不好，欢迎观察和反馈这个值。

delayed

这个值可以在不工作的时候改变，但是不会立即生效。

restart

工作进程会被停止，并且重新启动。

reload

VCL 程序会被重新装载。

下面是参数列表，目前是我们最后一次更新指南。下面的列表和 CLI 管理接口的 param.show 命令产生的列表相同，如果某些描述在这里找不到，你可以在 CLI 使用命令找到相关描述。

下面是在一个 32 位的系统上的默认值，比如 sess_workspace（=16K）和 thread_pool_stack（=64K），可以减少这些值保持虚拟内存空间。

acceptor_sleep_decay

Default：0.900 Flags：experimental

如果我们的文件描述，或者工作线程等资源耗尽，接收器在接收两次接收间隔中会休眠，这个参数减少成功接收的休眠时间。（ie：0.9 = reduce by 10%）

acceptor_sleep_incr

Units：s default：0.001 flags：experimental

如果我们的文件描述，或者工作线程等资源耗尽，接收器在两次接收间隔中会休眠，这个参数控制休眠的时间。

acceptor_sleep_max

Units:s default:0.050 flags:xperimental

如果我们的文件描述，或者工作线程等资源耗尽，接收器在接收两次接收间隔中会休眠，这个参数娴熟最长的休眠时间。

auto_restart

Units：bool default：on

如果子进程宕了，自动重启。

ban_lurker_sleep

Units：s default：0.0

How long time does the ban lurker thread sleeps between successfull attempts to push the last item up the purge list. It always sleeps a second when nothing can be done. A value of zero disables the ban lurker.

between_bytes_timeout

Units：s Default：60

在接收数据是，两个字节之间的超时时间，如果值是 0，就表示用不超时，VCL 配置文件的值可以覆盖这里的值，这个参数在 pipe 模式中无效。

cache_vbe_conns

Units：bool Default：off Flags：experimental

Cache vb_conn 依耐于 malloc，这是一个问题。

cc_command

Default：exec cc –fpic –shared –Wl，–x –o %o %s Flags：must_reload

编译 c 源代码的参数，%s 将替换源文件名字，,%o 将替换输出的文件名字。

cli_buffer

Units：bytes Default：8192

CLI 输入的缓冲区大小，如果你有一个很大 VCL 文件或者使用 vcl.inline 命令，那么就需要增大这个值，注意必须使用-p，让所有的生效。

cli_timeout

Units：seconds Default：10

管理员对 CLI 的请求超时时间。

clock_skew

Units：s Default：10

在后端服务器和 varnish 之间，多少时差愿意接受。

connect_timeout

Units：s Default：0.4

连接后端服务器默认的连接超时时间，VCL 的配置可以覆盖这个选项。

default_grace

Default:10Seconds Flags:delayed

定义 grace 宽度，我们会递交一个之前的内容，在其他线程想要新的拷贝。

default_ttl

Units：seconds Default：120

如果 backend 和 VCL 都没有给目标分配 TTL，那么这个参数将会生效，已经存在于缓存中的目标在他们从新从后端获取前不受影响，强制他们生效可以使用“purge.url”刷新 cache。

diag_bitmap

Units：bitmap default：0 bitmap controlling diagnostics code：

0x00000001 - CNT_Session states.

0x00000002 - workspace debugging.

0x00000004 - kqueue debugging.

0x00000008 - mutex logging.

0x00000010 - mutex contests.

0x00000020 - waiting list.
0x00000040 - object workspace.
0x00001000 - do not core-dump child process.
0x00002000 - only short panic message.
0x00004000 - panic to stderr.
0x00008000 - panic to abort2().
0x00010000 - synchronize shmlog.
0x00020000 - synchronous start of persistence.
0x80000000 - do edge-detection on digest.

err_ttl

Units： seconds Default： 0

TTl 分配给错误页面。

Esi_syntax

Units： bitmap Default： 0 bitmap controlling ESI parsing code：

0x00000001 - Don't check if it looks like XML

0x00000002 - Ignore non-esi elements

0x00000004 - Emit parsing debug records

fetch_chunksize

Units： kilobytes Default： 128flags： experimental

Fetcher 使用默认的 chunksize，这个值应该比多数目标大，而且 TTLS 更短，

first_byte_timeout

Units: s default:60

定义从后端服务器收到第一个数据的超时时间，我们只等待这个时间，如果超过，就放弃。如果值是 0，那么久永不放弃。VCL 的配置可以覆盖这个值，此值在 pipe 模式中无效。

group

Default： …. Flags： must_restart

使用哪个没有特权的组运行此进程。

http_headers

Units： header lines default： 64

可以处理的最大数目的 HTTP 头

listen_address

Default： 80 flags： must_restart

可以使使用的表达方式 host，host：port，port

listen_depth

Units： connections default： 1024 flags： must_restart

监听队列深度。

log_hashstring

Units： bool default： off

日志是否记录共享内存里的 hash 字符

log_local_address

Units： bool default： off

是否记录本地 IP 的 tcp 连接。

lru_interval

Units： seconds default： 2 flags:experimental

在目标从 LRU 表移除的前的宽限时间

max_esi_includes

Units： includes default： 5

最大数量的 Esi 进程。

max_restarts

Units： restarts default： 4

一个请求重试次数的最大限制，应该意识到重试次数会影响命中率，所以不要轻易改变这个值。

overflow_max

Units： % default： 100 flags： experimental

允许溢出队列长度的百分比。

这个设置排队请求工作的线程，超过上面的值就会丢弃。

ping_interval

Units： seconds default： 3 flags： must_restart

子进程 ping 主进程的间隔。0 就是禁止 ping。

pipe_timeout

Units： seconds default： 60

PIPE 会话的空闲超时时间，如果他们之间在此值期间没有数据收发，那么就会关闭 session。

prefer_ipv6

Units： bool default： off

当后端服务器支持 IPV4 和 IPV6，在连接后端服务器的时候更喜欢 IPv6 地址。

purge_dups

Units： bool default： on

发现和消除重复的 purges。

rush_exponent

Units： requests per request default： 3 flags： experimental

How many parked request we start for each completed request on the object. NB: Even with the implict delay of delivery, this parameter controls an exponential increase in number of worker threads.

Purge expressions

一个 purge 表达包含一个或者多个条件，一个条件包含一个字段，一个运算符，一个内容条件可以使用“&&”符号连接起来。

一个字段可以包含任何 VCL 的变量，例如 req.url，req.http.host 或者 obj.set-cookie。

运算符包括“==”直接比较，“~”正则表达式匹配，和“>”“<”用于 size 比较。先考虑运算符是否“!”，表示否者表达式。

内容需要引用字符串，正则表达式，整数。整数可以包含“KB,MB,GB,TB”相关的单位

例子：所有请求的 req.url 匹配/news，就 purge cache。

```
req.url == "/news"
```

例子：purge 所有名字不是”.ogg”结尾的，并且大小超过 10MB 的文件。

```
req.url !~ "\.ogg$" && obj.size > 10MB
```

例子：purge 所有提供服务主机名字为“example.com”和 www.example.com，以及所有从后端服务器中收到的头信息 Set-Cookie 的值为“USERID=1663”的文件：

```
req.http.host ~ "^(www\.)example.com$" && obj.set-cookie ~ "USERID=1663"
```

SEE ALSO

* varnishlog(1)
* varnishhist(1)
* varnishncsa(1)
* varnishstat(1)
* varnishtop(1)
* vcl(7)

HISTORY

The varnishd daemon was developed by Poul-Henning Kamp in cooperation with Verdens Gang AS, Linpro AS and Varnish Software.

This manual page was written by Dag-Erling Smørgrav with updates by Stig Sandbeck Mathisen ⟨ssm@debian.org⟩

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

●varnishhist（varnish 请求图）

Author: Dag-Erling Smørgrav
Date: 2010-05-31
Version: 1.0
Manual section: 1

Varnish request histogram（varnish 请求柱状图）

SYNOPSIS

varnishhist [-b] [-C] [-c] [-d] [-I regex] [-i tag] [-n varnish_name] [-r file] [-V] [-w delay] [-X regex] [-x tag]

DESCRIPTION

Varnishhist 工具读取 varnishd（1）的共享内存日志，生成一个连续不断更新的柱状图显示最后 N 个请求的分布。N 的值取决于左上角垂直刻度的高度。水平标尺是对数。如果命中，则使用“|”表示，如果没有命中，使用“#”表示。

下面的选项是可用的：

-b 分析指定后端服务器的日志，如果没有使用-b 和-c 参数，varnish 就充当他们。

-C 忽略正则表达式的大小写。

-c 分析指定客户端的日志。

-d 在启动过程中处理旧的日志，一般情况下，varnishhist 只会在进程写入日志后启动。

-I regex 匹配正则表达式的日志，如果没有使用-i 或者-I，那么所有的日志都

会匹配。

-i tag 匹配指定的 tag，如果没有使用-i 或者-I，那么所有的日志都会被匹配。

-n 指定 varnish 实例的名字，用来获取日志，如果没有指定，默认使用主机名。

-r file 读入日志文件，代替共享内存。

-V 显示版本号，然后退出。

-w delay 等待更新的延迟时间，默认是 1 秒。

-X regex 导入匹配表达式的日志。

-x tag 导入匹配 tag 的日志。

SEE ALSO

* varnishd(1)
* varnishlog(1)
* varnishncsa(1)
* varnishstat(1)
* varnishtop(1)

HISTORY

The varnishhist utility was developed by Poul-Henning Kamp in cooperation with Verdens Gang AS and Linpro AS. This manual page was written by Dag-Erling Smørgrav.

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

●varnishlog（varnish 日志）

Author: Dag-Erling Smørgrav
Author: Per Buer
Date: 2010-05-31
Version: 0.2
Manual section: 1

Display varnish logs

SYNOPSIS

varnishlog [-a] [-b] [-C] [-c] [-D] [-d] [-I regex] [-i tag] [-k keep] [-n varnish_name] [-o] [-P file] [-r file] [-s num] [-u] [-V] [-w file] [-X regex] [-x tag] [tag regex]

DESCRIPTION

varnishlog 工具读取和显示共享内存的日志。

下面的选项是可用的：

-a 当把日志写到文件里时，使用附加，而不是覆盖。

-b 只显示 varnishd 和后端服务器的日志。

-C 匹配正则表达式的时候，忽略大小写差异。

-c 只显示 varnishd 和客户端的日志。

-D 以进程方式运行

-d 在启动过程中处理旧的日志，一般情况下，varnishhist 只会在进程写入日志后启动。

-I regex 匹配正则表达式的日志，如果没有使用-i 或者-I，那么所有的日志都会匹配。

-i tag 匹配指定的 tag，如果没有使用-i 或者-I，那么所有的日志都会被匹配。

-k num 只显示开始的 num 个日志记录。

-n 指定 varnish 实例的名字，用来获取日志，如果没有指定，默认使用主机名。

-o 以请求 ID 给日志分组，这个功能没多大用。如果要写到一个文件里使用-w 选项。

-P file 记录 PID 号的文件

-r file 从一个文件读取日志，而不是从共享内存读取。

-s sum 跳过开始的 num 条日志。

-u 无缓冲的输出。

-V 显示版本，然后退出。

-w file 把日志写到一个文件里代替显示他们，如果不是用-a 参数就会发生覆盖，如果 varnishlog 在写日志时，接收到一个 SIGHUP 信号，他会创建一个新的文件，老的文件可以移走。

-X regex 排除匹配正则表达式的日志。

-x tag 排除匹配 tag 的日志。

如果-o 选项被指定，需要使用正则表达式和 tag 来制定需要的日志。

TAGS

下面的日志 tag 是正确定义的：

* Backend
* BackendClose
* BackendOpen
* BackendReuse
* BackendXID
* CLI
* ClientAddr
* Debug
* Error
* ExpBan
* ExpKill
* ExpPick
* Hit
* HitPass
* HttpError
* HttpGarbage
* Length
* ObjHeader
* ObjLostHeader
* ObjProtocol

* ObjRequest
* ObjResponse
* ObjStatus
* ObjURL
* ReqEnd
* ReqStart
* RxHeader
* RxLostHeader
* RxProtocol
* RxRequest
* RxResponse
* RxStatus
* RxURL
* SessionClose
* SessionOpen
* StatAddr
* StatSess
* TTL
* TxHeader
* TxLostHeader
* TxProtocol
* TxRequest
* TxResponse
* TxStatus
* TxURL
* VCL_acl
* VCL_call
* VCL_return
* VCL_trace
* WorkThread

EXAMPLES

下面的命令简单的打印日志到一个文件：

```
$ varnishlog -w /var/log/varnish.log
```

下面这条命令读取一个日志文件，然后显示请求的首页：

```
$ varnishlog -r /var/log/varnish.log -c -o RxURL '^/$'
```

SEE ALSO

* varnishd(1)
* varnishhist(1)
* varnishncsa(1)
* varnishstat(1)
* varnishtop(1)

HISTORY

The varnishlog utility was developed by Poul-Henning Kamp ⟨phk@phk.freebsd.dk⟩ in cooperation with Verdens Gang AS, Linpro AS and Varnish Software. This manual page was

initially written by Dag-Erling Smørgrav.

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

* Copyright (c) 2006 Verdens Gang AS
* Copyright (c) 2006-2008 Linpro AS
* Copyright (c) 2008-2010 Redpill Linpro AS
* Copyright (c) 2010 Varnish Software AS

●varnishncsa（以 NCSA 的格式显示日志）

Author: Dag-Erling Smørgrav
Date: 2010-05-31
Version: 1.0
Manual section: 1

Display varnish logs in apache/NCSA combined log format

SYNOPSIS

varnishncsa [-a] [-b] [-C] [-c] [-D] [-d] [-f] [-I regex] [-i tag] [-n varnish_name] [-P file] [-r file] [-V] [-w file] [-X regex] [-x tag]

DESCRIPTION

Varnishncsa 工具读取共享内存的日志，然后以 apache/NCSA 的格式显示出来。下面的选项可以用。

-a 当把日志写到文件里时，使用附加，而不是覆盖。

-b 只显示 varnishd 和后端服务器的日志。

-C 匹配正则表达式的时候，忽略大小写差异。

-c 只显示 varnishd 和客户端的日志。

-D 以进程方式运行

-d 在启动过程中处理旧的日志，一般情况下，varnishhist 只会在进程写入日志后启动。

-f 在日志输出中使用 X-Forwarded-For HTTP 头代替 client.ip。

-I regex 匹配正则表达式的日志，如果没有使用-i 或者-I，那么所有的日志都会匹配。

-i tag 匹配指定的 tag，如果没有使用-i 或者-I，那么所有的日志都会被匹配。

-n 指定 varnish 实例的名字，用来获取日志，如果没有指定，默认使用主机名。

-P file 记录 PID 号的文件

-r file 从一个文件读取日志，而不是从共享内存读取。

-w file 把日志写到一个文件里代替显示他们，如果不是用-a 参数就会发生覆盖，如果 varnishlog 在写日志时，接收到一个 SIGHUP 信号，他会创建一个新的文件，老的文件可以移走。

-X regex 排除匹配正则表达式的日志。

-x tag 排除匹配 tag 的日志。

SEE ALSO

* varnishd(1)

* varnishhist(1)
* varnishlog(1)
* varnishstat(1)
* varnishtop(1)

HISTORY

The varnishncsa utility was developed by Poul-Henning Kamp in cooperation with Verdens Gang AS and Linpro AS. This manual page was written by Dag-Erling Smørgrav ⟨des@des.no⟩.

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

●varnishreplay

HTTP traffic replay tool

Author: Cecilie Fritzvold
Author: Per Buer
Date: 2010-05-31
Version: 1.0
Manual section: 1

SYNOPSIS

Varnishreplay [-D] -a address:port –r file

DESCRIPTION

Varnishreplay 工具类似 varnish logs 尝试将流量复制。下面的参数是可用的：

-a backend 发送到这台服务器的 TCP 流量，指定一个地址和端口，这个选项只能被 IPV4 上支持

-D 打开 debug 模式。

-r file 使用文件里的语法分析日志，这个参数是强制的。

SEE ALSO

* varnishd(1)
* varnishlog(1)

HISTORY

The varnishreplay utility and this manual page were written by Cecilie Fritzvold and later updated by Per Buer.

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

●varnishsizes（请求 SIZE 图）

Varnish object size request histogram

Author: Dag Erling Smørgrav

Author: Kristian Lyngstøl

Author: Per Buer

Date: 2010-05-31

Version: 1.0

Manual section: 1

SYNOPSIS

varnishsizes [-b] [-C] [-c] [-d] [-I regex] [-i tag] [-n varnish_name][-r file] [-V] [-w delay] [-X regex] [-x tag]

DESCRIPTION

Varnishsizes 工具读取 varnishd（1）的共享内存日志，生成一个连续不断更新的柱状图显示最后 N 个请求的分布。N 的值取决于左上角垂直刻度的高度。水平标尺是对数。如果命中，则使用“|”表示，如果没有命中，使用“#”表示。

下面的选项是可用的：

-b 分析指定后端服务器的日志，如果没有使用-b 和-c 参数，varnish 就充当他们。

-C 忽略正则表达式的大小写。

-c 分析指定客户端的日志。

-d 在启动过程中处理旧的日志，一般情况下，varnishhist 只会在进程写入日志后启动。

-I regex 匹配正则表达式的日志，如果没有使用-i 或者-I，那么所有的日志都会匹配。

-i tag 匹配指定的 tag，如果没有使用-i 或者-I，那么所有的日志都会被匹配。

-n 指定 varnish 实例的名字，用来获取日志，如果没有指定，默认使用主机名。

-r file 读入日志文件，代替共享内存。

-V 显示版本号，然后退出。

-w delay 等待更新的延迟时间，默认是 1 秒。

-X regex 排除匹配表达式的日志。

-x tag 排除匹配 tag 的日志。

SEE ALSO

* varnishd(1)
* varnishlog(1)
* varnishncsa(1)
* varnishstat(1)
* varnishtop(1)

HISTORY

The varnishsizes utility was developed by Kristian Lyngstøl based on varnishhist. This manual page was written by Kristian Lyngstøl, Dag-Erling Smørgrav and Per Buer.

COPYRIGHT

●varnishstat

HTTP accelerator statistics

Author: Dag-Erling Smørgrav

Author: Per Buer

Date: 2010-06-1

Version: 1.0

Manual section: 1

SYNOPSIS

varnishstat [-1] [-f field_list] [-l] [-n varnish_name] [-V] [-w delay]

DESCRIPTION

Varnishstat 工具显示一个运行 varnishd 实例的相关统计数据。

下面的参数可以用：

-1 只显示一次就退出。

-f 使用逗号分隔字段列表来显示，使用“^”开始排除列表。

-l 监听有效的列使用-f 参数。

-n 指定 varnishd 实例来读取日志，如果没有指定，则默认使用主机名。

-V 显示版本号，然后退出。

-w delay 刷新间隔时间，默认 1s。

中心显示中每列的含义，从左到右：

1、 值

2、 从最后一秒更新以来的每秒的一个平均值，或者一个不能计算的周期

3、 从进程开始到现在每秒的平均值，或者是一个不能计算的周期。

4、 描述

当使用-1 选项，输出列的含义,从左到右：

1、 特征名字

2、 值

3、 从进程开始到现在每秒的平均值，或者是一个不能计算的周期。

4、 描述

SEE ALSO

* varnishd(1)
* varnishhist(1)
* varnishlog(1)
* varnishncsa(1)
* varnishtop(1)
* curses(3)

HISTORY

The varnishstat utility was originally developed by Poul-Henning Kamp

⟨phk@phk.freebsd.dk⟩ in cooperation with Verdens Gang AS, Linpro AS and Varnish Software. Manual page written by Dag-Erling Smørgrav, and Per Buer.

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

* Copyright (c) 2006 Verdens Gang AS
* Copyright (c) 2006-2008 Linpro AS
* Copyright (c) 2008-2010 Redpill Linpro AS
* Copyright (c) 2010 Varnish Software AS

●varnishtest

Test program for varnish

Author: Stig Sandbeck Mathisen

Date: 2010-05-31

Version: 1.0

Manual section: 1

SYNOPSIS（大纲）

varnishtest [-n iter] [-q] [-v] file [file ...]

DESCRIPTION

Varnishtest 程序是一个脚本驱动程序，用来测试 varnishd

由于这个命令很少用，所以暂时不翻译，你可以使用 man varnishtest 查看相关文档。

●varnishtop

Varnish log entry ranking

Author: Dag-Erling Smørgrav

Date: 2010-05-31

Version: 1.0

Manual section: 1

SYNOPSIS

varnishtop [-1] [-b] [-C] [-c] [-d] [-f] [-I regex] [-i tag] [-n varnish_name] [-r file] [-V] [-X regex] [-x tag]

DESCRIPTION

Varnishtop 工具读取 varnishd（1）共享内存的日志，连续不断的更新和显示日志记录。使用 –I,-i，-X 和-x 选项适当的过滤，他可以显示一个排序关于请求的内容，客户端，用户代理，或者其他记录在日志里的信息。

下面的参数可以使用：

-1 代替连续不断的更新和显示，只显示一次然后退出。暗示：-d

-b 包含指定后端服务器的日志，如果没有使用-b 或-c，那么 varnishtop 但当这两种角色。

-C 使用正则表达式的时候忽略大小写

-c 包含指定客户端的日志，如果没有使用-b 或-c，varnishtop 但当这两种角

色。

-d 启动的时候使用旧的日志记录，一般情况下，varnishtop 只读取他启动以后生成的日志。

-f 只显示日志的第一列。

-I regex 匹配正则表达式的日志，如果没有使用-i 或者-I，那么所有的日志都会匹配。

-i tag 匹配指定的 tag，如果没有使用-i 或者-I，那么所有的日志都会被匹配。

-n 指定 varnish 实例的名字，用来获取日志，如果没有指定，默认使用主机名。

-r file 读入日志文件，代替共享内存。

-V 显示版本号，然后退出。

-X regex 排除匹配表达式的日志。

-x tag 排除匹配 tag 的日志。

EXAMPLES

下面这个例子显示和更新收到的 URL。

```
varnishtop -i RxURL
```

下面的例子显示连续不断的更新用户使用的用户代理：

```
varnishtop -i RxHeader -C -I ^User-Agent
```

SEE ALSO

varnishd(1)

varnishhist(1)

varnishlog(1)

varnishncsa(1)

varnishstat(1)

HISTORY

The varnishtop utility was originally developed by Poul-Henning Kamp in cooperation with Verdens Gang AS and Linpro AS, and later substantially rewritten by Dag-Erling Smørgrav. This manual page was written by Dag-Erling Smørgrav.

COPYRIGHT

这个文档的版权和 varnish 自身的版权一样，请看 LICENCE。

●shared memory logging and statistics

Varnish 使用共享内存记录和统计。因为它更快更有效，但棘手的是他没有定期写 log 文件。

当你使用附加模式打开一个文件，无论怎么书写操作系统都不会覆盖以前的数据。这样很多个线程都可以操作这个文件，互相之间没有任何联系，他们可以按照你期望的工作。

四、Frequently ask questions（F@Q）

●GENERAL QUESTIONS

What is varnish?

Varnish 是一种状态艺术，高性能的 web 加速器，它运行在 linux2.6 kernel，freebsd6/7 和 solaris 10 系统上。

包含的一些特点：

1、 现代先进的架构设计

2、 VCL，一种非常灵活的配置语言

3、 后端服务器的负载均衡和健康检查

4、 局部支持 ESI

5、 URL 地址重写

6、 优雅的处理后端服务器宕机的问题

未来的特点（试验中）

1、 支持头文件分类

2、 支持持久缓存

Varnish 是一个自由软件，他使用改进的 BSD licenced。请阅读指南，开始你的 varnish 之旅。

●How……

我们需要分配多少内存/硬盘空间给 varnish？

需要根据实际情况而定。

我认为我们目前最好的指导就是你考虑一个符合成本效益的 RAM 配置，比如 1-16G，SSD 硬盘。

除非你肯定你需要它，因为很少有主板可以安装以 TB 为单位的 RAM。

在另一方面，如果你的流量在 Gb/s，那么你需要尽可能多的分配 RAM。

我怎么让 varnish 使用更少的内存？

你不能，varnish 操作时在内存中进行的，分配多少 RAM 给进程是 kernel 来确定。

我怎么命令 varnish 的一个实例忽略某个请求的参数？

你可以通过正则表达式删除一个请求的参数

```
sub vcl_recv {
        set req.url = regsub(req.url, "\?.*", "");
}
```

我怎么强制刷新 varnish cache 中的目标？

通常调用 purging 一个内容。在 varnish 中，你至少有两种方法完成 purge。

1、 命令行

在命令行输入以下代码：

```
url.purge ^/$
```

purge 你的/目录，你可以发现 url.purge 命令使用一个正则表达式来匹配，因此“^”和“$”分别代表开始和结尾。所有以/开始的内容都会从 cache 中删除

所以，删除 cache 中所有的内容的写法如下：

```
url.purge .*
```

2、 HTTP PURGE

使用 VCL 代码完成 HTTP PURGE 的方法可以在这里找到。注意，这个方法不支持通配符。

我怎么对一个单独的 client 请求 DEBUG？

Varnishlog 工具可以很好的整理输出，有能力对我们自己的流量做 DEBUG。

ReqStart 包含客户端 IP，我们可以查看匹配他的 IP

```
$ varnishlog -c -o ReqStart 192.0.2.123
```

想要查看这个 IP 地址在后端服务器的操作，我们可以匹配 TxHeader，客户端单独的 IP 地址包含在 X-Forwarded-For 头信息里，然后发送给后端服务器。

```
$ varnishlog -b -o TxHeader 192.0.2.123
```

我怎么在请求发送到后端服务器前重写 URLS 地址？

你可以使用 regsub（）函数来完成，下面是一个例子：

```
if (req.http.host ~ "^(www.)?example.com")
 {   set req.url = regsub(req.url, "^",
"/VirtualHostBase/http/example.com:80/Sites/example.com/VirtualHostRoot");
}
```

我有一个站点，它有很多主机名，我怎么保持 cache 的内容最少?

你可以在接收到头信息中规范所有主机名，VCL 代码如下:

```
if (req.http.host ~ "^(www.)?example.com") {
set req.http.host = "example.com";
}
```

我怎么修改到后端的请求？

你可以通过 bereq 修改到后端的请求，但是你只能使用“set”值，因此你可以使用 req.url 构造一个 request：

```
sub vcl_miss {
        set bereq.url = regsub(req.url,"stream/","/");
        fetch;
}
```

我怎么定制 varnish 返回的错误信息？

使用 vcl_error 可以定制一个错误页面。默认的错误页面如下：

```
sub vcl_error {
    set obj.http.Content-Type = "text/html; charset=utf-8";

    synthetic {"
    <?xml version="1.0" encoding="utf-8"?>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
    <html>
      <head>
        <title>"} obj.status " " obj.response {"</title>
      </head>
      <body>
      <h1>Error "} obj.status " " obj.response {"</h1>
      <p>"} obj.response {"</p>
        <h3>Guru Meditation:</h3>
        <p>XID: "} req.xid {"</p>
        <address><a
href="http://www.varnish-cache.org/">Varnish</a></address>
      </body>
     </html>
     "};
    deliver;
}
```

怎样让 varnish 忽略请求的参数和只缓存目标的实例之一？

删除请求的参数使用一下表达式：

```
sub vcl_recv {
    set req.url = regsub(req.url, "\?.*", "");
}
```

●where……

我怎么找到适合我自己操作系统的安装包？

我们知道 varnish 提供 debin，ubuntu，RHEL，centos，（open）SuSE，gentoo 和 freebsd，这些系统的二进制包，不管你使用什么包管理器，请阅读“安装 varnish”

●Can I ……

我可以使用 varnish 做正向代理吗？

不可以，varnish 需要配置所有后端服务器到 VCL。

我可以在 32 位机器上运行 varnish 吗？

可以，不过在 32 位机器上，会收到一定约束。Varnish 在 32 位机器上不能支持大

于 2G 的存储。推荐使用 64 位机器。

我能够在一台机器上同时运行 varnish 和 apache 吗？

可以，很多人成功使用了这种方法，不过这会导致资争用。

我能在一个系统上运行多个 varnish 吗？

可以，指定不同的 tcp 端口就可以了。

我可以使用一个 varnish 缓存多个虚拟主机吗？

可以，但是注意内存空间。

我能查看 varnish 缓存了哪些东西吗？

一些原因导致不可能，如果一个命令列出所有缓存的内容，那么缓存的内容是上千万或者更多，这样会导致系统资源紧张，varnish 暂停服务。

我能用 varnish 加速 https 吗？

目前还不行，请密切关注我们，我们现在还没计划添加 HTTPS 支持，不过我们会想办法完成，就像 nginx 那样。

我能够使用 varnish 负载均衡后端服务器吗？

可以，你可以使用以下 VCL 代码：

```
director foobar round-robin {
        { .backend = { .host = "www1.example.com; .port = "http"; } }
        { .backend = { .host = "www2.example.com; .port = "http"; } }
}

sub vcl_recv {
        set req.backend = foobar;
}
```

(XXX: reference to docs, once written)

●Why……

为什么 varnish 将所有的请求都发到后端服务器去了？

通常是两个问题导致的：

1、 目标的 TTL 过短。通常的解决方式是后端服务器不设置 TTL，使用 varnish 默认的 TTL。（默认是 120s）

2、 你使用了 cookies：

默认情况下，varnish 不会缓存 backend 返回的带 cookie 的结果。

默认情况下，varnish 不对包含 cookie 的请求服务，直接 pass 他们到后端服务器。

为什么正则表达式有大小写之分？

在 2.10 版本之前使用 POSIX 的正则表达式没有大小写区分，但是 2.10 以后，使用了 PCRE 正则表达式，它就对大小写敏感了。（我们保证以后不会在改变这种思想）

让 PCRE 正则表达式不区分大小写，使用（？!）开始：

```
if (req.http.host ~ "?iexample.com$") {
        ...
}
```

查看 PCRE man pages 了解更多信息。

为什么头信息中“Via：”显示 1.1，而 varnish 版本是 2.1.x？

Via：number 是 HTTP 协议，不是 varnish 版本号。

为什么我们叫它“varnish”？

1、 像 varnish 一样覆盖

2、 顺利和平滑的完成任务

3、 掩盖所有，赋予光滑的表面

为什么安装 varnish 的系统需要 C 编译器？

VCL 编译需要 C 编译器来完成。系统的 C 编译器把 VCL 编译成动态共享库，然后供 varnish 使用。所以没有 C 编译器，varnish 不能运行。

安全问题?

你可以在必须的时候修改 c 编译器的权限或者删除 C 编译器。

●排错

为什么我收到一个缓存命中，而请求还是会发送到后端服务器？

Varnish 有一个特色，就是 hit for pass，当 varnish 得到一个后端服务器的响应但是发现这个目标不能被缓存，在这种情况，varnish 将创建一个缓存目标记录这个事件，所以下次请求就会直接被 pass。

●HTTP

X-Varnish HTTP 头的用途？

X-Varnish HTTP 头可以让你找到正确的日志记录，比如命中 cache，X-Varnish 控制当前请求的次数和增加 cache 中的请求数。它让 debugging 非常容易。

Varnish 支持压缩吗？

这个简单的问题有一个复杂的回答，请看 WIKI

我怎么样添加一个 HTTP 头？

添加一个 HTTP 头，除非你想要添加一些关于客户端请求的东西，添加 HTTP 头最好在 vcl_fetch 中完成，这个方法将处理每个 fetched 的目标：

```
sub vcl_fetch {
  # Add a unique header containing the cache servers IP address:
  remove obj.http.X-Varnish-IP;
  set       obj.http.X-Varnish-IP = server.ip;
```

```
    # Another header:
    set       obj.http.Foo = "bar";
}
```

我怎么样才能在后端服务器记录客户端的 IP 地址？

通常我们看见的 IP 地址是 varnish 服务器的，怎么样我们才能看见客户端的 IP 地址呢？

我们需要添加这些 IP 地址到一个头，然后和请求一起发送给后端服务器，然后配置后端服务器的日志记录这个头信息的内容：

Varnish configuration：

```
sub vcl_recv {
    # Add a unique header containing the client address
    remove req.http.X-Forwarded-For;
    set       req.http.X-Forwarded-For = client.ip;
    # [...]
}
```

以 apache 配置为例，我们拷贝 combined 日志格式，改名为“varnishcombined”，我们在格式中加入 varnish 定义的头：

```
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" varnishcombined
```

然后在你的主机配置部分，使用定制的日志格式替换标准的格式：

```
<VirtualHost *:80>
    ServerName www.example.com
    # [...]
    varnishcombined /var/log/apache2/www.example.com/access.log varnishcombined
    # [...]
</VirtualHost>
```

The [http://www.openinfo.co.uk/apache/index.html mod_extract_forwarded Apache module] might also be useful.

●Configuration

什么是 VCL？

VCL 是首字母 Varnish Configuration Language。在 VCL 文件中，你配置 varnish 怎么运转，VCL 示例文件爱呢包含在此 WIKI 的后面位置。

哪里有 VCL 的说明文档？

我们正在整理 VCL 的说明文档，在 WIKI 有一些示例。

或者请查看 man 7 vcl

我怎么在 varnish 运行的时候加载 VCL 配置文件？

1、 把 VCL 文件放到对应服务器上

2、 进入 varnishadm 管理界面

3、 使用 vcl.load <configname> <filename> 加载配置文件，configname 是配置的名字，自己随便取，filename，就是配置文件的绝对路径

4、 使用 vcl.use <configname> 启用新的配置文件。

我是否应该使用 “pipe”、“or”，“pass”在我的 VCL 代码中？他们有什么不同？

当 varnish 使用 pass，他的动作就想正常的 HTTP 代理，他读取请求然后推送到后端服务器，下一个 HTTP 请求仍然可以做任何其他处理。

pipe 是在某些在不能使用 pass 的特殊情况，pipe 读取请求，然后把通一个客户端的其他请求都连续不断的推送给后端服务器，没有其他动作。

因为很多 HTTP 客户端在一个连接中，有数个连接使用 pipe 模式，这将给你带来不好的后果，每个子请求都会使用存在的 pipe 模式。

在 varnish2.0 版本还没支持 pass 对请求 body 的处理，所以在这个版本必须正确使用 pipe 模式。

在 2.0 以后的版本，pass 将正确的处理请求 body。

如果你的一个请求获得 503 错误，务必在 vcl_recv pass 前检查你指定的后端。

●Logging

我们在哪里可以找到日志文件？

Varnish 默认不保存日志，但是可以通过 varnishlog 工具打印共享内存的日志，或者使用 varnishncsa 工具保存日志成 Apache/NCSA 格式。

●Varnish Glossary(varnish 术语)

backend

需要 varnish cache 的 HTTP 服务器，可以处理一些通过策略分类后的 HTTP 请求，但是不限制于一个 web 服务器，一个 CMS，一个负载均衡器，或者其他 varnish。

body

构成目标的 bytes，varnish 不在乎他们是 HTML，XML，JPEG 或者 EBCDIC，对 vanrish 来说他们都是是 bytes 而已。

client

给 varnish 发送 HTTP 请求的程序，典型的比如一个浏览器，还有搜索引擎的蜘蛛，脚本，criminals 等。

header

一个 HTTP 协议的头部，比如“Accept-Encoding：”。

hit

从 varnish cache 中获得目标。

master（process）

Varnish 两个进程中的一个，master 进程就是一个管理程序，负责管理配置，参数，编辑他们，但是他从来不与真正的 HTTP 流接触。

miss

一个目标从后端服务器获得，根据情况，是否把目标放入 varnish cache。

object

Cached 中的一个内容回应。varnish 从后端服务器收到一个回复，并创建一个目标，那些被 cache 的内容可以回应客户端。

pass

Varnish 不尝试在 cache 中寻找目标，简单的从后端服务器获取。

pipe

Varnish 只是转发请求到后端，具体请求做什么，完全不管。

request

Client 发送给 varnish 和 varnish 发送给后端的东西。

response

后端发送给 varnish 和 varnish 发送给 client 的东西，当 response 存储在 varnish 的 cache 里，我们叫他 object。

varnish（NB:with ‘d’）

这是真正的 varnish cache 程序，他是只是一个程序，不过，当你运行的时候，就会产生两个进程，master 和 worker（或者叫 child）。

varnishhist

Eye-candy program showing responsetime histogram in 1980ies ASCII-art style.

varnishlog

使用指定格式显示 varnish 的事务处理日志的程序

varnishncsa

使用 NCSA 格式显示 varnish 的事务处理日志的程序。

varnishstat

varnish 事务处理计数器程序。

varnishtest

使用指定脚本测试 varnish。

varnishtop

跟踪指定日志使用的时间.

vcl

Varnish 配置文件

worker（process）

这个进程在试 master 进程配置和启动的，他处理几乎所有的我们希望 varnish 处理的工作，当它宕了以后，master 会尝试启动它，保持 website 可用。