# varnish 管理员文档

varnish缓存是web应用加速器，同时也作为http反向缓存代理被人熟知。你可以安装varnish在任何http的前端，同时配置它缓存内容。varnish真的很快，单个代理的分发速度可以达到300-1000x，依赖与你的服务器架构。

开始使用varnish我们建议你阅读安装向导，[Varnish Installation](https://www.varnish-cache.org/docs/4.0/installation/index.html)。一旦varnish启动运行，我们建议你使用基础教程[Thte Varnish tutorial](https://www.varnish-cache.org/docs/4.0/tutorial/index.html),最后再是[The Varnish User Guide](https://www.varnish-cache.org/docs/4.0/users-guide/index.html).

如果你想了解怎样使用varnish特定工具，[The Varnish Reference Manual](https://www.varnish-cache.org/docs/4.0/reference/index.html)包含这些工具的详细文档。 varnish4.0较3.x的变动，你可以查看[What's new in Varnish 4.0](https://www.varnish-cache.org/docs/4.0/whats-new/index.html)。最后，我们从[Poul-Henning Kamp](https://www.varnish-cache.org/docs/4.0/phk/index.html)收集了一些关于varnish和http的博客。

通常情况可以使用：

service varnish restart

要么使用终端或者使用工具。 /usr/local/, varnishadm, sess\_timeout 各种各样的配置参数或者路径

http://www.varnish-cache.org/ 官方网站的超链接

长的列表命令输出和VCL看起来像下面这样：

$ /opt/varnish/sbin/varnishd -V

varnishd (varnish-4.0.0-tp1 revision ddd00e1)

Copyright (c) 2006 Verdens Gang AS

Copyright (c) 2006-2011 Varnish Software AS

**Varnish Installation varnish的安装**

本节包含了varnish安装的准备工作，一步一步的安装过程，怎样获取帮助，在哪里寻求帮助，已经怎样报告bug。也包含了帮助你在某些特殊的系统上安装varnish的注意事项.

**Prerequisites 需要的配置**

如果你要安装Varnish你必须满足一下条件:

* 全新的，高配置的电脑。
* 64位的Linux、FreeBSD或者Solaris(x86 only)
* Root 权限

varnish也可以安装在其他的Unix操作系统上，但是我们是没有广泛和系统的测试。varnish也时不时的被提及工作在:

* 32位的Linux、FreeBSD
* OS X,
* NetBSD,
* OpenBSD,
* Windows 上的Cywin

**Installing Varnish安装Varnish**

对于开源软件你可以选择二进制包安装或者自己编译源码安装。二进制安装或者源码安装只和个人的口味有关系。如果你不知道选哪一个，我们推荐你读整个章节，然后选你觉得最合适那一个。

**Source or packages? 源码还是二进制包**

安装Varnish具体的系统包管理和当前操作系统相关的，典型例子:

**FreeBSD:**

二进制包安装

pkg\_add -r varnish

源码安装

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

**Red-Hat/CentOS**

我们尽量保持最新版的varnish预编译rpm在repo.varnish-cache.org 查看在线的Red-Hat安装指南获取更多信息 vanish包含在EPEL仓库里，由于新版本的varnish语法和老版本的冲突，所以只要老版本的可用。 因此我们建议你直接从我们的仓库安装最新版的varnish，就像之前说的那样。

**Debian/Ubuntu**

Varnish是独立的在Debian和Ubuntu上。获取varnish可以运行

sudo apt-get install varnish

请注意这个varnish可能不是最新版的。如果你需要最新版的Varnish，请查看Debian和Ubuntu在线安装指南

**Compiling Varnish from source 源码编译安装**

如果你的系统没有二进制包，或者你想通过源码包编译亦或是其他原因，跟进下面的步骤:

从http://repo.varnish-cache.org/source/下载合适的tar包，

或者你想修改Varnish，可以clone我们的git仓库，

git clone git://git.varnish-cache.org/varnish-cache

**构建Debian/Ubuntu上的依赖环境**

为了从源码构建Varnish你需要安装一些包，在Debian或者Ubuntu上，需要这些：

* automake
* autotools-dev
* libedit-dev
* libjemalloc-dev
* libncurses-dev
* libpcre3-dev
* libtool
* pkg-config
* python-docutils
* python-sphinx

**构建Red-Hat/CentOS的依赖环境**

在Red-had或者CentOS下面这些包是需要安装的：

* autoconf
* automake
* jemalloc-devel
* libedit-devel
* libtool
* ncurses-devel
* pcre-devel
* pkgconfig
* python-docutils
* python-sphinx

**编译varnish**

配置的是基于依赖关系满足的条件，一旦满足了，你就可以执行以下指令:

cd varnish-cache

sh autogen.sh

sh configure

make

configure脚本可以引入一些参数，但是现在请忘记这些，我们可以在varnish运行的时候来添加参数。 安装之前，你可能想运行测试套件，测试的话要花上好几分钟，你可以来杯茶了。。

make check

如果出现一两个错误，请不要担心，但是如果有一些失败的，特别的像 b00000.vtc是测试失败，那确实是有问题了，你需要检查下configure是否异常了。

**Installing 安装**

最后呢，真正对与勇敢的心的测试：

sudo make install

Varnish将安装在/usr/local ，varnishd二进制是在/usr/local/sbin/varnishd ,默认的配置文件在/usr/local/etc/varnish/default.vcl。

**Get Help 获取帮助**

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. UPLEX info@uplex.de

**Reporting Bugs 报告bug**

**Platform specific notes 特殊系统注意事项**

本节你会了解到某些系统在运行varnish之前，你需要如何调整系统。

**varnish 4.0简要教程**

这部分教程包含了varnish基本原理。囊括了什么是varnish、它怎样工作，同时也包含了开始使用和运行varnish。这部分过后你可能想继续了解用户向导(varnish用户引导)。 如果你正在web上读这些呢 注意每页左边的Next topic和Previous topic

**使用varnish作为web代理缓存的原理**

varnish是一个http反向代理的缓存。它从客户端接收请求然后尝试从缓存中获取数据来响应客户端的请求，如果varnish不能从缓存中获得数据来响应客户端，它将转发请求到后端（backend servers）,获取响应同时存储，最后交付给客户端。

如果varnish已经缓存了某个响应，它比你传统的后端服务器的响应要快很多，所以你需要尽可能是更多的请求直接从varnish的缓存中获取响应。

varnish决定是缓存内容或者是从后端服务器获取响应。后端服务器能通过http响应头中的Cache-Control来同事varnish缓存内容。在某些条件下varnish将不缓存内容，最常见的是使用cookie。当一个被标记有cookie的客户端web请求，varnish默认是不缓存。这些众多的varnish功能特点都是可以通过写vcl来改变的。

**performance 性能**

varnish使用现代先进的架构设计。通常依赖于网络的速度，而不是性能问题。你应该更关心你的web应用的处理能力，而不是varnish的性能。

**Flexibility 灵活的**

提高varnish缓存性能的关键点，是它灵活的配置语言，VCL。VCL允许你编写如何处理请求的规则。通过使用规则你可以决定如何处理内容，从哪里获取内容，以及怎样修改请求和响应。

**Supported platforms 支持的平台**

varnish开发来是运行当前的Linux和FreeBSD的版本上，当然最好能就运行在这些系统之上。感谢这些贡献者也让varnish能运行在 NetBSD, OpenBSD, OS X 以及各种各样的Solaris系统，像Oracle Solaris, OmniOS and SmartOS。

**About the Varnish development processvarnish开发流程**

varnish是一个社区驱动项目。Varnish Governing Board监督varnish的开发，成员包括Poul-Henning Kamp (Architect), Rogier Mulhuijzen (Fastly) and Lasse Karstensen (Varnish Software). 如果你想为varnish做出贡献，请查看这个链接：https://www.varnish-cache.org/trac/wiki/Contributing

**Starting Varnish**

本教程假定你已经运行了Varnish在Ubuntu，Debian，Enterprise Linux 或者是 CentOS上。 如果你要运行varnish在另一些系统上，你需要费点脑力来转换了。以后你就可以尽情的使用它了，即便是在你的weird（怪癖的）的系统上确定你已经成功的安装好了varnish（下面的操作在Installing Varnish之上） 当你正确的安装好varnish后就可以使用

service varnish start

来启动varnish，如果varnish没有启动，刚刚的指令将启动varnish,现在你的vanish运行起来了。 通过浏览器来访问http://127.0.0.1:6081/（使用你自己机器的ip来替换127.0.0.1）。默认配置转发请求给web应用。Varnish认为web应用是监听在8080端口。 如果没有web应用监听在本机的8080端口，varnish将报错。当出现问题的时候，varnish cache友好的提供给你一个页面像"Error 503 Service Unavailable" 你可以运行web应用在其他端口或者其他主机。现在我们编辑配置文件，让他指向某个工作的主机。 选一个你喜欢的编辑器，编辑/etc/varnish/default.vcl 这个文件大部分是注释掉的，有些没有，像下面的：

vcl 4.0;

backend default {

    .host = "127.0.0.1";

    .port = "8080";

}

现在我们改变这个vcl配置文件，指向某个活动主机，幸运的是http://www.varnish-cache.org/是活动的，我们就用这个http://www.varnish-cache.org/ 在vcl中替换成下面这样。 vcl 4.0;

backend default {

    .host = "www.varnish-cache.org";

    .port = "80";

}

执行

service varnish reload

使Varnish重载它的配置文件。如果成功，你在浏览器访问http://127.0.0.1:6081 将看到一些目录列表。你没有看到varnish的官方站点是因为你客户端请求的头里没有设置合适的Host,然后服务器以默认的文件夹列表来响应。

**Put Varnishon port 80 让varnish监听在80端口**

出于测试目的我们一直让varnish运行在一个高的端口。现在我们让varnish监听在80端口 首先停止varnish ：service varnish stop 我们需要编辑配置文件然后启动Varnish

**Debian/Ubuntu**

Debian/Ubuntu是这个配置文件/etc/default/varnish。这个文件看起来像这样： DAEMON*OPTS="-a :6081 \ -T localhost:6082 \ -f /etc/varnish/default.vcl \ -S /etc/varnish/secret \ -s malloc,256m" 修改成这样 DAEMON*OPTS="-a :80 \ -T localhost:6082 \ -f /etc/varnish/default.vcl \ -S /etc/varnish/secret \ -s malloc,256m"

**Red Hat Enterprise Linux / CentOS**

Red Hat/CentOS配置文件路径为/etc/sysconfig/varnish，修改方式相同

**Restarting Varnish again**

修改完成后就可以重启Varnish：

service varnish start.

现在所有人访问你的网站都会通过Varnish了。

**Backend servers 后端服务**

Varnish有个后端或者源服务器的概念。一个后端服务器是提供内容的服务器，Varnish将通过使用缓存内容来加速。 第一个任务是告知Varnish哪里可以获取到这些内容。开启你喜欢的编辑器，打开默认的配置文件。如果你通过源码安装默认的配置文件是这个/usr/local/etc/varnish/default.vcl 如果你通过包安装（想apt-get或者yum），它可能是这个/etc/varnish/default.vcl。 如果你跟进这个教程，本节你的配置文件看起来是这样的：

vcl 4.0;

backend default {

    .host = "www.varnish-cache.org";

    .port = "80";

}

配置文件的意思是我们在varnish里配置了一个后端，从www.varnish-cache.org的80端口获取内容。 如果你不想做一个varnish-cache.org的镜像，我们需要调整Varnish从你自己的源服务器获取内容。我们已经绑定Varnish在80端口，现在我们需要把它和源服务器联系起来。 像下面这个例子，我们假冒了个监听在localhost 8080端口的源服务器。

vcl 4.0;

backend default {

.host = "127.0.0.1";

.port = "8080";

}

varnish可以定义很多个后端，甚至是将这个后端添加到一个集群里面使它们达到服务在均衡的目的，Varnish根据当前的调度算法挑选出一个后端 下一步，让我们看看varnish有什么独特的地方，同时我们能用它做点什么。

**Peculiarities 特性**

varnish cache和其他的一些项目是有所不同的，其中一个就是你已经看到的VCL。本机我们将快速向你展示你需要了解到的关于varnish的其他特性

**Configuration 配置**

varnish用VCL来编写配置文件。当varnish使用配置文件，它将配置文件转换成C代码，载入到C的编译器，然后执行 这样的话你调整的各种各样设置，你编写的处理请求的vcl规则将被处理。

**varnishadm**

varnish缓存有个管理控制台(varnishadm) ,你可以通过使用命令varnishadm来连接管理控制台，为了认证你需要有/etc/varnish/secret的读取权限。 你可以利用varnishadm来停止、开启缓存进程、载入VCL规则、调整负载均衡器、清除缓存等等，然后退出。 varnishadm 内建命令help帮助你了解到varnishadm能做些什么。

**varnishlog**

Varnish默认不记录日志到磁盘，而是记录在一块内存里，实际上是日志流。任何时间你都可以连接它，看看发生了什么。Varnish记录了一些信息。你也可以使用varnishlog来查看日志流

**Now what?**

varnish简要教程现在就完了，你的varnish现在应该已经跑起来了。你应该看看varnish的日志，你的VCL配置文件也是粗糙的。下一步，你可能需要看看Varnish用户引导，我们能看到更多varnish详细的特点，

# The Varnish Users Guide varnish用户手册

# The Big Varnish Picture varnish概况

本节将回答这个问题：“varnish”到底是什么，命名的点点滴滴。

varnish两个主要的部分是在varnisd项目中的两个进程。第一个进程被叫做“the manager”,主要功能是告诉你，系统管理员，发生了什么。

第二个进程被叫做“the worker”或者是“the child”，这个进程是实际处理你http请求的进程。

当你启动varnishd，manager进程启动，一旦manager处理完所有的命令行参数，child进程就开始启动了。如果child进程down掉，manager进程会重启child，自动的，无需干预。

这样设计的主要原因是为了安全考虑，manager为了能监听在80端口，以root运行，但是child进程只需要很小的权限。

manager进程是互动的，提供了CLI-命令行接口，可以手动，脚本以及程序调用。CLI提供Varnish处理HTTP请求的几乎全部控制，同时我们已经竭尽全力，以确保你不应该需要重新启动varnish，除非你需要修改最底层的东西。

CLI是远程访问安全的，使用简单灵活的PSK -- Pre Shared Key，所以很容易集成到你的业务或者绑定到你的CMS

所有的这些都包含在[Starting and running Varnish](https://www.varnish-cache.org/docs/4.0/users-guide/running.html) 比如，子进程怎样处理http请求，缓存什么，哪个http头需要移除等等，都是用一个轻量级的编程语言，叫VCL--Varnish配置语言。manager进程将编译VCL程序同时检查是否出错，但是是child进程来运行VCL程序，在处理请求时。

因为VCL编译成C代码，C代码编译成机器指令，因此即便是复杂的VCL程序执行也是毫秒级的，不会对性能产生影响。 如果你不是一个真正的程序员，不要烦恼，VCL是很简单：

sub vcl\_recv {

        # Remove the cookie header to enable caching

        unset req.http.cookie;

}

CLI允许你随时编译和加载行的VCL程序，你也可以实时的转换两套程序，不需要重启child进程，同时不会丢失http请求。 VCL代码可以使用VMODs扩展模块来扩展，如果你可以甚至是C代码。因此varnish能对你的http请求做什么，真的是没有限制的。

[VCL - Varnish Configuration Language](https://www.varnish-cache.org/docs/4.0/users-guide/vcl.html)描绘了VCL以及它能详细的做什么， varnish使用共享内存分块的方式报告和记录实时状态。对于每个http请求，大量的详细的记录是追加到log的内存块。其他进程可以查看日志记录，过滤，以Apache/NCSA风格为例。 其他的内存块是用来统计计数的，实时的，精确到毫秒级的监控缓存，资源使用情况以及各项性能指标。

varnish有大量的工具可以使用共享内存，varnishlog，varnishstats，varnishncsa等等，你也可以使用API库编写你自己的工具，只要你需要。

[Reporting and statistics](https://www.varnish-cache.org/docs/4.0/users-guide/report.html#users-report) 报告和统计 说明所有的工作原理。 对varnish感兴趣的最可能的原因，是你想让你的网站工作的更好。加速一个web站点是有很多方面的。一个简单的策略是使用缓存，设计物理分割的多等级的使用ESI的CDN缓存，宕机自动切换。

# 启动和运行varnish

本节包含了启动，运行，停止varnish，命令行参数，如何与运行着的varnish进程进行通信，配置存储和套接字，安全性和如何保护varnish免受攻击。

# Security first 安全第一

如果你仅仅抓专注于运行varnish,或者所有人都是这个观点,那么就可以跳过该节了.我们保护你的方方面面当http请求来的时候.

如果你的web基础设施是外包的,管理是分开的,那么你就需要考虑安全了. Varnish在4个权威的等级提供安全,大致关系到怎样让命令生效和在哪儿生效.

## Command line arguments 命令行参数

最高安全定理是决定和定义varnish的启动参数在一个表单里面,我们使用这个策略是为了新的操作也是安全的. 最重要的的决定是确保这些:

* Who should have access to the Command Line Interface? 谁有命令行的权限?
* Which parameters can they change? 他们可以修改哪些参数?
* Will inline-C code be allowed? 哪些内建C代码是被允许的?
* If/how VMODs will be restricted? 是否VMODs是被限制,怎样被限制?

## CLI interface access CLI接口权限

命令行接口可以通过三个方式进入. Varnish监听状态时，提供给CLI链接一个TCP套接字。你可以绑定这个套接字在内核接受的任意地方。

-T 127.0.0.1:631

-T localhost:9999

-T 192.168.1.1:34

-T '[fe80::1]:8082'

默认配置'-T localhost:0'将挑选一个随机端口，通过使用varnishadm(8)可以在共享内存中了解到。

使用localhost地址，仅仅限制本机的链接。

你也可以绑定端口在网络可达的物理地址，然后让其他机器直接连接。

这不是保密的，CLI命令讲作为非加密的ASCII文本进行网络传输，但是-S/PSK需要远程终端知道共享密钥。

你可以绑定CLI端口在localhost，使用ssh/vpn或者类似的协议，给远程用户一个安全连接到本机的权限。

如果使用ssh每个用户可以执行那些命令，就像varnishadm，或者封装一个仅仅允许执行特定CLI命令的varnishadm.

你也可以配置varnishd反向监听模式，使用'-M'参数。varnishd尝试给指定的地址打开一个TCP连接，然后给vanish管理设备发起一个CLI连接。

这个连接也是非加密的，但是远程终端必须满足-S/PSK认证。

最后，如果你使用'-d'模式运行varnish，你可以获得一个CLI命令在标准输入和标准输出，但是一旦你开始进程，还有什么能阻止你活取CLI权限，不是么？

## CLI interface authentication CLI接口认证

默认的CLI接口被一个简单却强壮的"Pre Shared Key"认证模式，但是不提供加密（例如，CLI命令和响应是非加密的）.

-S/PSK工作原理真的很简单：当varnish安装，使用随机内容创建了一个文件，这个文件仅仅只有启动varnish的用户（或者超级用户）才有权限。

为了认证和使用CLI连接，你需要知道这个文件的内容，为了了解挑战加密的问题。

(XXX: xref to algo in refman) .. XXX:Dunno what this is? benc

varnishadm 使用这些策略来限制权限，仅仅在varnishadm起作用和需要它的时候才读取密钥文件。

如果你打算允许其他用户，本地的和远程的，为了获得CLI的权限你可以创建自己的密钥文件，同时只可能被这些用户读取。

一个创建密钥文件的好方法是：

dd if=/dev/random of=/etc/varnish\_secret count=1

当你启动varnishd，你通过'-S'指定密钥文件名,同时没有说明varnishd主进程也需要读取文件。
当varnishd运行的时候你也可以修改密钥文件的内容，因为每次CLI连接认证时都读取密钥文件。

在本地系统你可以通过共享内存来恢复密钥文件名，但是在远程系统，你需要使用'-S'参数给varnishadm复制一个密钥文件。

如果你要禁用 -S/PSK认证，指定'-S'一个空的参数给varnishd：

varnishd [...] -S "" [...]

## Parameters 参数

varnish可以是通过命令行设置参数，同时添加'-r'参数作为只读模式，这样的话就不能在CLI接口修改参数了。

相当多的参数可能搞砸你的http服务,但是有些是更严重的:

user and group 用户和组

Access to local system via VCL 通过VCL认证本地系统.

listen\_address 监听地址

Trojan other TCP sockets, like ssh 入侵其他tcp套接字,不如ssh

cc\_command

Execute arbitrary programs 执行随意程序.

vcc\_allow\_inline\_c

Allow inline C in VCL, which would any C code from VCL to be executed by Varnish. 允许C代码在VCL中,哪些代码C代码可以在varnish里呗执行.

Furthermore you may want to look at and lock down: 此外你可能想看:

syslog\_cli\_traffic

Log all CLI commands to syslog(8), so you know what goes on.CLI命令的所有日志你可以查看syslog(8)

vcc\_unsafe\_path

Restrict VCL/VMODS to vcl\_dir and vmod\_dir 限制VCL/VMODS的vcl\_dir和vmod\_dir.

vmod\_dir

The directory where Varnish will will look for modules. This could potentially be used to load rouge modules into Varnish.

这个目录里可以看到很多模块,罗列了可以载入到varnish的潜在模块.

## The CLI interface CLI接口

CLI接口功能强大,只要你有权限连接到CLI接口,你可以对varnish进程做几乎所有的事情.

正如之前的描述,一些有危害的参数可以被限制,但是这些都只能保护本地文件系统和操作系统,并不能保护你的http服务.

当前我们还没有办法对特定的CLI连接限制特定的CLI命令.有个办法就是封装可能有影响的连接在一个预认证的脚本里,

使用varnishadm(1)提交CLI审核命令,同时限定远程用户只能通过这些脚本来连接,例如使用sshd(8)的配置.

## VCL programs VCL程序

在VCL模式中有两个危险的机制:VMODs和inline-C.

这两个机制允许执行任意代码,因此允许用户获得机器的权限和说有child进程的权限.

如果varnish使用root/supper启动,我们在沙箱里运行的child进程可以使用任何可以在操作系统上使用的工具,但是如果不是,就不需要考虑这些了.不要问我为什么你会用超级用户来启动一个只需要低权限的child进程.

Inline-C在varnish4.x启动时是不起作用的,除非启用了Inline-C,所以你不用担心它.

之前提到的参数可以限制加载VMODS只能在指定的目录,限制VCL争议者在一个VMODs的预认证子集.

如果你这样做,我们确信你的本地系统在使用VCL模式不可能缺乏攻击抵抗能力.

## HTTP requests http请求

我们竭尽所能的使vanish抵抗那些通过套接字接收的http请求,同时你可能,通俗来说,不需要你进一步的保护.

这个警告自从VCL变成可编程语言,让你决定完全对HTTP请求做些什么的语言,你也可以使用VCL决定做些愚蠢和潜在危险的事情,包括开启你自己的各种攻击和破坏性的活动.

如果你需要管理(包括,修改,删除,产生新动作等等)'HTTP'请求,例如PURGE请求,我们强烈建议你限制这些操作,使用VCL的权限控制列表来允许信任的ip或着网络才能管理HTTP请求.

# 重要的命令行参数

当启动varnish时有两个重要的参数你必须设置:

一个是处理http请求的tcp监听端口,另一个是处理真实请求的后端server

如果你使用操作系统自带的包管理工具安装的varnish,你将在下面的文件找到启动参数:

Debian, Ubuntu: /etc/default/varnish

Red Hat, Centos: /etc/sysconfig/varnish

FreeBSD: /etc/rc.conf (See also: /usr/local/etc/rc.d/varnishd)

## '-a' listen\_address

'-a' 参数定义了varnish监听在哪个地址,并用该地址处理http请求

你可能想设置这个参数在众所周知的http 80端口.

你也指定多个地址用逗号隔开就行了,你可以使用数字或者主机/服务名只要你喜欢,varnish尝试打开,如果不能打开,varnish将不能启动.

例子:

-a :80

-a localhost:80

-a 192.168.1.100:8080

-a '[fe80::1]:80'

-a '0.0.0.0:8080,[::]:8081'

如果你的webserver和varnish运行在同一台机器,你必须换一个监听地址.

## '-f' VCL-file or '-b' backend

-f添加vcl文件,-b定义后端server

varnish需要知道从哪里找到这个需要缓存的http server.你可以用-b参数指定,或者帮把它放在vcl文件中,然后使用-f参数指定.

在启动的时候使用-b是一个快捷的方式.

-b localhost:81

-b thatotherserver.example.com:80

-b 192.168.1.2:80

注意:如果你指定的是name,这个name必须能解析成一个IPv4或者IPv6的地址

如果你使用-f参数,你启动的时候可以在-f指定的vcl文件中包含,像这样:

backend default {

        .host = "localhost:81";

}

这恰恰是-b参数做的事

这两种情况下都是将配置参数追加到内建的VCL代码.

# 其他选项

Varnish拥有大量的有用的命令行参数.我们建议你学习他们但是没必要全部使用,如果仅仅是启动,这些是足够了的.

默认的varnish使用100M的内存来缓存对象,如果你想缓存更多,你应当查看-s参数.

如果你运行一个大的web站点,你可能想使用-p参数调整work线程的数量,但是我们通常不建议这样做除非你需要这样.

在部署到生产环境前,如果你限制管理权限,你可能需要重新查看[Security First](http://my.oschina.net/monkeyzhu/blog/413059)章节 查看OPTIONS获得全部的命令行参数

**CLI-bossing Varnish around**

CLI 向Vanish下发指令

一旦vanish启动，你可以使用命令行接口来控制它。

最轻松的使用CLI的方法是在运行varnishd的机器上运行，varnishadm

varnishadm help

如果你想远程使用varnishadm，有两种方法。

你可以使用ssh登录到运行varnishd的机器上然后varnishadm

ssh $http\_front\_end varnishadm help

同时你也可以配置vanishd接受远程的CLI连接，（使用-T和-S参数）

varnishd -T :6082 -S /etc/varnish\_secret

然后在远程主机上运行vanishadm，像这样

varnishadm -T $http\_front\_end -S /etc/copy\_of\_varnish\_secret help

但是你也看到，ssh是更方便的。

如果运行vanishadm没有添加参数，它将会从标准输入读取CLI命令，如果你指定了参数，它将这些参数作为单个CLI命令来执行。

CLI总是返回一个状态码，告诉你操作的结果，'200'为成功，其他状态码就有问题了。

vanishadm退出时使用状态1，并且如果不是200的状态码，它将在标准错误输出中打印状态码。

**What can you do with the CLI**

你能用CLI做些什么呢？

CLI提供给你几乎对vanishd所有的控制，一些重要的操作你可以执行：

* load/use/discard VCL programs 载入/使用/禁用VCL程序
* ban (invalidate) cache content 清除缓存
* change parameters 改变参数
* start/stop worker process 启动和停止work进程

接下来我们先简单的讨论这些。

**Load, use and discard VCL programs**

所有的缓存和策略定义都是通过VCL程序。

你可以载入多个VCL程序，同时它们中被指定成“active”的VCL程序，这些active的VCL程序将处理新的请求。

加载新的VCLc程序：

varnish> vcl.load some\_name some\_filename

加载过程将读取VCL程序文件，然后编译它，如果编译失败，就会提示类似如下错误：

.../mask is not numeric.

('input' Line 4 Pos 17)

                "192.168.2.0/24x",

----------------#################-

Running VCC-compiler failed, exit 1

VCL compilation failed

如果编译成功，VCL程序就被加载成功，同时你就可以激活它了，随你什么时候：

varnish> vcl.use some\_name

如果你觉得有异常（通常是在测试后），你可以又回滚到之前的VCL程序：

varnish> vcl.use old\_name

这种转换是瞬时的，所有新的请求就立即开始使用active的VCL程序。当前的请求将被使用VCL来处理。

有个好主意就是设计一个应急的VCL，同时总是加载它，那么你就可以转换它使用单个的vcl.use 命令。

**Ban cache content**禁止缓存

varnish提供"purges"来从缓存中移除某些内容，假如你明确的知道他们是些什么。

但是有时它不需要你指定明确的需要移除的内容。

想象个场景，公司的logo变了，现在需要从varnish cache中清除老的log:

varnish> ban req.url ~ "logo.\*[.]png"

就应该这样做，是的，正则表达式。

我们调用"banning"因为这些对象仍然是在cache中的，但是它们在传递过程中被禁用了。

当接收到http请求，我们测试的正则表达式的每个对象，而不是立即检查每一个缓存了的对象。

Banning实现比直接重启varnish来摆脱这些错误的缓存内容要更廉价。

**Change parameters**

改变参数 参数可以通过命令行添加'-p'来设置，同时他们也被检查和改变从运行的CLI中。

varnish> param.show prefer\_ipv6

200

prefer\_ipv6         off [bool]

                    Default is off

                    Prefer IPv6 address when connecting to backends

                    which have both IPv4 and IPv6 addresses.

varnish> param.set prefer\_ipv6 true

200

通常修改参数不是个好主意，除非你有个好主意，像性能调优或者安全配置。

但是某些参数是需要重启子进程才能生效。这些总是在参数的描述标明。

**Starting and stopping the worker process**启动和停止worker进程

通常你应当让worker进程运行着，但是如果你需要停止或者启动它，可以使用下面的命令：

varnish> stop

然后：

varnish> start

如果你启动varnishd使用了-d（debugging）参数，你将总是需要明确地启动child进程。

有可能child进程die（not work），主进程将自动重启child进程，但是你也可以禁用自动重启，使用 'auto\_restart'参数

# Storage backends存储后端

## Intro

varnish拥有可插入式的存储后端，它可以存储数据在在各种不同表现特性的后端中。默认的配置就是使用限制了大小的malloc后端（基于内存的）。对于一些vanish部署方式，你可能需要调整储存设置来适合相应的部署方式。

## malloc

syntax: malloc[,size]

malloc是基于内存的后端。每个对象都将从内存中分配资源来存储。如果你的系统运行内存较小，swap将会被使用。

请注意，该尺寸限制只限制了实际存储，同时使用各种内部机制使得每个对象在内存中大约占1k，这种情况也包括在真实的存储中。

size参数指定最大分配给vanishd的内存数量。size的假定单位是byte，除非你指定下面的一个后缀：

K, k The size is expressed in kibibytes.

M, m The size is expressed in mebibytes.

G, g The size is expressed in gibibytes.

T, t The size is expressed in tebibytes.

默认size是不限制的

malloc的性能依赖于内存速度，所以很快。如果数据集大于可用内存，malloc的性能将依赖操作系统的实际分页能力。

## file

syntax: file[,path[,size[,granularity]]]

file后端使用mmap通过非软连（unlinked）的磁盘文件来存储内存中的的对象。

path参数指定存放cache对象的文件（backing file）路径或者是varnishd将创建存放cache文件的目录，默认是/tmp.

size参数指定存放cache文件的大小。size的假定单位是byte，除非你指定下面的一个后缀：

K, k The size is expressed in kibibytes.

M, m The size is expressed in mebibytes.

G, g The size is expressed in gibibytes.

T, t The size is expressed in tebibytes.

% The size is expressed as a percentage of the free space on the file system where it resides.

    % 表示varnish安装文件系统中，varnish可以用空闲空间百分比

默认size是使用50%的设备可用空间。 如果backing文件存在，它将缩小或者扩张这个文件到指定的尺寸。

注意，如果是varnishd必须创建或者扩充这个文件，它将不预分配增加的空间，导致分裂，这可能对旋转的硬盘驱动器产生不利影响。预创建存储文件使用dd将减小分裂到最小化。

granularity参数指定分配的粒度。所有的粒度构成size。粒度假定用byte表示，除非添加了描述size的后缀除了%。

默认的granularity 是VM page的尺寸。如果你有大量的小对象，你需要减小VM page的尺寸。

file的性能通常受限于设备的读写速度、使用、查找时间。

## persistent (experimental)持久化（试验性的）

syntax: persistent,path,size {experimental}

持久化存储。varnish将存储对象到文件中，保护尽可能多的对象不受计划或非计划的关闭varnish。

path参数指定backfile的path。如果文件不存在，varnish将创建它。

size参数指定存放cache文件的大小。size的假定单位是byte，除非你指定下面的一个后缀：

K, k The size is expressed in kibibytes.

M, m The size is expressed in mebibytes.

G, g The size is expressed in gibibytes.

T, t The size is expressed in tebibytes.

vanish将分割文件到逻辑silos(仓库)，然后以环形buffer的方式写这个silos。任意时间点只有一个silo是保持打开的状态。所有的silos是被密封的。当varnish启动或者关闭，varnish将从哪些不是封闭的silos中清除内容。

注意脱机使用持久化的同时使用bans 会引发一些问题。这是由于在silo离线期间所做的ban操作，在缓存的对象再次进入cache时将不被应用到新silo。因此，已经被ban掉的对象将再次进入cache。

## Transient Storage

如果你指定你的后端为Transient，这对暂存（short lived）对象是有用的。默认varnish可以使用不受限的malloc后端来完成。

如果TTL是标示为shortlived，varnish将认为该对象short lived（Transient ）

# Parameters

Varnish cache 拥有一个参数的集合，可以影响varnish的行为和性能。基本上所有的参数都可以在Varnish命令行接口（varnishadm）使用param.set关键词来设置。

一些参数，处于安全的目的只读的参数，只能使用-r参数来开启。同时在varnishd的man手册中也有相关的说明

-r param[,param...]

  Make  the listed parameters read only.

  This gives the system administrator a way to limit what the Varnish CLI can do.

  Consider making parameters such as user, group, cc\_command, vcc\_allow\_inline\_c

  read only as these can potentially  be used  to  escalate

  privileges from the CLI.  Protecting listen\_address may also be a good idea.

我们不建议你调整参数除非你确定你做的是什么。我们努力的使用稳定的默认值，同时varnish使用默认设置应当能处理大多数的工作情况。

完整的参数列表可以通过在命令行使用param.show来查看。

varnish> param.show

200

acceptor\_sleep\_decay       0.9 (default)

acceptor\_sleep\_incr        0.001 [seconds] (default)

acceptor\_sleep\_max         0.050 [seconds] (default)

auto\_restart               on [bool] (default)

ban\_dups                   on [bool] (default)

ban\_lurker\_age             60.000 [seconds] (default)

ban\_lurker\_batch           1000 (default)

ban\_lurker\_sleep           0.010 [seconds] (default)

between\_bytes\_timeout      60.000 [seconds] (default)

busyobj\_worker\_cache       off [bool] (default)

cc\_command                 "exec gcc -std=gnu99  -O2 -g -Wp,-D\_FORTIFY\_SOURCE=0 -Wall -Werror -pthread -fpic -shared -Wl,-x -o %o %s" (default)

cli\_buffer                 8k [bytes] (default)

cli\_limit                  48k [bytes] (default)

cli\_timeout                60.000 [seconds] (default)

clock\_skew                 10 [seconds] (default)

connect\_timeout            3.500 [seconds] (default)

critbit\_cooloff            180.000 [seconds] (default)

debug                      none (default)

default\_grace              10.000 [seconds] (default)

default\_keep               0.000 [seconds] (default)

default\_ttl                120.000 [seconds] (default)

feature                    none (default)

fetch\_chunksize            16k [bytes] (default)

fetch\_maxchunksize         0.25G [bytes] (default)

first\_byte\_timeout         60.000 [seconds] (default)

group                      GID 0 (default)

group\_cc                   <not set> (default)

gzip\_buffer                32k [bytes] (default)

gzip\_level                 6 (default)

gzip\_memlevel              8 (default)

http\_gzip\_support          on [bool] (default)

http\_max\_hdr               64 [header lines] (default)

http\_range\_support         on [bool] (default)

http\_req\_hdr\_len           8k [bytes] (default)

http\_req\_size              32k [bytes] (default)

http\_resp\_hdr\_len          8k [bytes] (default)

http\_resp\_size             32k [bytes] (default)

idle\_send\_timeout          60.000 [seconds] (default)

listen\_address             :80 (default)

listen\_depth               1024 [connections] (default)

lru\_interval               2.000 [seconds] (default)

max\_esi\_depth              5 [levels] (default)

max\_restarts               4 [restarts] (default)

max\_retries                4 [retries] (default)

nuke\_limit                 50 [allocations] (default)

pcre\_match\_limit           10000 (default)

pcre\_match\_limit\_recursion 10000 (default)

ping\_interval              3 [seconds] (default)

pipe\_timeout               60.000 [seconds] (default)

pool\_req                   10,100,10 (default)

pool\_sess                  10,100,10 (default)

pool\_vbc                   10,100,10 (default)

pool\_vbo                   10,100,10 (default)

prefer\_ipv6                off [bool] (default)

rush\_exponent              3 [requests per request] (default)

send\_timeout               600.000 [seconds] (default)

session\_max                100000 [sessions] (default)

shm\_reclen                 255b [bytes] (default)

shortlived                 10.000 [seconds] (default)

sigsegv\_handler            off [bool] (default)

syslog\_cli\_traffic         on [bool] (default)

tcp\_keepalive\_intvl        75.000 [seconds] (default)

tcp\_keepalive\_probes       9 [probes] (default)

tcp\_keepalive\_time         7200.000 [seconds] (default)

thread\_pool\_add\_delay      0.000 [seconds] (default)

thread\_pool\_destroy\_delay  1.000 [seconds] (default)

thread\_pool\_fail\_delay     0.200 [seconds] (default)

thread\_pool\_max            1000 [threads]

thread\_pool\_min            50 [threads]

thread\_pool\_stack          48k [bytes] (default)

thread\_pool\_timeout        120.000 [seconds]

thread\_pools               2 [pools] (default)

thread\_queue\_limit         20 (default)

thread\_stats\_rate          10 [requests] (default)

timeout\_idle               5.000 [seconds] (default)

timeout\_linger             0.050 [seconds] (default)

timeout\_req                2.000 [seconds] (default)

user                       nobody (99) (default)

vcc\_allow\_inline\_c         off [bool] (default)

vcc\_err\_unref              on [bool] (default)

vcc\_unsafe\_path            on [bool] (default)

vcl\_dir                    /etc/varnish (default)

vmod\_dir                   /usr/lib64/varnish/vmods (default)

vsl\_buffer                 4k [bytes] (default)

vsl\_mask                   -VCL\_trace,-WorkThread,-Hash (default)

vsl\_reclen                 255b [bytes] (default)

vsl\_space                  80M [bytes] (default)

vsm\_space                  1M [bytes] (default)

waiter                     epoll (possible values: epoll, poll) (default)

workspace\_backend          64k [bytes] (default)

workspace\_client           64k [bytes] (default)

workspace\_session          384b [bytes] (default)

workspace\_thread           2k [bytes] (default)

为了检查某一的参数和获取参数的用法、描述、默认值，通过键入param.show 和参数名，像这样：

varnish> param.show shortlived

200

shortlived                  10.000000 [s]

                            Default is 10.0

                            Objects created with TTL shorter than this are

                            always put in transient storage.

varnish> param.show thread\_pools

200

thread\_pools

        Value is: 2 [pools] (default)

        Default is: 2

        Minimum is: 1

        Number of worker thread pools.

        Increasing number of worker pools decreases lock contention.

        Too many pools waste CPU and RAM resources, and more than one

        pool for each CPU is probably detrimal to performance.

        Can be increased on the fly, but decreases require a restart to

        take effect.

        NB: This parameter may take quite some time to take (full)

        effect.

        NB: We do not know yet if it is a good idea to change this

        parameter, or if the default value is even sensible.  Caution

        is advised, and feedback is most welcome.

# VCL - Varnish Configuration Language

本节包含了怎样通过varnish配置语言vcl来通知varnish如何处理http请求。

varnish是高度结构化的系统。大多数其他的系统使用配置指令，在那里你基本上打开和关闭大量的开关。而我们选择使用领域特定语言，被叫做VCL，来完成之前的功能。

每个进入的请求流通过varnish，同时你可以支配如何通过VCL代码来处理请求。你可以让某些后端直接处理请求，你可以改变请求和响应，或者让varnish拥有各种各样动作依赖于特定的处理请求和响应的阶段。这些特性让Varnish变成特别强力的http程序，而不仅仅是缓存。

varnish转换VCL成二进制代码，当请求到达这些代码将被执行。VCL对varnish的性能损耗是可以忽略不计的。

VCL文件将子程序组织在一起。不同的子程序在不同阶段被执行。一个是在我们收到请求时执行，其他的当从后端收到文件时执行。

如果你在你的子程序不调用某个动作，varnish将执行一些built-in（内建）的VCL代码。你可以看到这些内建的代码在builtin.vcl中注释的部分

本节包含以下：

VCL Syntax

Built in subroutines

Request and response VCL objects

actions

Backend servers

Multiple backends

Backends and virtual hosts in Varnish

Directors

Health checks

Hashing

Misbehaving servers

Using inline C to extend Varnish

VCL Examples

Device detection

# VCL Syntax

varnish从C继承了很多，同时它读起来像简单的C或者是Perl。

块由大括号分隔，语句用分号结束，注释可以根据自己的喜好来写为C，C ++或Perl的注释。

注意VCL不包含任何循环和跳转语句。

本节提供大致最重要部分的语法。完整的VCL语法文档请查看<https://www.varnish-cache.org/docs/4.0/reference/vcl.html#reference-vcl>

## Strings

基本的字符是被双引号引起来，像"..."，同时不可以跨行。 反斜杠不是特别的，例如regsub你不需要对\进行转义：

regsub("barf", "(b)(a)(r)(f)", "\4\3\2p") -> "frap"

长字符应该被{"..."}。它们可以包含任何字符包括"、跨行、除了NUL（0×00）字符等控制字符。如果你真的想字符串中使用NUL字符，VMOD，使得它可以创建这样的字符串。

## Access control lists (ACLs)

ACL申明创建和初始化一个权限控制列表，通常用来匹配客户端地址：

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ACL条目指定了varnish无法解析的主机名，它会匹配任何和它相比的地址。因此，如果它前面有一个否定标记，它将拒绝任何和它相比的地址，这可能和你想的不一致。如果该条目括号的，它将简单地被忽略。

匹配ip地址，使用VCL，简单的使用匹配操作：

if (client.ip ~ local) {

    return (pipe);

}

## Operators

下面这些操作时可以在VCL中使用的，查看例子来进一步了解这些操作。

=

Assignment operator.赋值操作

==

Comparison.比较

~

Match. Can either be used with regular expressions or ACLs.匹配操作，可以使用正则或者ACLs

!

Negation.非

&&

Logical and 逻辑与

||

Logical or  逻辑或

## Subroutines

子程序可以让一段代码易读的同时可以重用代码。

sub pipe\_if\_local {

  if (client.ip ~ local) {

    return (pipe);

  }

}

在VCL的子程序，不能引入参数，也不会返回值

调用子程序，使用call关键字，后面跟上被调用的子程序的名字：

call pipe\_if\_local;

varnish已经有了一些内建的子程序，当请求流通过varnish时将会调用这些内建的子程序。这些所有的内建子程序被命名成vcl.\* 你自己编写的子程序不能以vcl开始。

# Built in subroutines

## vcl\_recv

在接收到完整的客户端请求后调用这个子程序。它决定是否处理请求，怎样处理请求，使用哪一个后端。

它也可运用于修改请求，一些经常自己做的事情。

vcl\_recv子程序可以通过调用return()来结束，通过以下关键字：

synth(status code, reason)

    Return a synthetic object with the specified status code to the client and abandon the request.

    返回一个人为对象使用指定的状态码给客户端或者放弃请求。

pass

    Switch to pass mode. Control will eventually pass to vcl\_pass.

    处理跳转到vcl\_pass

pipe

    Switch to pipe mode. Control will eventually pass to vcl\_pipe.

    处理跳转到vcl\_pipe

hash

    Continue processing the object as a potential candidate for caching. Passes the control over to vcl\_hash.

    处理跳转到vcl\_hash，在缓存中查找对象，如果有则为命中，

purge

    Purge the object and it's variants. Control passes through vcl\_hash to vcl\_purge.

    删除对象和它的变体。控制pass通过vcl\_hash到vcl\_purge

## vcl\_pipe

在进入pipe模式时被调用。将请求直接传递至后端主机，并将后端响应原样返回客户端

vcl\_pipe子程序可以通过调用return()来结束，通过以下关键字：

synth(status code, reason)

    Return a synthetic object with the specified status code to the client and abandon the request.

    返回一个人为对象使用指定的状态码给客户端或者放弃请求。

pipe

    Proceed with pipe mode.

    使用pip模式处理

## vcl\_pass

此函数在进入pass模式时被调用，用于将请求直接传递至后端主机，但后端主机的响应并不缓存直接返回客户端；

vcl\_pass子程序可以通过调用return()来结束，通过以下关键字：

synth(status code, reason)

    Return a synthetic object with the specified status code to the client and abandon the request.

    返回一个人为对象使用指定的状态码给客户端或者放弃请求。

fetch

    Proceed with pass mode.

    继续pass模式

restart

    Restart the transaction. Increases the restart counter. If the number of restarts is higher than max\_restarts Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

## vcl\_hit

在执行 lookup 指令后，在缓存中找到请求的内容后将自动调用该函数；

vcl\_hit子程序可以通过调用return()来结束，通过以下关键字：

restart

    Restart the transaction. Increases the restart counter. If the number of restarts is higher than max\_restarts Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

deliver

    Deliver the object. Control passes to vcl\_deliver.

    传递对象，使用vcl\_deliver处理。

synth(status code, reason)

    Return a synthetic object with the specified status code to the client and abandon the request.

    返回一个人为对象使用指定的状态码给客户端或者放弃请求。

## vcl\_miss

在执行 lookup 指令后，在缓存中没有找到请求的内容时自动调用该方法，此函数可用于判断是否需要从后端服务器获取内容，从哪一个后端获取内容；

vcl\_miss子程序可以通过调用return()来结束，通过以下关键字：

synth(status code, reason)

    Return a synthetic object with the specified status code to the client and abandon the request.

    返回一个人为对象使用指定的状态码给客户端或者放弃请求。

pass

    Switch to pass mode. Control will eventually pass to vcl\_pass.

    处理跳转到vcl\_pass

fetch

    Retrieve the requested object from the backend. Control will eventually pass to vcl\_backend\_fetch.

    从后端获取请求对象。控制最终传递到vcl\_backend\_fetch。

restart

    Restart the transaction. Increases the restart counter. If the number of restarts is higher than max\_restarts Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

## vcl\_hash

在vcl\_recv调用后为请求创建一个hash值时，调用此函数；此hash值将作为varnish中搜索缓存对象的key；

vcl\_hash子程序可以通过调用return()来结束，通过以下关键字：

lookup

    Look up the object in cache. Control passes to vcl\_miss, vcl\_hit or vcl\_purge.

    从缓存中查找对象。控制传递至vcl\_miss, vcl\_hit 或者 vcl\_purge.

## vcl\_purge

pruge操作执行后调用此函数，所有他的变种将被回避；

vcl\_purge子程序可以通过调用return()来结束，通过以下关键字：

synth(status code, reason)

    Return a synthetic object with the specified status code to the client and abandon the request.

    返回一个人为对象使用指定的状态码给客户端或者放弃请求。

restart

    Restart the transaction. Increases the restart counter. If the number of restarts is higher than max\_restarts Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

## vcl\_deliver

将在缓存中找到请求的内容发送给客户端前调用此方法；

vcl\_deliver子程序可以通过调用return()来结束，通过以下关键字：

deliver

    Deliver the object to the client.

    传递对象给客户端。

restart

    Restart the transaction. Increases the restart counter. If the number of restarts is higher than max\_restarts Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

## vclbackendfetch

向后端主机发送请求前，调用此函数，可修改发往后端的请求；

vclbackendfetch子程序可以通过调用return()来结束，通过以下关键字：

fetch

    Fetch the object from the backend.

    从后端获取对象

abandon

    Abandon the backend request and generates an error.

    丢弃后端请求，同时生成一个错误页面。

## vclbackendresponse

获得后端主机的响应后，可调用此函数；

vclbackendresponse子程序可以通过调用return()来结束，通过以下关键字：

deliver

    Possibly insert the object into the cache, then deliver it to the Control will eventually pass to vcl\_deliver.

    可以将对象插入到缓存，然后传递它到控制器，最终传递到vcl\_deliver。

abandon

    Abandon the backend request and generates an error.

    丢弃后端请求，同时生成一个错误页面。

retry

    Retry the backend transaction. Increases the retries counter. If the number of retries is higher than max\_retries Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

## vclbackenderror

当从后端主机获取源文件失败时，调用此函数；

vclbackenderror子程序可以通过调用return()来结束，通过以下关键字：

deliver

    Deliver the error.

    传递错误页面

retry

    Retry the backend transaction. Increases the retries counter. If the number of retries is higher than max\_retries Varnish emits a guru meditation error.

    重新启动事务。增加了重新启动计数器。如果重新启动的次数比max\_restarts高,varnish将返回权威的默认错误。

## vcl\_synth

调用是返回一个定义的的对象，定义的对象在VCL中生成，不是从后端获取。典型就是使用synthetic()函数；

vcl\_synth子程序可以通过调用return()来结束，通过以下关键字：

deliver

    Deliver the object. If the object has a positive TTL then the object is also stored in cache.

    传递对象，如果对象设置了TTL，该对象还会存储如缓存中。

restart

    Restart processing the object.

    重新启动对对象的处理。

## vcl\_init

VCL加载时调用此函数，经常用于初始化varnish模块(VMODs)；

vcl\_init子程序可以通过调用return()来结束，通过以下关键字：

ok

    Normal return, VCL continues loading.

    正常返回，VCL继续加载。

## vcl\_fini

当所有请求都离开当前VCL，且当前VCL被弃用时，调用此函数，经常用于清理varnish模块；

vcl\_init子程序可以通过调用return()来结束，通过以下关键字：

ok

    Normal return, VCL will be discarded.

    正常返回，VCL将不再使用。

# Request and response VCL objects

VCL中有些需要你注意的重要对象。这些对象可以在VCL被使用和操作。

req

The request object. When Varnish has received the request the req object is created and populated. Most of the work you do in vcl\_recv you do on or with the req    object.

请求对象。当vanish接收到请求后，req对象被创建和生成。你可以在vcl\_recv中使用req对象做很多事。

bereq

The backend request object. Varnish contructs this before sending it to the backend. It is based on the req object.

后端请求对象。varnish在发送请求到后端之前构建这个对象。它基于req对象。

beresp

The backend response object. It contains the headers of the object coming from the backend. If you want to modify the response coming from the server you modify    this object in vcl\_backend\_response.

后端响应对象。它包含在从后端响应对象的头里。如果你想修改后端server返回的响应信息，你可以在vcl\_backend\_response中修改beresp对象。

resp

The HTTP response right before it is delivered to the client. It is typically modified in vcl\_deliver.

传递给客户端响应之前的response对象。通常在vcl\_deliver中修改。

obj

The object as it is stored in cache. Read only.

存储在缓存中的对象。 只读。

# actions

最常见的动作return这些：

pass

When you return pass the request and subsequent response will be passed to and from the backend server. It won't be cached. pass can be returned from vcl\_recv.

当你在子程序中return(pass)请求和随后的响应将被传递到后端server和从后端server回传回来。响应将不会被缓存。pass可以从vcl\_recv中返回。

hash

When you return hash from vcl\_recv you tell Varnish to deliver content from cache even if the request othervise indicates that the request should be passed.

在\*vcl\_recv\*中return(hash)，通知varnish从cache查找请求内容，除非这个请求不被标示，那么请求应当被pass。

pipe .. XXX:What is pipe? benc

Pipe can be returned from vcl\_recv as well. Pipe short circuits the client and the backend connections and Varnish will just sit there and shuffle bytes back and   forth. Varnish will not look at the data being send back and forth - so your logs will be incomplete.

pipe也可以在\*vcl\_recv\*中返回，return(pipe)。

deliver

Deliver the object to the client. Usually returned from vcl\_backend\_response.

传递对象给客户端。通常在vcl\_backend\_response中return。

restart

Restart processing of the request. You can restart the processing of the whole transaction. Changes to the req object are retained.

重新对请求进行处理。你可以在整个请求处理的阶段重启。更改过的req对象将被保留。

retry

Retry the request against the backend. This can be returned from vcl\_backend\_response or vcl\_backend\_error if you don't like the response that the backend delivered    .

重启指向后端的请求。如果你不想从后端获得响应，你可以在vcl\_backend\_response 或者vcl\_backend\_error 中return。

# Backend servers

varnish有"后端"或者"源"服务器的概念。backend server提供给varnish加速的内容。

第一步设置是告诉varnish从哪儿找到backend server。使用你喜欢的编辑器打开varnishd引入的VCL文件。

VCL文件的开头有一小段有点像这样：

# backend default {

#     .host = "127.0.0.1";

#     .port = "8080";

# }

去掉注释后

backend default {

    .host = "127.0.0.1";

    .port = "8080";

}

这样一段配置在varnish中定义了一个后端，被叫做default。（和c的函数定义有点相似），当varnish需要从后端获得内容时，它将连接127.0.0.1的8080端口。

varnish可以定义多个后端，也可以将几个后端放在一个后端集群里面已达到负载均衡的目的。

# Multiple backends

某些情况下你可能需要让varnish缓存多个后端的内容。你可能想让varnish映射所有的url在单个后端上，或者是多个后端。这里有些参数可以满足这样的需求。

现在我们需要在PHP站点中引入java应用。java应用的链接都是以/java/开头的。 处理java应用的服务器监听在8000端口上。默认的default.vcl文件这样的：

backend default {

    .host = "127.0.0.1";

    .port = "8080";

}

增加一个后端：

backend java {

    .host = "127.0.0.1";

    .port = "8000";

}

现在需要告诉varnish将不同的url发送到不同的后端server。在vcl\_recv中：

sub vcl\_recv {

    if (req.url ~ "^/java/") {

        set req.backend\_hint = java;

    } else {

        set req.backend\_hint = default;

    }

}

真的很简单。停下来思考下。正如你所看到的，你可以定义如何根据真实的任意数据来选择后端。你想发送手机设备的请求到不同的后端？没有问题。 可以满足你的需求。

if (req.http.User-Agent ~ "(?i)MIDP|WAP|UP.Browser|Smartphone|Obigo|Mobile|AU.Browser|wxd.Mms|WxdB.Browser|CLDC|UP.Link|KM.Browser|UCWEB|SEMC\-Browser|Mini|Symbian|Palm|Nokia|Panasonic|MOT|SonyEricsson|NEC|Alcatel|Ericsson|BENQ|BenQ|Amoisonic|Amoi|Capitel|PHILIPS|SAMSUNG|Lenovo|Mitsu|Motorola|SHARP|WAPPER|LG|EG900|CECT|Compal|kejian|Bird|BIRD|G900/V1.0|Arima|CTL|TDG|Daxian|DAXIAN|DBTEL|Eastcom|EASTCOM|PANTECH|Dopod|Haier|HAIER|KONKA|KEJIAN|LENOVO|Soutec|SOUTEC|SAGEM|SEC|SED|EMOL|INNO55|ZTE|iPhone|Android|Windows CE|Wget|Java|Opera") {

    set req.backend\_hint = mobile;

}

# Backends and virtual hosts in Varnish

varnish完全支持虚拟主机。他们从来没有明确宣布这点是因为varnish以非常规的方式来实现这个功能。在vcl\_recv我们设置处理HTTP请求的路由。如果你想让路由基于基本的虚拟主机做点什么，你只需要检查req.http.host便可。

sub vcl\_recv {

    if (req.http.host ~ "foo.com") {

        set req.backend\_hint = foo;

    } elsif (req.http.host ~ "bar.com") {

        set req.backend\_hint = bar;

    }

}

~符号后面的理解为正则表达式。这里可以匹配 "foo.com", "www.foo.com", "zoop.foo.com.cn" 还有其他包含foo.com的域名。该实例是有意写作如此，如果你明确了你的域名，你需要使用==来替代~符号。

sub vcl\_recv {

    if (req.http.host == "foo.com" || req.http.host == "www.foo.com") {

        set req.backend\_hint = foo;

    }

}

# Directors

你也可以将几个后端组成一组后端。这个组被叫做Directors。可以提高性能和弹性（增加一台app，或者停用一台app等等）。

你可以定义几个后端，然后将他们组织在一个Directors。这些操作需要你载入VMOD（varnish module），然后在vcl\_init中调用这个VMOD。

import directors;    # load the directors

backend server1 {

    .host = "192.168.0.10";

}

backend server2 {

    .host = "192.168.0.10";

}

sub vcl\_init {

    new bar = directors.round\_robin();

    bar.add\_backend(server1);

    bar.add\_backend(server2);

}

sub vcl\_recv {

    # send all traffic to the bar director:

    set req.backend\_hint = bar.backend();

}

该调度器是round-robin调度器。这意味着Directors将以轮询的算法来分发请求。还有一个random分配请求的director，你猜对了，随机的方式。

但是如果其中的某个server宕机了呢？varnish能将所有的请求分发到健康的server么？当然！ Health Checks就是干这事的！

# Health checks

现在我们给一个调度器设置两个后端和健康检测。先定义后端：

backend server1 {

    .host = "server1.example.com";

    .probe = {

        .url = "/";

        .timeout = 1s;

        .interval = 5s;

        .window = 5;

        .threshold = 3;

    }

}

backend server2 {

    .host = "server2.example.com";

    .probe = {

        .url = "/";

        .timeout = 1s;

        .interval = 5s;

        .window = 5;

        .threshold = 3;

    }

}

.probe是新的参数。在上面的例子中varnish将每5s检测后端，超时设为1s。每个检测将会发送get /的请求。如果5个检测中大于3个是成功，varnish就认为后端是健康的，反之，后端就有问题了。

更多.probe的信息可以在varnish中查看[Probes](https://www.varnish-cache.org/docs/4.0/reference/vcl.html#reference-vcl-probes)章节

我们现在定义director

import directors;

sub vcl\_init {

    new vdir = directors.round\_robin();

    vdir.add\_backend(server1);

    vdir.add\_backend(server2);

}

你使用vir director作为处理请求的后端，就像你使用一个简单的后端那样。Varnish不会将请求发送到标记为不健康的后端上。

如果所有的后端都宕机了，varnish也可以返回给用户过期了的内容。查看[Misbehaving servers](https://www.varnish-cache.org/docs/4.0/users-guide/vcl-grace.html#users-guide-handling-misbehaving-servers)获得更多如何开启这个功能的信息。

请注意，varnish将保持健康运行的探针对所有加载VCLs。Varnish将探针合并成似乎完全一体的-因此如果你有很多载入的VCL，不要修改probe的配置，卸载VCL将禁用probes。更多信息请查看ref:reference-vcl-director.

# Hashing

实质上，当Varnish存储内容在缓存中时，它也把如何发现这个对象的哈希键和对象一起存储。默认设置的hash键是基于内容的的主机名或者ip地址和URL计算出的。（url+host或者url+ip）

默认的VCL：

sub vcl\_hash {

    hash\_data(req.url);

    if (req.http.host) {

        hash\_data(req.http.host);

    } else {

        hash\_data(server.ip);

    }

    return (lookup);

}

正如你看见那样，首先检查req.url然后如果req.http.host存在的话检查req.http.host。

需要指出的是在哈希之前varnish不会将hostname或者URL转换成小写，因此“Varnish.org/”和“varnish.org/”理论会导致不同的缓存条目。然而，浏览器，往往小写主机名。

你可以修改hash。这种方式你可以让Varnish根据任意算法提供不同的内容给不同的客户端。

你如果想基于源ip地址提供不同语言的页面给你的用户。你需要一些[根据ip地址判断是哪个国家的VMOD](https://github.com/lampeh/libvmod-geoip)，然后放在哈希中。看起来像这样：

In vcl\_recv:

set req.http.X-Country-Code = geoip.lookup(client.ip);

然后在vcl\_hash中增加：

sub vcl\_hash {

    hash\_data(req.http.X-Country-Code);

}

默认的VCL将增加host和url到hash，尽管我们没有做任何事情。当心调用return(lookup),这将中止默认VCL的执行,同时varnish可以在结束时基于随机输入的或多或少的数据。

**Misbehaving servers**

varnish有个关键特性，为misbehaving（行为不端，诡异的）web服务器或者应用服务器提供保护的能力。

**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，然后让其他几个请求挂起并等待返回结果；获得结果后，其它请求再复制后端的结果发送给客户端。有些产品中需要调用其他来将请求合并，而varnish自动做这些。

但如果同时有数以千计的请求，那么这个等待队列将变得庞大，这将导致2类潜在问题：

* 惊群问题(thundering herd problem)，即突然释放大量的线程去复制后端返回的结果，将导致负载急速上升；
* 没有用户喜欢等待；

为了解决这类问题，可以配置varnish在缓存对象因超时失效后再保留一段时间，以给那些等待的请求返回过期的内容(stale content)。

为了提供给用户过期的内容，我们必须先有这些内容。因此我们在VCL中配置如下，使得varnish能在内容过期过后依然保持2分钟：

sub vcl\_backend\_response {

    set beresp.grace = 2m;

}

现在Varnish允许在对象过期后2分钟内提供给客户端。同时varnish也将刷新这个对象。刷新动作是异步发生的，发生在新的对象将替换老对象的同时。

我们可以在vcl\_hit中增加代码来影响这个逻辑的工作。默认VCL是这样：

sub vcl\_hit {

    if (obj.ttl >= 0s) {

        // A pure unadultered hit, deliver it

        return (deliver);

    }

    if (obj.ttl + obj.grace > 0s) {

        // Object is in grace, deliver it

        // Automatically triggers a background fetch

        return (deliver);

    }

    // fetch & deliver once we get the result

    return (fetch);

}

如果你开启了健康检测，要是后端有问题只能提供给客户端过期的对象时，你就能检测到。用如下的if语句替换掉上面的第二个if语句块：

if (!std.healthy(req.backend\_hint) && (obj.ttl + obj.grace > 0s)) {

    return (deliver);

} else {

    return (fetch);

}

总之，优雅模式解决下面2个问题：

* 提供过期内容给客户端以避免请求堆积。
* 如果你允许varnish可以提供过期的内容。

**Using inline C to extend Varnish**

你可以使用*inline C*来扩展Varnish。注意，这种方式您可能严重损毁varnish。C代码运行在Varnish缓存进程中，如果你的C代码有错误，缓存进程将crash。

一个使用inline C的功能是记录日志到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

}

要使用inline C你需要通过[*vcc\_allow\_inline\_c*](https://www.varnish-cache.org/docs/4.0/reference/varnishd.html?highlight=vcc_allow_inline_c)参数开启此功能。

# VCL Examples

## Manipulating request headers in VCL

在VCL中操作请求的header。

当我们想移除发送到web服务器/images目录的所有对象的cookie可以这样：

sub vcl\_recv {

    if (req.url ~ "^/images") {

    unset req.http.cookie;

    }

}

这样当请求被后端处理时在http的header中将不再有cookie信息。有用的是使用if语句。在匹配的URL中，也支持正则表达式。注意匹配操作。

## Altering the backend response

修改后端响应。

如果匹配到当前规则我们可以重新设置从后端响应的对象的TTL：

sub vcl\_backend\_response {

    if (bereq.url ~ "\.(png|gif|jpg)$") {

        unset beresp.http.set-cookie;

        set beresp.ttl = 1h;

    }

}

我们也移除http头中的Set-Cookie，是为了避免hit-for-pass对象被创建。查看[actions](http://my.oschina.net/monkeyzhu/blog/467267)。

## ACLs

你可以使用acl关键字创建一个权限控制列表。你可以让客户端的ip地址匹配到acl列表，使用匹配操作：

# 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.method == "PURGE") {

        if (client.ip ~ local) {

            return(purge);

        } else {

            return(synth(403, "Access denied."));

        }

    }

}

## Implementing websocket support

实现WebSocket的支持

WebSockets是一种用于创建基于HTTP的双向基于流的通道的技术。为了通过Varnish运行WebSockets你需要用管道传递它，并复制Upgrade头。使用下面的VCL来实现：

sub vcl\_pipe {

    if (req.http.upgrade) {

        set bereq.http.upgrade = req.http.upgrade;

    }

}

sub vcl\_recv {

    if (req.http.Upgrade ~ "(?i)websocket") {

        return (pipe);

    }

}

# Device detection

设备检测基于请求中的User-agent找出哪种内容返回给客户端。

使用Device detection情况的一个例子，发送给小屏幕的手机客户端或者复杂的网络环境，减少发送文件的数量，或者提供给不能解码的客户端一个video解码器。

有些使用这种方案的典型场景：

* 重写url
* 使用不同的后端给客户端
* 改变后端请求为了让后端发送裁剪的内容

可能为了便于理解，下文中假定req.http.X-UA-Device头表示当前前客户端，同时每种客户端是唯一的。

简单的像这样设置头：

sub vcl\_recv {

    if (req.http.User-Agent ~ "(?i)iphone" {

        set req.http.X-UA-Device = "mobile-iphone";

    }

}

不同的处理请求，同时免费提供组织和验证更详细的的客户端。基本且通用的是使用正则表达式来设置。查看<https://github.com/varnish/varnish-devicedetect/>。

## Serve the different content on the same URL

这个戏法表明：

* 验证客户端（相当简单，只需要包含devicedetect.vcl，然后调用它）
* 根据客户端的类型找出怎样发送请求到后端。这包含了些例子，设置header，改变header，甚至改变发送到后端的url。
* 修改后端响应，增加miss标志'Vary'头。
* 修改发送到客户端的输出，这样如果任何cache不受控，可以不提供错误的内容给客户端。

所有这些都需要确定我们每种设备只会获得一个缓存的对象。

### Example 1: Send HTTP header to backend

基本情况是Varnish增加X-UA-Device头在发送到后端的请求上，然后后端在响应header中添加Vary头，响应的内容依赖发送到后端的header。

从Varnish角度看一切开箱即用。

VCL：

sub vcl\_recv {

    # call some detection engine that set req.http.X-UA-Device

    # 调用一些检测机来设置req.http.X-UA-Device。

}

    # req.http.X-UA-Device is copied by Varnish into bereq.http.X-UA-Device

    # req.http.X-UA-Device 被复制到bereq.http.X-UA-Device，当请求传递到后端时。

    # so, this is a bit counterintuitive. The backend creates content based on

    # the normalized User-Agent, but we use Vary on X-UA-Device so Varnish will

    # use the same cached object for all U-As that map to the same X-UA-Device.

    #

    # 因此这是有点反直觉的。后端创建响应内容基于正常的User-Agent，但是在X-UA-Device使用

    # Vary，varnish将使用相同的已缓存的对象，对所有的U-As映射到相同的X-UA-Device。

    # If the backend does not mention in Vary that it has crafted special

    # content based on the User-Agent (==X-UA-Device), add it.

    # 如果基于User-Agent的后端响应不带有Vary，在header种增加它。

    # If your backend does set Vary: User-Agent, you may have to remove that here.

    # 如果后端设置了Vary：User-Agent，你可以在此删除它，或者设置成其他的。

sub vcl\_backend\_response {

    if (bereq.http.X-UA-Device) {

        if (!beresp.http.Vary) { # no Vary at all

            set beresp.http.Vary = "X-UA-Device";

        } elseif (beresp.http.Vary !~ "X-UA-Device") { # add to existing Vary

            set beresp.http.Vary = beresp.http.Vary + ", X-UA-Device";

        }

    }

    # comment this out if you don't want the client to know your

    # classification

    # 如果不想客户端知道你的分类，注释掉即可。

    set beresp.http.X-UA-Device = bereq.http.X-UA-Device;

}

# to keep any caches in the wild from serving wrong content to client #2

# behind them, we need to transform the Vary on the way out.

sub vcl\_deliver {

    if ((req.http.X-UA-Device) && (resp.http.Vary)) {

        set resp.http.Vary = regsub(resp.http.Vary, "X-UA-Device", "User-Agent");

        # 在resp.http.Vary的值中将X-UA-Device替换为User-Agent。

    }

}

### Example 2: Normalize the User-Agent string

标准化User-Agent字符。

另一种发送设备类型的目的是为了重写或者标准化发送到后端的User-Agent。

例如：

User-Agent: Mozilla/5.0 (Linux; U; Android 2.2; nb-no; HTC Desire Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1

变成：

User-Agent: mobile-android

如果你在后端上不需要源http header的任何信息可以这样设置。一种可能性就是使用CGI脚本，只需小小设置下预定义的头信息在可以用的脚本中。

VCL：

sub vcl\_recv {

    # call some detection engine that set req.http.X-UA-Device

}

# override the header before it is sent to the backend

# 发送到后端之前重写header。

sub vcl\_miss { if (req.http.X-UA-Device) { set bereq.http.User-Agent = req.http.X-UA-Device; } }

sub vcl\_pass { if (req.http.X-UA-Device) { set bereq.http.User-Agent = req.http.X-UA-Device; } }

# standard Vary handling code from previous（之前的） examples.

sub vcl\_backend\_response {

    if (bereq.http.X-UA-Device) {

        if (!beresp.http.Vary) { # no Vary at all

            set beresp.http.Vary = "X-UA-Device";

        } elseif (beresp.http.Vary !~ "X-UA-Device") { # add to existing Vary

            set beresp.http.Vary = beresp.http.Vary + ", X-UA-Device";

        }

    }

    set beresp.http.X-UA-Device = bereq.http.X-UA-Device;

}

sub vcl\_deliver {

    if ((req.http.X-UA-Device) && (resp.http.Vary)) {

        set resp.http.Vary = regsub(resp.http.Vary, "X-UA-Device", "User-Agent");

    }

}

### Example 3: Add the device class as a GET query parameter

以GET参数作为设备类型

上面两种方式都不行的话你可使用GET参数来增加设备类型。

http://example.com/article/1234.html --> http://example.com/article/1234.html?devicetype=mobile-iphone

客户端不会查看分类，仅仅是后端请求被修改了。

VCL：

sub vcl\_recv {

    # call some detection engine that set req.http.X-UA-Device

}

sub append\_ua {

    if ((req.http.X-UA-Device) && (req.method == "GET")) {

        # if there are existing GET arguments;

        if (req.url ~ "\?") {

            set req.http.X-get-devicetype = "&devicetype=" + req.http.X-UA-Device;

        } else {

            set req.http.X-get-devicetype = "?devicetype=" + req.http.X-UA-Device;

        }

        set req.url = req.url + req.http.X-get-devicetype;

        # 重写url

        unset req.http.X-get-devicetype;

        # 移除req.http.X-get-devicetype

    }

}

# do this after vcl\_hash, so all Vary-ants can be purged in one go. (avoid ban()ing)

sub vcl\_miss { call append\_ua; }

sub vcl\_pass { call append\_ua; }

# Handle redirects, otherwise standard Vary handling code from previous

# examples.

sub vcl\_backend\_response {

    if (bereq.http.X-UA-Device) {

        if (!beresp.http.Vary) { # no Vary at all

            set beresp.http.Vary = "X-UA-Device";

        } elseif (beresp.http.Vary !~ "X-UA-Device") { # add to existing Vary

            set beresp.http.Vary = beresp.http.Vary + ", X-UA-Device";

        }

        # if the backend returns a redirect (think missing trailing slash),

        # we will potentially show the extra address to the client. we

        # don't want that.  if the backend reorders the get parameters, you

        # may need to be smarter here. (? and & ordering)

        if (beresp.status == 301 || beresp.status == 302 || beresp.status == 303) {

            set beresp.http.location = regsub(beresp.http.location, "[?&]devicetype=.\*$", "");

        }

    }

    set beresp.http.X-UA-Device = bereq.http.X-UA-Device;

}

sub vcl\_deliver {

    if ((req.http.X-UA-Device) && (resp.http.Vary)) {

        set resp.http.Vary = regsub(resp.http.Vary, "X-UA-Device", "User-Agent");

    }

}

## Different backend for mobile clients

对手机客户端使用不同的后端。

如果对于手机客户端你有不同的后端来提供页面，或者在VCL种有任何特殊需求，你可以使'X-UA-Device'头像这样：

backend mobile {

    .host = "10.0.0.1";

    .port = "80";

}

sub vcl\_recv {

    # call some detection engine

    if (req.http.X-UA-Device ~ "^mobile" || req.http.X-UA-device ~ "^tablet") {

        set req.backend\_hint = mobile;

    }

}

sub vcl\_hash {

    if (req.http.X-UA-Device) {

        hash\_data(req.http.X-UA-Device);

    }

}

## Redirecting mobile clients

如果想重定向手机客户端可以使用下面的代码片段：

sub vcl\_recv {

    # call some detection engine

    if (req.http.X-UA-Device ~ "^mobile" || req.http.X-UA-device ~ "^tablet") {

        return(synth(750, "Moved Temporarily"));

    }

}

sub vcl\_synth {

    if (obj.status == 750) {

        set obj.http.Location = "http://m.example.com" + req.url;

        set obj.status = 302;

        return(deliver);

    }

}

## 自己的例子

sub vcl\_recv {

    if (req.http.host == "www.example.com" && req.http.User-Agent ~ "(?i)MIDP|WAP|UP.Browser|Smartphone|Obigo|Mobile|AU.Browser|wxd.Mms|WxdB.Browser|CLDC|UP.Link|KM.Browser|UCWEB|SEMC\-Browser|Mini|Symbian|Palm|Nokia|Panasonic|MOT|SonyEricsson|NEC|Alcatel|Ericsson|BENQ|BenQ|Amoisonic|Amoi|Capitel|PHILIPS|SAMSUNG|Lenovo|Mitsu|Motorola|SHARP|WAPPER|LG|EG900|CECT|Compal|kejian|Bird|BIRD|G900/V1.0|Arima|CTL|TDG|Daxian|DAXIAN|DBTEL|Eastcom|EASTCOM|PANTECH|Dopod|Haier|HAIER|KONKA|KEJIAN|LENOVO|Soutec|SOUTEC|SAGEM|SEC|SED|EMOL|INNO55|ZTE|iPhone|Android|Windows CE|Wget|Java|Opera") {

        set req.backend\_hint = mobile;

        return(pass);

    }

}

# Reporting and statistics

本部分包括如何查看Varnish正在做什么，从每个http请求流的详细记录到汇总统计计数器。

## Logging in Varnish

Varnish中一个很棒的特点是工作日志的记录方式。varnish不记录日志到日志文件，而是调用VSL记录到共享内存片段，VSL-varnish共享日志。当这个片段被写完后，varnish开始覆盖老数据。

这种方式比记录到文件快的多得多，并且不需要磁盘空间。除此之外VSL可以给到你需要的尽可能多的信息。

另一方面，如果你忘记有程序实际在写日志到磁盘，日志将会占用过多的磁盘。

varnishlog是可以用来查看varnish记录了什么的程序。varnish提供原生的日志，所有的都被写到日志里。其他一些客户端也能访问日志，一会儿向你展示。

在启动varnish的终端窗口键入varnishlog(varnish-4.0.3: varnishlog -v) 然后回车。

你将看到一些行，缓慢滚动（取决于你的日志量）像这样：

0 CLI          - Rd ping

0 CLI          - Wr 200 PONG 1273698726 1.0

varnish 主进程检查cache进程，为了查看是否一切ok。

刷新指向varnish的web页面，我们可以看到有些输出：

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

* 第一列是随机数，它是为了辨认访问会话。数字相同表示来自同一个会话，同时会被相同的进程处理。
* 第二列是日志信息的标签。所有的请求都被一种活动类型的标签标示着。Rx开始的标签表示是Varnish接收的数据，Tx表示发送的数据。
* 第三列表示数据是从客户端来或者发送到客户端(c)，或者是发送到后端或者从后端返回(b).
* 第四列表示被记录的数据。

基本的选项：

'-b'：只展示varnish和后端server交互的日志。当我们要优化缓存命中率时，非常有用。

[root@aliyun-web1 ~]# varnishlog -b|more

\*   << BeReq    >> 513184412

-   Begin          bereq 513184411 pass

-   Timestamp      Start: 1435903539.233425 0.000000 0.000000

-   BereqMethod    GET

-   BereqURL       /

-   BereqProtocol  HTTP/1.1

-   BereqHeader    Host: www.example.com

-   BereqHeader    Accept: \*/\*

-   BereqHeader    Accept-Charset: GB2312,utf-8;q=0.7,\*;q=0.7

-   BereqHeader    Accept-Language: zh-cn, zh

-   BereqHeader    User-Agent: Mozilla/5.0 (Linux; U; Android 4.4.2; zh-cn; GT-I9500 Build/KOT49H) AppleWebKit/537.36 (KHTML, like Gecko)Version/4.0 MQQBrowser/5.0     QQ-Manager Mobile Safari/537.36

-   BereqHeader    Referer: http://www.example.com/

-   BereqHeader    X-Forwarded-For: 180.153.206.22

-   BereqHeader    X-Forward-For: 180.153.206.22

-   BereqHeader    X-Varnish: 513184412

-   VCL\_call       BACKEND\_FETCH

-   VCL\_return     fetch

-   Backend        723 default default(10.171.199.83,,81)

-   Timestamp      Bereq: 1435903539.233538 0.000113 0.000113

-   Timestamp      Beresp: 1435903539.372638 0.139213 0.139100

-   BerespProtocol HTTP/1.1

-   BerespStatus   200

-   BerespReason   OK

-   BerespHeader   Server: openresty

-   BerespHeader   Date: Fri, 03 Jul 2015 06:05:39 GMT

-   BerespHeader   Content-Type: text/javascript; charset=utf-8

-   BerespHeader   Transfer-Encoding: chunked

-   BerespHeader   Connection: keep-alive

-   BerespHeader   Status: 200 OK

-   BerespHeader   X-UA-Compatible: IE=Edge,chrome=1

-   BerespHeader   ETag: "a8bc468ef546910351aff1736593a25e"

-   BerespHeader   Cache-Control: max-age=0, private, must-revalidate

-   BerespHeader   X-Request-Id: 091be77be62554630e058bb89e6152e8

-   BerespHeader   X-Runtime: 0.099064

-   BerespHeader   X-Rack-Cache: miss

'-c'：和-b类似，不过是client和varnish的交互.

[root@aliyun-web1 ~]# varnishlog -c|more

\*   << Request  >> 617872310

-   Begin          req 511130871 rxreq

-   Timestamp      Start: 1435903851.946365 0.000000 0.000000

-   Timestamp      Req: 1435903851.946365 0.000000 0.000000

-   ReqStart       211.144.121.242 10509

-   ReqMethod      GET

-   ReqURL         /images/example\_v4/logo.png?1435895693

-   ReqProtocol    HTTP/1.1

-   ReqHeader      Accept: image/png, image/svg+xml, image/\*;q=0.8, \*/\*;q=0.5

-   ReqHeader      Referer: http://www.example.com.cn/articles/2015-04-16/909879.html

-   ReqHeader      Accept-Language: zh-CN

-   ReqHeader      User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko

-   ReqHeader      Accept-Encoding: gzip, deflate

-   ReqHeader      Host: static.example.com.cn

-   ReqHeader      DNT: 1

-   ReqHeader      Connection: Keep-Alive

-   ReqHeader      X-Forwarded-For: 211.144.121.242

-   VCL\_call       RECV

-   ReqHeader      X-Forward-For: 211.144.121.242

-   VCL\_return     hash

-   ReqUnset       Accept-Encoding: gzip, deflate

-   ReqHeader      Accept-Encoding: gzip

-   VCL\_call       HASH

-   VCL\_return     lookup

-   Hit            3490076170

-   VCL\_call       HIT

-   VCL\_return     deliver

-   RespProtocol   HTTP/1.1

-   RespStatus     200

-   RespReason     OK

-   RespHeader     Server: openresty

-   RespHeader     Date: Fri, 03 Jul 2015 06:09:53 GMT

-   RespHeader     Last-Modified: Fri, 03 Jul 2015 03:54:53 GMT

-   RespHeader     ETag: "5596078d-242a"

-   RespHeader     Content-Type: image/png

-   RespHeader     Vary: Accept-Encoding

-   RespHeader     Content-Encoding: gzip

-   RespHeader     X-Varnish: 617872310 268850698

-   RespHeader     Age: 58

-   RespHeader     Via: 1.1 varnish-v4

-   VCL\_call       DELIVER

-   RespHeader     X-Cache: HIT from example 317

-   VCL\_return     deliver

译者注：由于各个版本参数的不同。译者使用的varnish-4.0.3和4.0的输出也不尽相同。 不过还是可以通过varnishlog -h查看确定的参数使用方法。 当然如果你安装了varnish对应版本的varnish-docs，你可通过man手册查看具体的用法,[4.0.3官方varnishlog链接.](https://www.varnish-cache.org/docs/4.0/reference/varnishlog.html)

## Statistics

varnish拥有一组很好且非常实用的状态生成工具，实时的生成状态同时分析共享内存日志的日志数据。

### varnishtop

varnishtop读取共享内存日志同时呈现连续最常发生的日志项的更新列表。

list length 6652                                                   web1

4693.76 VCL\_return     deliver

2531.67 RespProtocol   HTTP/1.1

2343.12 VCL\_call       HASH

2343.12 VCL\_call       RECV

2343.12 VCL\_call       DELIVER

2343.12 VCL\_return     lookup

2343.12 RespHeader     Via: 1.1 varnish-v4

2343.12 RespHeader     Server: openresty

2336.85 ReqMethod      GET

2310.85 RespStatus     200

2310.85 RespReason     OK

2310.12 ReqProtocol    HTTP/1.1

2208.67 RespHeader     Connection: keep-alive

2200.67 VCL\_return     hash

2142.21 RespHeader     Accept-Ranges: bytes

2047.21 ReqHeader      Accept-Encoding: gzip

2017.12 Debug          XXX REF 2

1967.76 VCL\_call       HIT

1851.67 Debug          RES\_MODE 2

1726.94 ReqHeader      Accept: \*/\*

1711.48 RespHeader     Last-Modified: Fri, 03 Jul 2015 09:04:41 GMT

使用适当的过滤参数-I, -i, -X and -x ，他可以按照请求的资源，客户端，user-agent，以及其他记录在日志里的信息来分层级显示。

varnishtop -i ReqURL显示客户端请求的URL。varnishtop -i BereqURL显示后端请求URL.varnishtop -I ReqHeader:Accept-Encoding显示客户端发送最频繁Accept-Encoding头。

更多用法参考[varnishtop](https://www.varnish-cache.org/docs/4.0/reference/varnishtop.html#ref-varnishtop)

### varnishhist

varnishhist工具读取varnishd的共享内存日志，同时实时的呈现最近被处理的N个请求的柱状图。

1:20, n = 2000                                                     web1

          |  |

          |  |

          | |||

          | |||

          | |||

          |||||

          |||||

          |||||

          |||||

          |||||

          |||||

          ||||||

          ||||||

          ||||||                                  #

         |||||||                                 ##

         ||||||||                                ###

+-------+-------+-------+-------+-------+-------+-------+-------+-------

|1e-6   |1e-5   |1e-4   |1e-3   |1e-2   |1e-1   |1e0    |1e1    |1e2

N的值和垂直比例显示在左上角位置。水平比例是对数的，命中标记为|，未命中标记为#。

更多信息查看[varnishhist](https://www.varnish-cache.org/docs/4.0/reference/varnishhist.html#ref-varnishhist)

### varnishstat

varnish有许多计数器。它统计miss，hit，存储信息，创建的线程，被删除的对象。varnishstat导出这些计数。当我在调优varnish这个工具显得十分有用。

Uptime mgt:   30+23:10:33                 Hitrate n:        5        5        5

Uptime child: 30+23:10:32                    avg(n):   0.7449   0.7449   0.7449

  NAME                        CURRENT       CHANGE      AVERAGE       AVG\_10

MAIN.uptime                   2675432         1.00         1.00         1.00

MAIN.sess\_conn              305793621       223.68       114.00       161.26

MAIN.sess\_fail                     29         0.00          .           0.00

MAIN.client\_req\_400             24209         0.00          .           0.00

MAIN.client\_req             698099671       352.50       260.00       340.48

MAIN.cache\_hit              537081611       289.59       200.00       287.56

MAIN.cache\_hitpass           17285224         5.99         6.00         6.24

MAIN.cache\_miss              78237657        22.97        29.00        21.22

MAIN.backend\_conn             4715330         0.00         1.00         0.00

MAIN.backend\_fail                1557         0.00          .           0.00

MAIN.backend\_reuse          158437412        58.92        59.00        52.42

MAIN.backend\_toolate           400204         0.00          .           0.00

MAIN.backend\_recycle        158838527        59.92        59.00        53.67

MAIN.backend\_retry              37981         0.00          .           0.00

MAIN.fetch\_head                661309         0.00          .           0.00

Child process uptime:

How long the child process has been running.

一些工具可以规律的导出varnishstat同时利于查看的显示这些计数。其中一个便是[Munin](http://munin-monitoring.org/)。

更多信息请查看[varnishstat](https://www.varnish-cache.org/docs/4.0/reference/varnishstat.html#ref-varnishstat)

# Varnish and Website Performance

本节集中解决如何调优varnish server，同时如何使用varnish优化你的web站点.

一共有三小节.第一小节你应该想到varnish的各种工具和功能,下一小节如何从cache中清除已经缓存的内容.清除内容是一项基本功能,因为它允许给缓存的对象增加 TTL.TTL越大varnish保持在缓存中的时间越久,这意味着varnish处理更多的请求,只将少部分的请求传递到相对较慢的后端.

最后一小节,处理web内容的压缩.当从后端获得内容时varnish可以压缩它,然后传递被压缩后的内容.这种方式可以减少客户现在内容的时间,从而提高你的web站点的性能.

## Achieving a high hitrate

现在，varnish已经启动和运行，你可以通过varnish访问您的Web应用程序。除非你的应用程序是专门工作在网络加速器之后的，你可能需要在varnish的配置或应用程序上做一些改变，以提高varnish的命中率。

varnish不会缓存你的数据，除非它是绝对肯定操作是安全的。因此，对于你为了了解varnish如何决定是否和如何来缓存页面，我们会通过一些工具引导你，你可以找到一些工具便于理解在varnish的配置中发生了什么。

注意，你需要一个工具来查看穿越varnish和后端之间的HTTP头。在varnish服务器，最简单的方法是使用varnishlog和varnishtop，但有时客户端工具也是有意义的。这些工具是我们常用 的

**Tool: varnishtop**

可以使用varnishtop来找出什么url正在被命中。varnishtop -i BereqURL 是基本的命令，向你展示varsnih发送到后端的top请求。你可以在Statistics 中查看varnishtop 用法的其他列子。

**Tool: varnishlog**

当你找出一个频繁发送到后端的URL，你可以使用varnishlog查看该请求。varnishlog -q 'ReqURL ~ "^/foo/bar"' 向你展示来自客户端匹配到/foo/bar的请求。

关于varnishlog如何工作的更多信息请查看Logging in Varnish 或者产看varnishlog的man手册。

扩展的http头，http://www.varnish-cache.org/trac/wiki/VCLExampleHitMissHeader

**Tool: lwp-request**

lwp-request 是Perl www库中的工具。可以发送请求同时向你展示结果的基本程序。 该工具主要是查看http响应头。很多工具都可以实现。比如linux自带的curl工具。

# curl -I http://vg.no/

HTTP/1.1 301 Moved Permanently

Server: Apache/2.2.15 (CentOS)

X-VG-WebServer: vgphoenix-web-02

Location: http://www.vg.no/

Content-Type: text/html; charset=iso-8859-1

X-VG-SolveMe: uggc://jjj.it.ab/ynxfrgngg.ugzy

Date: Mon, 20 Jul 2015 03:28:46 GMT

Connection: keep-alive

X-Cache: HIT:40

Vary: Accept-Encoding,User-Agent

X-VG-WebCache: hmg9-varnish-01

X-Age: 46

Age: 0

还firefox的firebug工具，chrome自带的工具都可以查看。

## Cookies

默认情况，varnish不缓存从后端响应的http头中带有Set-Cookie的对象。如果客户端发送的请求带有Cookie header，varnish将忽略缓存，直接将请求传递到后端。

这可能是多度保守。大量站点使用Google Analytics来分析他们的流量，GA设置cookie信息来跟踪你。该cookie被客户端的js所使用，服务端不感兴趣的。

**Cookies from the client**

对于大量的web应用完全清楚cookie是有意义的，除非你指定不需要忽略cookie的web站点的部分。

在vcl\_recv中的vcl片段将忽略cookie，除非你访问的url中匹配到/admin/

if (!(req.url ~ "^/admin/")) {

    unset req.http.Cookie;

}

相当简单。然而如果你需要做些复杂的事情，像移除几个cookie中的一个，这就变得困难了。不幸的是，varnish没有一些好的工具来操控cookies。我们不得不使用正则表达式来实现。如果你熟练正则表达式你将理解到发生了什么，不然我们还是建议你先看看pcrepattern（PCRE - Perl-compatible regular expressions），或则查看丰富的在线文档。

我们来看看Varnish Software(VS) 的例子。非常简明的设置，varnish cache在前端，后端使用Drupal-based。VS使用一些跟踪谷歌分析的cookie还有一些其他相似的工具。所有的cookie变成一个集合，被js所使用。varnish和Drupal都不需要查看这些cookie，同时自从客户端发送cookie到varnish，varnish将终止正在缓存的页面，varnish需要在VCL中丢弃不必要的cookie。

下面的VCL中我们丢弃以下划线开始的所有cookie：

# 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 = regsub(req.http.Cookie, "^;\s\*", "");

下面的例子，我们移除所有的cookie除了叫做COOKIE1和COOKIE2，你会惊奇的发现，如此完美：

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 == "") {

            unset req.http.Cookie;

        }

    }

}

一个稍微简单的例子。使用相同的功能，能几乎完成。从“other”cookie中过滤出“the one”我们需要的cookie，复制它给header中的另一个，然后复制回请求中，再删除原来的cookie header。验证请求的正确性。：

sub vcl\_recv {

    # save the original cookie header so we can mangle it

    set req.http.X-Varnish-PHP\_SID = req.http.Cookie;

    # using a capturing sub pattern, extract the continuous string of

    # alphanumerics that immediately follows "PHPSESSID="

    set req.http.X-Varnish-PHP\_SID =

        regsuball(req.http.X-Varnish-PHP\_SID, ";? ?PHPSESSID=([a-zA-Z0-9]+)( |;| ;).\*","\1");

    set req.http.Cookie = req.X-Varnish-PHP\_SID;

    unset req.X-Varnish-PHP\_SID;

}

在Varnish Cache Wiki 有更多varnish在VCL中能做什么的惊奇例子。

**Cookies coming from the backend**

使用默认设置的话，后端server如果设置了Set-Cookie头，varnish将不缓存该内容。一个hit-for-pass 的对象被创建。因此如果后端行为诡异同时设置些不需要的cookie，unset 'Set-Cookie'头，一切将变得ok。

**Cache-Control**

Cache-Control头通知缓存，如何处理该内容。Varnish关心' max-age '参数，并且是用它来计算一个对象的TTL。

因此，确定你用max-ag设置了'Cache-Control'头。你可以看看Varnish Software的Drupalserver的响应。

$ curl -I http://www.varnish-software.com/|grep 'Cache-Control'

Cache-Control: max-age=1209600

**Age**

Varnish 增加'Age'头来标明对象被varnish缓存多久。你可以使用\* varnishlog -I RespHeader:^Age\*来从varnishlog筛选出Age。

**Pragma**

HTTP 1.0server可能发送Pragma: nocache 头。varnish忽略pragma头。你很容易增加对该pragma头的支持。 在vclbackendresponse：

if (beresp.http.Pragma ~ "nocache") {

    set beresp.uncacheable = true;

    set beresp.ttl = 120s; # how long not to cache this url.

}

**Authorization**

如果varnish看到header中有'Authorization'头，它将pass请求。如果你不需要这个头，你可以unset。

**Overriding the time-to-live (TTL)**

有时候你的后端会抽下风。在varnish可以容易的重写TTL，然后你再修复你笨重的后端。 需要在VCL中指定想要设置的对象，然后设置'beresp.ttl'

sub vcl\_backend\_response {

    if (bereq.url ~ "^/legacy\_broken\_cms/") {

        set beresp.ttl = 5d;

    }

}

**Forcing caching for certain requests and certain responses**

你的后端可能仍然是笨重的，且不能很好的工作，你可能想存放更多的资源到varnish。我们建议你依赖尽可能多的默认缓存规则。强制varnish在高速缓存中查找对象是很容易的，但它并不是真正的建议。

**Normalizing your namespace**

有些站点是有许多域名的。 http://www.varnish-software.com/, http://varnish-software.com/ and http://varnishsoftware.com/ 都指向同一个站点。而varnish不知道他们是相同的。varnish会根据不同的域名缓存同一个对象不同的版本。你可以减缓这种情况，通过在webserver中配置redirect或者使用下面的VCL：

if (req.http.host ~ "(?i)^(www.)?varnish-?software.com") {

    set req.http.host = "varnish-software.com";

}

## HTTP Vary

HTTP Vary 并不是微不足道的概念。这是迄今为止最被误解的HTTP标头。

大量的响应头告诉客户端关于该HTTP对象是否该被传递。客户端可以基于他们的参数选择请求不同的http对象。他们的参数选择包括像编码和域名。当客户端倾向于UK English时，通过Accept-Language: en-uk 表明。缓存需要保持不同的版本，通过响应中的'Vary'头来实现。

当后端server被Vary:Accept-Language标示时，它告诉Varnish需要对每个不同的Accept-Language保存不同的版本。

如果两个客户端分别接受"en-us, en-uk" 和 "da, de"，同时如果后端标明不同Accept-Language，varnish将缓存和提供两个不同的版本。

请注意Vary头需要完全匹配。因此如果Vary中是 "en-us, en-uk" 和"en-us,en-uk"（多个空格），varnish将保存同一个页面的两个副本。

获得搞得命中率的同时使用Vary，关键是标准化后端的header。记住，哪怕是有一点不同，会导致不同的缓存条目。

下面的vcl代码将标准化Accept-Language头，不管"en", "de" 或者 "fr",以这个优先顺序。

if (req.http.Accept-Language) {

    if (req.http.Accept-Language ~ "en") {

        set req.http.Accept-Language = "en";

    } elsif (req.http.Accept-Language ~ "de") {

        set req.http.Accept-Language = "de";

    } elsif (req.http.Accept-Language ~ "fr") {

        set req.http.Accept-Language = "fr";

    } else {

        # unknown language. Remove the accept-language header and

        # use the backend default.

        unset req.http.Accept-Language

    }

}

**Vary parse errors**

如果varnish解析Vary头出错，将会返回503网络错误。或者客户端的头大小超过的65k. 在这种情况下SLT\_Error日志会被增加。

**Pitfall - Vary: User-Agent**

user-Agent的陷阱。

一些应用或者应用服务器，把'Vary: User-Agent'随着内容一起发送。这个行为通知varnish缓存每个'User-Agent'版本的单独副本。即时是同一个浏览器的单个补丁版本对于不同的运行操作系统也至少有10中不同的User-Agent。

如果你真的需要基于User-Agent多样化，一定要初始化标准化header或者你能忍受varnish的命中率。使用上述代码作为一个模板。