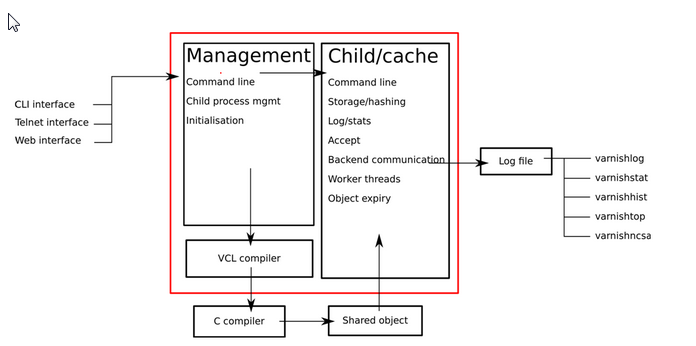
Vanish 详解

varnish是一款高性能且开源的方向代理服务器和HTTP加速器，它的开发者poul-Henning kamp FreeBSD 核心的开发人员之一。varnish采用全新的软件体系机构，和现在的硬件体系配合紧密，

varnish是一个轻量级的cache和反向代理软件。先进的设计理念和成熟的设计框架式varnish的主要特点。现在的varnish总共代码量不大，虽然功能在不断改进，但是还需要继续丰富加强。    varnish的特点：

* 基于内存进行缓存，重启后数据将消失。
* 利用虚拟内存方式，I\O性能好。
* 支持设置0~60秒的精确缓存时间。
* VCL配置管理比较灵活。
* 具有强大的管理功能，例如top、stat、admin、list 等。
* 状态机设计巧妙、结构清晰。
* 利用二叉堆管理缓存文件，可达到积极删除目的。

# vanish系统架构



varnish主要运行两个进程：Management进程和Child进程(也叫Cache进程)。Management进程主要实现应用新的配置、编译VCL、监控varnish、初始化varnish以及提供一个命令行接口等。Management进程会每隔几秒钟探测一下Child进程以判断其是否正常运行，如果在指定的时长内未得到Child进程的回应，Management将会重启此Child进程。

Child进程包含多种类型的线程，常见的如：

* Acceptor线程：接收新的连接请求并响应；
* Worker线程：child进程会为每个会话启动一个worker线程，因此，在高并发的场景中可能会出现数百个worker线程甚至更多；
* Expiry线程：从缓存中清理过期内容；

Varnish依赖“工作区(workspace)”以降低线程在申请或修改内存时出现竞争的可能性。在varnish内部有多种不同的工作区，其中最关键的当属用于管理会话数据的session工作区。

|  |
| --- |
| # ps -aux | grep varnish  root 5436 0.0 0.2 112300 1196 ? Ss 04:19 0:00 /usr/sbin/varnishd -P /var/run/varnish.pid -a :80 -f /etc/varnish/default.vcl -T 127.0.0.1:6082 -t 120 -w 50,1000sh -g varnish -S /etc/varnish/secret -s malloc,100M  varnish 5437 0.0 0.7 1242232 3464 ? Sl 04:19 0:00 /usr/sbin/varnishd -P /var/run/varnish.pid -a :80 -f /etc/varnish/default.vcl -T 127.0.0.1:6082 -t 120 -w 50,1000sh -g varnish -S /etc/varnish/secret -s malloc,100M |

# varnish日志

为了与系统的其它部分进行交互，Child进程使用了可以通过文件系统接口进行访问的共享内存日志(shared memory log)，因此，如果某线程需要记录信息，其仅需要持有一个锁，而后向共享内存中的某内存区域写入数据，再释放持有的锁即可。而为了减少竞争，每个worker线程都使用了日志数据缓存。

共享内存日志大小一般为90M，其分为两部分，前一部分为计数器，后半部分为客户端请求的数据。varnish提供了多个不同的工具如varnishlog、varnishncsa或varnishstat等来分析共享内存日志中的信息并能够以指定的方式进行显示。

|  |
| --- |
| # rpm -ql varnish  # varnish 默认启用了日志滚动的功能；日志滚动的脚本文件；  /etc/logrotate.d/varnish  # varnish 的服务启动脚本文件；  /etc/rc.d/init.d/varnish  # 分析共享内存日志中信息工具的服务启动脚本；  /etc/rc.d/init.d/varnishlog  /etc/rc.d/init.d/varnishncsa  # varnish 全局配置文件  /etc/sysconfig/varnish  # varnish 主配置文件目录  /etc/varnish  # varnish 默认的配置文件  /etc/varnish/default.vcl  # 相关的一些二进制程序  /usr/bin/varnish\_reload\_vcl |
| varnishstat： 因为这几个参数比较重要   * client connections accepted ： 表示客户端反向代理服务器成功发起HTTP请求的数量 * client request recceived： 表示到现在为止：浏览器向反向代理服务器发送HTTP请求累计数，由于可能使用长连接因此这个值一般会大于"client connections accepted的值" * cache his ： 表示反向代理服务器在缓存区中查找并且命中的次数。 * cache misses： 表示直接访问后端主机的请求数量：也就是非命中数： * N struct object ： 表示当前的缓存内容数量； * N expired object ： 表示当过期的缓存内容数量； * N LRU moved objects ： 表示淘汰的缓存内容个数； |
| varnishlog 自带的参数如下：   * -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。 |
| varnishncsa 工具详解：   * -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-ForHTTP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。 |
| varnishd命令参# varnishd -help：  -a address:port # 表示Varnish对httpd的监听地址及端口  -b address:port #表示后端服务器地址及端口  -d # 表示后端服务器地址及端口  -f file # 指定vanish服务器的配置文件  -F # Run in foreground  -n dir # 指定Varnish服务器的配賢文件 指定服务器参数，用来优化Varnish性能  -P file # varnish进程pid文件存放路径  -p param=value # 指定服务器参数，用来优化vanish性能  -t # 指定默认ttl值；单位为s；<br>  -S secret-file # 指定认证文件  -T address:port # 设定varnish的telnet管理地址及端口 telnet 交互式模式调试服务器  -V # 显示varnish版本号和版权信息  -w int[,int[,int]] # 设定Varnish的工作线程数； |

# VCL

Varnish Configuration Language (VCL)是varnish配置缓存策略的工具，它是一种基于“域”(domain specific)的简单编程语言，它支持有限的算术运算和逻辑运算操作、允许使用正则表达式进行字符串匹配、允许用户使用set自定义变量、支持if判断语句，也有内置的函数和变量等。使用VCL编写的缓存策略通常保存至.vcl文件中，其需要编译成二进制的格式后才能由varnish调用。事实上，整个缓存策略就是由几个特定的子例程如vcl\_recv、vcl\_fetch等组成，它们分别在不同的位置(或时间)执行，如果没有事先为某个位置自定义子例程，varnish将会执行默认的定义。

VCL策略在启用前，会由management进程将其转换为C代码，而后再由gcc编译器将C代码编译成二进制程序。编译完成后，management负责将其连接至varnish实例，即child进程。正是由于编译工作在child进程之外完成，它避免了装载错误格式VCL的风险。因此，varnish修改配置的开销非常小，其可以同时保有几份尚在引用的旧版本配置，也能够让新的配置即刻生效。编译后的旧版本配置通常在varnish重启时才会被丢弃，如果需要手动清理，则可以使用varnishadm的vcl.discard命令完成。

# varnish的后端存储

varnish支持多种不同类型的后端存储，这可以在varnishd启动时使用-s选项指定。后端存储的类型包括：

file：使用特定的文件存储全部的缓存数据，并通过操作系统的mmap()系统调用将整个缓存文件映射至内存区域(如果条件允许)；

malloc：使用malloc()库调用在varnish启动时向操作系统申请指定大小的内存空间以存储缓存对象；

persistent(experimental)：与file的功能相同，但可以持久存储数据(即重启varnish数据时不会被清除)；仍处于测试期；

varnish无法追踪某缓存对象是否存入了缓存文件，从而也就无从得知磁盘上的缓存文件是否可用，因此，file存储方法在varnish停止或重启时会清除数据。而persistent方法的出现对此有了一个弥补，但persistent仍处于测试阶段，例如目前尚无法有效处理要缓存对象总体大小超出缓存空间的情况，所以，其仅适用于有着巨大缓存空间的场景。

选择使用合适的存储方式有助于提升系统性，从经验的角度来看，建议在内存空间足以存储所有的缓存对象时使用malloc的方法，反之，file存储将有着更好的性能的表现。然而，需要注意的是，varnishd实际上使用的空间比使用-s选项指定的缓存空间更大，一般说来，其需要为每个缓存对象多使用差不多1K左右的存储空间，这意味着，对于100万个缓存对象的场景来说，其使用的缓存空间将超出指定大小1G左右。另外，为了保存数据结构等，varnish自身也会占去不小的内存空间。

为varnishd指定使用的缓存类型时，-s选项可接受的参数格式如下：

malloc[,size] 或

file[,path[,size[,granularity]]] 或

persistent,path,size {experimental}

file中的granularity用于设定缓存空间分配单位，默认单位是字节，所有其它的大小都会被圆整。

# Varnish的安装

*7.1、接着，建立Vanish用于以及用户组，并且创建Varnish缓存目录和日志目录*

# useradd -s /sbin/nologin varnish

# mkdir /data/vanish/cache -pv

# mkdir /data/vanish/log

# chown -R varnish:varnish /data/vanish/cache/

# chown -R varnish:varnish /data/vanish/log/

7.2、获取Varnish软件

        Varnish的官方网址为 https://www.varnish-cache.org/   ，这里有varnish的最新说明文档及版本升级记录，在此网站中可以找到varnish在SourceForge中的下载链接。目前，The current stable release of Varnish Cache 3 is 3.0.5，下载完成后包名为varnish-3.0.5.tar.gz，这里以此版本为例，进行安装配置。

7.3、 安装pcre 官方站点： http://www.pcre.org/  下载完成的包名为 pcre-8.35.zip

        如果没有安装pcre，在编译varnish-3.0.5.tar.gz 以上版本时，会提示找不到pcre库，而pcre库是为了兼容正则表达式，所以必须安装pcre库。下面是pcre的安装过程：

如果安装pcre出现如下错误时：

# useradd -s /sbin/nologin varnish

# mkdir /data/vanish/cache -pv

# mkdir /data/vanish/log

# chown -R varnish:varnish /data/vanish/cache/

# chown -R varnish:varnish /data/vanish/log/

解决办法： 按提示应该是文件时间问题，新创建的时间既然比现在的文件时间晚，系统时间问题

hwclock --set --date="月/日/年 小时：分钟：秒钟"

hwclock --hctosys

# hwclock --set --date "04/09/2014 00:00:00"

# hwclock --hctosys

# unzip pcre-8.35.zip

# cd pcre-8.35

# ./configure --prefix=/usr/local/pcre/

# make && make install

 hwclock --hctosys是让上面设置的硬件时间同系统时间同步

7.4、安装Varnish

        这里讲Varnish 安装到/usr/loca 目录下,操作如下：

# tar xvf varnish-3.0.5.tar.gz

# cd varnish-3.0.5

# export PKG\_CONFIG PATH=/usr/local/pcre/lib/pkgconfig

# ./configure --prefix=/usr/local/varnish \

--enable-dependency-trackin \

--enable-debugging-symbols \

--enable-developer-warnings

# cp redhat/varnish.initrc /etc/init.d/vanish

# cp redhat/varnish.sysconfig /etc/sysconfig/varnish

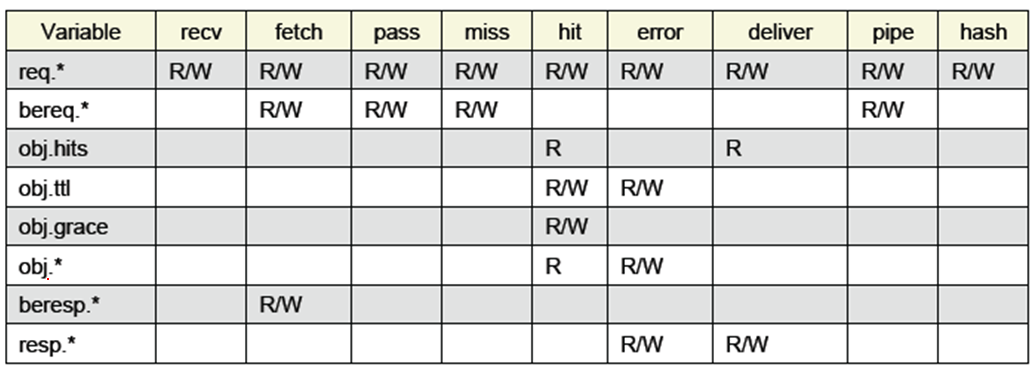
   其 中，"PKG\_CONFIG\_PATH" 是指定Varnish查找pcre库的路径。如果pcre安装在其它路径下，在这里指定相应路径即 可，varnish默认查找pcre库的路径为/usr/local/lib/pkgconfig。最后两步操作时复制一些Varnish守护进程的初始 化脚本文件，这些脚本文件用户varnish的启动，关闭等方面。

    至此，varnish安装完毕了！

**8、配置Varnish**

**8.1、 VCL 使用说明**

VCL 即 为 varnish configuration Language，用来定义varnish的存取策略。VCL 语法比较简单，跟C和perl比较相 似，可以使用指定运算符"="、比较运算符"="、逻辑运算符"!&&!!"等形式；还支持正则表达式和用"~"进行ACL匹配运算；还 可以使用"set"这样的关键字指定变量。 VCL 的语法遵循特定的格式：



**VCL内置函数**

**(1)**

用于接收和处理请求。当请求到达并被成功接收后调用，通过判断请求的数据来决定如何处理请求

pass：表示进入pass模式，把请求控制权交给val\_pass函数。

pipe：表示进入pipe模式，把请求控制权交个vcl\_pipe函数。

error code[reason]：表示返回"code"给客户端，并放弃处理该请求。"code"是错误标识，例如200和405等。"reason"是错误提示信息。

**(2)vcl\_pipe函数**

此函数在进入pipe模式时被调用，用户将请求直接传递至后端主机，在请求和返回的内容没有改变的情况下，将不变的内容返回给客户端，直到这个链接被关闭。

此函数一般以如下几个关键字结束。

erro code[reason]

pipe

**(3) vcl\_pass 函数**

此函数在进入pass模式时被调用，用户将请求直接传递至后端主机。后端主机在应答数据后将应答数据发送给客户端，但不进行任何缓存，在当前链接下每次都返回最新的内容。

此函数一般以如下几个关键字结束。

error code [reason]

pass

**(4)lookup**

表示在缓存中查找被请求的对象，并且根据查找的结果把控制权交给函数vcl\_hit 或函数vcl\_miss

**(5)vcl\_hit 函数**

在执行lookip指令后，在缓存中找到请求的内容后将自动调用该函数。此函数一般以如下几个关键字结束。

deliver：表示将找到的内容发送给客户端，并把控制权交个函数vcl\_deliver

error code[reason]

pass

**(6) vcl\_miss 函数**

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

此函数一般如下几个关键字结束。

fetch：表示从后端获取请求的内容，并把控制权交个vcl\_fetch函数

error code [reason]

pass

**(7)vcl\_fetch函数**

在后端主机更新缓存并且获取内容后调用该方法，接着通过判断获取的内容来决定是将内容放入缓存，还是直接返回给客户端。

此函数一般以如下几个关键字结束。

error code [reason]

pass

deliver

**(8)vcl\_deliver函数**

将在缓存中找到请求的内容发送给客户端前调用此方法。

此函数一般以如下几个关键字结束。

error code [reason]

deliver

**(9)vcl\_timeout 函数**

在缓存内同到期前吊桶此函数。

此函数一般以如下几个关键字结束。

discard：表示从缓存中清除该内同

fetch。

**(10)vcl\_discard函数**

在缓存内容到期后或缓存空间不够时，自动吊桶该函数。

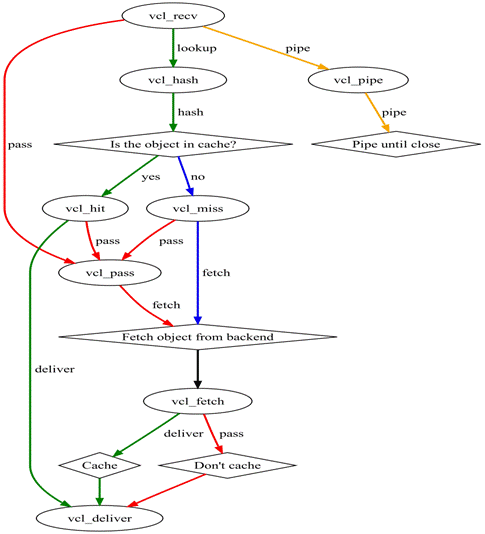
此函数一般如下几个关键字结束。

keep:表示将内容继续保存在缓存中。

discard。

**8.2、VCL处理流程图**

通过以上对VCL函数的介绍，其实你们应该都发现了，其实每个函数之间都是相互关联的。 如下如所示：Varnish处理HTTP请求的运行流程图：



**Varnish处理HTTP请求的过程大致分为如下几个步骤；**

(1)Receive状态。也就是请求处理的入口状态，根据VCL规则判断该请求应该pass或者pipe，还是进入lookup(本地查询)

(2)Lookup状态。进入此状态后，会在hash表中查找数据，若找到，则进入Hit状态，否则进入Miss状态。

(3)Pass状态。在此状态下，会进入后端请求，即进入fetch状态。

(4)Fetch状态。在fetch状态下，对请求进行后端获取，发送请求，获得数据，并进行本地存储。

(5)Deliver状态。将获取到的数据发送给客户端，然后完成本次请求。

**8.3、内置公用变量**

**VCL内置的公用变量可以在不同的VCL函数中。下面根据这些公用变量使用的不同阶段依次介绍。当请求到达后，可以使用的公用变量如下表：**

|  |  |
| --- | --- |
| **公用变量名称** | **含义** |
| req.backend | 指定对应后端主机 |
| server.ip | 表示服务器IP |
| client.ip | 表示客户端IP |
| req.request | 指定请求的类型，例如GET、HEAD和POST等 |
| req.url | 指定请求的地址 |
| req.proto | 表示客户端发起请求的HTTP协议版本 |
| req.http.header | 表示对应请求中的HTTP头部信息 |
| req.restarts | 表示请求重启的次数，默认最大值为4 |

**Varnish 在向后端主机请求时，可以使用的公用变量如下表：**

|  |  |
| --- | --- |
| **公用变量名称** | **含义** |
| beresp.request | 指定请求的类型，例如GET或HEAD等 |
| beresp.url | 指定请求的地址 |
| beresp.proto | 表示客户端发起请求中的HTTP协议版本 |
| beresp.http.header | 表示对应请求中的HTTP头部信息 |
| beresp.ttl | 表示缓存的生存周期,也就是cache保留多长时间单位是秒 |

**从cache或后端主机获取内容后，可以使用的公用变量如下表所示：**

|  |  |
| --- | --- |
| **公用变量名称** | **含义** |
| obj.status | 表示返回内容的请求状态码,例如200、302、504等 |
| obj.cacheable | 表示返回的内容是否可以缓存，也就是说,如果HTTP返回的是200、203、300、301、302、404或410等，并且有非0的生存期，则可以缓存 |
| obj.valid | 表示是否是有效的HTTP应答 |
| obj.response | 表示返回内容的请求状态信息 |
| obj.proto | 表示返回内容的HTTP协议版本 |
| obj.ttl | 表示返回内容的生存周期，也就是缓存时间，单位是秒 |
| obj.lastuse | 表示返回上一次请求到现在的间隔时间，单位是秒 |

**对客户端应答时，可以使用的公用变量，如下表所示：**

|  |  |
| --- | --- |
| **公用变量名称** | **含义** |
| resp.status | 表示返回客户端的HTTP状态代码 |
| resp.proto | 表示返回客户端的HTTP协议版本 |
| resp.http.header | 表示返回客户端的HTTP头部信息 |
| resp.response | 表示返回客户端的HTTP状态信息 |

九、HTTP协议与varnish

1、缓存相关的HTTP首部

HTTP协议提供了多个首部用以实现页面缓存及缓存失效的相关功能，这其中最常用的有：  
(1)Expires：用于指定某web对象的过期日期/时间，通常为GMT格式；一般不应该将此设定的未来过长的时间，一年的长度对大多场景来说足矣；其常用于为纯静态内容如JavaScripts样式表或图片指定缓存周期；  
(2)Cache-Control：用于定义所有的缓存机制都必须遵循的缓存指示，这些指示是一些特定的指令，包括public、private、no-cache(表示可以存储，但在重新验正其有效性之前不能用于响应客户端请求)、no-store、max-age、s-maxage以及must-revalidate等；Cache-Control中设定的时间会覆盖Expires中指定的时间；  
(3)Etag：响应首部，用于在响应报文中为某web资源定义版本标识符；  
(4)Last-Mofified：响应首部，用于回应客户端关于Last-Modified-Since或If-None-Match首部的请求，以通知客户端其请求的web对象最近的修改时间；  
(5)If-Modified-Since：条件式请求首部，如果在此首部指定的时间后其请求的web内容发生了更改，则服务器响应更改后的内容，否则，则响应304(not modified)；  
(6)If-None-Match：条件式请求首部；web服务器为某web内容定义了Etag首部，客户端请求时能获取并保存这个首部的值(即标签)；而后在后续的请求中会通过If-None-Match首部附加其认可的标签列表并让服务器端检验其原始内容是否有可以与此列表中的某标签匹配的标签；如果有，则响应304，否则，则返回原始内容；  
(7)Vary：响应首部，原始服务器根据请求来源的不同响应的可能会有所不同的首部，最常用的是Vary: Accept-Encoding，用于通知缓存机制其内容看起来可能不同于用户请求时Accept-Encoding-header首部标识的编码格式；  
(8)Age：缓存服务器可以发送的一个额外的响应首部，用于指定响应的有效期限；浏览器通常根据此首部决定内容的缓存时长；如果响应报文首部还使用了max-age指令，那么缓存的有效时长为“max-age减去Age”的结果；

十、Varnish状态引擎(state engine)

VCL用于让管理员定义缓存策略，而定义好的策略将由varnish的management进程分析、转换成C代码、编译成二进制程序并连接至child进程。varnish内部有几个所谓的状态(state)，在这些状态上可以附加通过VCL定义的策略以完成相应的缓存处理机制，因此VCL也经常被称作“域专用”语言或状态引擎，“域专用”指的是有些数据仅出现于特定的状态中。  
1、VCL状态引擎

在VCL状态引擎中，状态之间具有相关性，但彼此间互相隔离，每个引擎使用return(x)来退出当前状态并指示varnish进入下一个状态。  
varnish开始处理一个请求时，首先需要分析HTTP请求本身，比如从首部获取请求方法、验正其是否为一个合法的HTT请求等。当这些基本分析结束后就需要做出第一个决策，即varnish是否从缓存中查找请求的资源。这个决定的实现则需要由VCL来完成，简单来说，要由vcl\_recv方法来完成。如果管理员没有自定义vcl\_recv函数，varnish将会执行默认的vcl\_recv函数。然而，即便管理员自定义了vcl\_recv，但如果没有为自定义的vcl\_recv函数指定其终止操作(terminating)，其仍将执行默认的vcl\_recv函数。事实上，varnish官方强烈建议让varnish执行默认的vcl\_recv以便处理自定义vcl\_recv函数中的可能出现的漏洞。

2、VCL语法

VCL的设计参考了C和Perl语言，因此，对有着C或Perl编程经验者来说，其非常易于理解。其基本语法说明如下：  
(1)//、#或/\* comment \*/用于注释  
(2)sub $name 定义函数  
(3)不支持循环，有内置变量  
(4)使用终止语句，没有返回值  
(5)域专用  
(6)操作符：=(赋值)、==(等值比较)、~(模式匹配)、!(取反)、&&(逻辑与)、||(逻辑或)  
  
VCL的函数不接受参数并且没有返回值，因此，其并非真正意义上的函数，这也限定了VCL内部的数据传递只能隐藏在HTTP首部内部进行。VCL的return语句用于将控制权从VCL状态引擎返回给Varnish，而非默认函数，这就是为什么VCL只有终止语句而没有返回值的原因。同时，对于每个“域”来说，可以定义一个或多个终止语句，以告诉Varnish下一步采取何种操作，如查询缓存或不查询缓存等。

3、VCL的内置函数

VCL提供了几个函数来实现字符串的修改，添加bans，重启VCL状态引擎以及将控制权转回Varnish等。

regsub(str,regex,sub)  
regsuball(str,regex,sub)：这两个用于基于正则表达式搜索指定的字符串并将其替换为指定的字符串；但regsuball()可以将str中能够被regex匹配到的字符串统统替换为sub，regsub()只替换一次；  
ban(expression)：  
ban\_url(regex)：Bans所有其URL能够由regex匹配的缓存对象；  
purge：从缓存中挑选出某对象以及其相关变种一并删除，这可以通过HTTP协议的PURGE方法完成；  
hash\_data(str)：  
return()：当某VCL域运行结束时将控制权返回给Varnish，并指示Varnish如何进行后续的动作；其可以返回的指令包括：lookup、pass、pipe、hit\_for\_pass、fetch、deliver和hash等；但某特定域可能仅能返回某些特定的指令，而非前面列出的全部指令；  
return(restart)：重新运行整个VCL，即重新从vcl\_recv开始进行处理；每一次重启都会增加req.restarts变量中的值，而max\_restarts参数则用于限定最大重启次数。

4、vcl\_recv

vcl\_recv是在Varnish完成对请求报文的解码为基本数据结构后第一个要执行的子例程，它通常有四个主要用途：  
(1)修改客户端数据以减少缓存对象差异性；比如删除URL中的www.等字符；  
(2)基于客户端数据选用缓存策略；比如仅缓存特定的URL请求、不缓存POST请求等；  
(3)为某web应用程序执行URL重写规则；   
(4)挑选合适的后端Web服务器；

可以使用下面的终止语句，即通过return()向Varnish返回的指示操作：  
pass：绕过缓存，即不从缓存中查询内容或不将内容存储至缓存中；  
pipe：不对客户端进行检查或做出任何操作，而是在客户端与后端服务器之间建立专用“管道”，并直接将数据在二者之间进行传送；此时，keep-alive连接中后续传送的数据也都将通过此管道进行直接传送，并不会出现在任何日志中；  
lookup：在缓存中查找用户请求的对象，如果缓存中没有其请求的对象，后续操作很可能会将其请求的对象进行缓存；  
error：由Varnish自己合成一个响应报文，一般是响应一个错误类信息、重定向类信息或负载均衡器返回的后端web服务器健康状态检查类信息；

vcl\_recv也可以通过精巧的策略完成一定意义上的安全功能，以将某些特定的攻击扼杀于摇篮中。同时，它也可以检查出一些拼写类的错误并将其进行修正等。

Varnish默认的vcl\_recv专门设计用来实现安全的缓存策略，它主要完成两种功能：  
(1)仅处理可以识别的HTTP方法，并且只缓存GET和HEAD方法；  
(2)不缓存任何用户特有的数据；

安全起见，一般在自定义的vcl\_recv中不要使用return()终止语句，而是再由默认vcl\_recv进行处理，并由其做出相应的处理决策。

下面是一个自定义的使用示例：

sub vcl\_recv {

if (req.http.User-Agent ~ "iPad" ||

req.http.User-Agent ~ "iPhone" ||

req.http.User-Agent ~ "[Android](http://www.it165.net/pro/ydad/)") {

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

} else {

set req.http.X-Device = "desktop";

}

}

此例中的VCL创建一个X-Device请求首部，其值可能为mobile或desktop，于是web服务器可以基于此完成不同类型的响应，以提高用户体验。

5、vcl\_fetch

如前面所述，相对于vcl\_recv是根据客户端的请求作出缓存决策来说，vcl\_fetch则是根据服务器端的响应作出缓存决策。在任何VCL状态引擎中返回的pass操作都将由vcl\_fetch进行后续处理。vcl\_fetch中有许多可用的内置变量，比如最常用的用于定义某对象缓存时长的beresp.ttl变量。通过return()返回给varnish的操作指示有：  
(1)deliver：缓存此对象，并将其发送给客户端(经由vcl\_deliver)；  
(2)hit\_for\_pass：不缓存此对象，但可以导致后续对此对象的请求直接送达到vcl\_pass进行处理；  
(3)restart：重启整个VCL，并增加重启计数；超出max\_restarts限定的最大重启次数后将会返回错误信息；  
(4)error code [reason]：返回指定的错误代码给客户端并丢弃此请求；

默认的vcl\_fetch放弃了缓存任何使用了Set-Cookie首部的响应。

**十一、修剪缓存对象**

1、缓存内容修剪

提高缓存命中率的最有效途径之一是增加缓存对象的生存时间(TTL)，但是这也可能会带来副作用，比如缓存的内容在到达为其指定的有效期之间已经失效。因此，手动检验缓存对象的有效性或者刷新缓存是缓存很有可能成为服务器管理员的日常工作之一，相应地，Varnish为完成这类的任务提供了三种途径：HTTP 修剪(HTTP purging)、禁用某类缓存对象(banning)和强制缓存未命令(forced cache misses)。

这里需要特殊说明的是，Varnish 2中的purge()操作在Varnish 3中被替换为了ban()操作，而Varnish 3也使用了purge操作，但为其赋予了新的功能，且只能用于vcl\_hit或vcl\_miss中替换Varnish 2中常用的set obj.ttl=0s。

在具体执行某清理工作时，需要事先确定如下问题：  
(1)仅需要检验一个特定的缓存对象，还是多个？  
(2)目的是释放内存空间，还是仅替换缓存的内容？  
(3)是不是需要很长时间才能完成内容替换？  
(4)这类操作是个日常工作，还是仅此一次的特殊需求？

2、移除单个缓存对象

purge用于清理缓存中的某特定对象及其变种(variants)，因此，在有着明确要修剪的缓存对象时可以使用此种方式。HTTP协议的PURGE方法可以实现purge功能，不过，其仅能用于vcl\_hit和vcl\_miss中，它会释放内存工作并移除指定缓存对象的所有Vary:-变种，并等待下一个针对此内容的客户端请求到达时刷新此内容。另外，其一般要与return(restart)一起使用。下面是个在VCL中配置的示例。

acl purgers {

"127.0.0.1";

"192.168.0.0"/24;

}

sub vcl\_recv {

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

if (!client.ip ~ purgers) {

error 405 "Method not allowed";

}

return (lookup);

}

}

sub vcl\_hit {

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

purge;

error 200 "Purged";

}

}

sub vcl\_miss {

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

purge;

error 404 "Not in cache";

}

}

sub vcl\_pass {

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

error 502 "PURGE on a passed object";

}

}

客户端在发起HTTP请求时，只需要为所请求的URL使用PURGE方法即可，其命令使用方式如下：

# curl -I -X PURGE http://varniship/path/to/someurl

3、强制缓存未命中

在vcl\_recv中使用return(pass)能够强制到上游服务器取得请求的内容，但这也会导致无法将其缓存。使用purge会移除旧的缓存对象，但如果上游服务器宕机而无法取得新版本的内容时，此内容将无法再响应给客户端。使用req.has\_always\_miss=ture，可以让Varnish在缓存中搜寻相应的内容但却总是回应“未命中”，于是vcl\_miss将后续地负责启动vcl\_fetch从上游服务器取得新内容，并以新内容缓存覆盖旧内容。此时，如果上游服务器宕机或未响应，旧的内容将保持原状，并能够继续服务于那些未使用req.has\_always\_miss=true的客户端，直到其过期失效或由其它方法移除。

4、Banning

ban()是一种从已缓存对象中过滤(filter)出某此特定的对象并将其移除的缓存内容刷新机制，不过，它并不阻止新的内容进入缓存或响应于请求。在Varnish中，ban的实现是指将一个ban添加至ban列表(ban-list)中，这可以通过命令行接口或VCL实现，它们的使用语法是相同的。ban本身就是一个或多个VCL风格的语句，它会在Varnish从缓存哈希(cache hash)中查找某缓存对象时对搜寻的对象进行比较测试，因此，一个ban语句就是类似匹配所有“以/downloads开头的URL”，或“响应首部中包含nginx的对象”。例如：  
ban req.http.host == "magedu.com" && req.url ~ "\.gif$"  
  
定义好的所有ban语句会生成一个ban列表(ban-list)，新添加的ban语句会被放置在列表的首部。缓存中的所有对象在响应给客户端之前都会被ban列表检查至少一次，检查完成后将会为每个缓存创建一个指向与其匹配的ban语句的指针。Varnish在从缓存中获取对象时，总是会检查此缓存对象的指针是否指向了ban列表的首部。如果没有指向ban列表的首部，其将对使用所有的新添加的ban语句对此缓存对象进行测试，如果没有任何ban语句能够匹配，则更新ban列表。

对ban这种实现方式持反对意见有有之，持赞成意见者亦有之。反对意见主要有两种，一是ban不会释放内存，缓存对象仅在有客户端访问时被测试一次；二是如果缓存对象曾经被访问到，但却很少被再次访问时ban列表将会变得非常大。赞成的意见则主要集中在ban可以让Varnish在恒定的时间内完成向ban列表添加ban的操作，例如在有着数百万个缓存对象的场景中，添加一个ban也只需要在恒定的时间内即可完成。其实现方法本处不再详细说明。

十二、Varnish检测后端主机的健康状态

Varnish可以检测后端主机的健康状态，在判定后端主机失效时能自动将其从可用后端主机列表中移除，而一旦其重新变得可用还可以自动将其设定为可用。为了避免误判，Varnish在探测后端主机的健康状态发生转变时(比如某次探测时某后端主机突然成为不可用状态)，通常需要连续执行几次探测均为新状态才将其标记为转换后的状态。

每个后端服务器当前探测的健康状态探测方法通过.probe进行设定，其结果可由req.backend.healthy变量获取，也可通过varnishlog中的Backend\_health查看或varnishadm的debug.health查看。

backend web1 {

.host = "www.magedu.com";

.probe = {

.url = "/.healthtest.html";

.interval = 1s;

.window = 5;

.threshold = 2;

}

}

.probe中的探测指令常用的有：

(1) .url：探测后端主机健康状态时请求的URL，默认为“/”；

(2) .request: 探测后端主机健康状态时所请求内容的详细格式，定义后，它会替换.url指定的探测方式；比如：

.request =

"GET /.healthtest.html HTTP/1.1"

"Host: www.magedu.com"

"Connection: close";

(3) .window：设定在判定后端主机健康状态时基于最近多少次的探测进行，默认是8；

(4) .threshold：在.window中指定的次数中，至少有多少次是成功的才判定后端主机正健康运行；默认是3；

(5) .initial：Varnish启动时对后端主机至少需要多少次的成功探测，默认同.threshold；

(6) .expected\_response：期望后端主机响应的状态码，默认为200；

(7) .interval：探测请求的发送周期，默认为5秒；

(8) .timeout：每次探测请求的过期时长，默认为2秒；

因此，如上示例中表示每隔1秒对此后端主机www.magedu.com探测一次，请求的URL为http://www.magedu.com/.healthtest.html，在最近5次的探测请求中至少有2次是成功的(响应码为200)就判定此后端主机为正常工作状态。

如果Varnish在某时刻没有任何可用的后端主机，它将尝试使用缓存对象的“宽容副本”(graced copy)，当然，此时VCL中的各种规则依然有效。因此，更好的办法是在VCL规则中判断req.backend.healthy变量显示某后端主机不可用时，为此后端主机增大req.grace变量的值以设定适用的宽容期限长度。

七、Varnish使用多台后端主机

Varnish中可以使用director指令将一个或多个近似的后端主机定义为一个逻辑组，并可以指定的调度方式(也叫挑选方法)来轮流将请求发送至这些主机上。不同的director可以使用同一个后端主机，而某director也可以使用“匿名”后端主机(在director中直接进行定义)。每个director都必须有其专用名，且在定义后必须在VCL中进行调用，VCL中任何可以指定后端主机的位置均可以按需将其替换为调用某已定义的director。

backend web1 {  
.host = "backweb1.magedu.com";  
.port = "80";  
}

director webservers random {  
.retries = 5;  
{  
.backend = web1;  
.weight = 2;  
}  
{  
.backend = {  
.host = "backweb2.magedu.com";  
.port = "80";  
}  
.weight = 3;  
}  
}

如上示例中，web1为显式定义的后端主机，而webservers这个directors还包含了一个“匿名”后端主机(backweb2.magedu.com)。webservers从这两个后端主机中挑选一个主机的方法为random，即以随机方式挑选。

Varnish的director支持的挑选方法中比较简单的有round-robin和random两种。其中，round-robin类型没有任何参数，只需要为其指定各后端主机即可，挑选方式为“轮叫”，并在某后端主机故障时不再将其视作挑选对象；random方法随机从可用后端主机中进行挑选，每一个后端主机都需要一个.weight参数以指定其权重，同时还可以director级别使用.retires参数来设定查找一个健康后端主机时的尝试次数。

Varnish 2.1.0后，random挑选方法又多了两种变化形式client和hash。client类型的director使用client.identity作为挑选因子，这意味着client.identity相同的请求都将被发送至同一个后端主机。client.identity默认为client.ip，但也可以在VCL中将其修改为所需要的标识符。类似地，hash类型的director使用hash数据作为挑选因子，这意味着对同一个URL的请求将被发往同一个后端主机，其常用于多级缓存的场景中。然而，无论是client还hash，当其倾向于使用后端主机不可用时将会重新挑选新的后端其机。

另外还有一种称作fallback的director，用于定义备用服务器，如下所示：

director b3 fallback {  
{ .backend = www1; }  
{ .backend = www2; } // will only be used if www1 is unhealthy.  
{ .backend = www3; } // will only be used if both www1 and www2  
// are unhealthy.  
}

set client.identity = req.http.cookie

十三、varnish管理进阶

1、可调参数

Varnish有许多参数，虽然大多数场景中这些参数的默认值都可以工作得很好，然而特定的工作场景中要想有着更好的性能的表现，则需要调整某些参数。可以在管理接口中使用param.show命令查看这些参数，而使用param.set则能修改这些参数的值。然而，在命令行接口中进行的修改不会保存至任何位置，因此，重启varnish后这些设定会消失。此时，可以通过启动脚本使用-p选项在varnishd启动时为其设定参数的值。然而，除非特别需要对其进行修改，保持这些参数为默认值可以有效降低管理复杂度。

2、共享内存日志

共享内存日志(shared memory log)通常被简称为shm-log，它用于记录日志相关的数据，大小为80M。varnish以轮转(round-robin)的方式使用其存储空间。一般不需要对shm-log做出更多的设定，但应该避免其产生I/O，这可以使用tmpfs实现，其方法为在/etc/fstab中设定一个挂载至/var/lib/varnish目录(或其它自定义的位置)临时文件系统即可。

3、线程模型(Trheading model)

varnish的child进程由多种不同的线程组成，分别用于完成不同的工作。例如：  
cache-worker线程：每连接一个，用于处理请求；  
cache-main线程：全局只有一个，用于启动cache；  
ban lurker线程：一个，用于清理bans；  
acceptor线程：一个，用于接收新的连接请求；  
epoll/kqueue线程：数量可配置，默认为2，用于管理线程池；  
expire线程：一个，用于移除老化的内容；  
backend poll线程：每个后端服务器一个，用于检测后端服务器的健康状况；

在配置varnish时，一般只需为关注cache-worker线程，而且也只能配置其线程池的数量，而除此之外的其它均非可配置参数。与此同时，线程池的数量也只能在流量较大的场景下才需要增加，而且经验表明其多于2个对提升性能并无益处。

4、线程相关的参数(Threading parameters)

varnish为每个连接使用一个线程，因此，其worker线程的最大数决定了varnish的并发响应能力。下面是线程池相关的各参数及其配置：

thread\_pool\_add\_delay 2 [milliseconds]

thread\_pool\_add\_threshold 2 [requests]

thread\_pool\_fail\_delay 200 [milliseconds]

thread\_pool\_max 500 [threads]

thread\_pool\_min 5 [threads]

thread\_pool\_purge\_delay 1000 [milliseconds]

thread\_pool\_stack 65536 [bytes]

thread\_pool\_timeout 120 [seconds]

thread\_pool\_workspace 16384 [bytes]

thread\_pools 2 [pools]

thread\_stats\_rate 10 [requests]

其中最关键的当属thread\_pool\_max和thread\_pool\_min，它们分别用于定义每个线程池中的最大线程数和最少线程数。因此，在某个时刻，至少有thread\_pool\_min\*thread\_pools个worker线程在运行，但至多不能超出thread\_pool\_max\*thread\_pools个。根据需要，这两个参数的数量可以进行调整，varnishstat命令的n\_wrk\_queued可以显示当前varnish的线程数量是否足够，如果队列中始终有不少的线程等待运行，则可以适当调大thread\_pool\_max参数的值。但一般建议每台varnish服务器上最多运行的worker线程数不要超出5000个。

当某连接请求到达时，varnish选择一个线程池负责处理此请求。而如果此线程池中的线程数量已经达到最大值，新的请求将会被放置于队列中或被直接丢弃。默认线程池的数量为2，这对最繁忙的varnish服务器来说也已经足够。

十四、Varnish的命令行工具

1、varnishadm命令

命令语法：varnishadm [-t timeout] [-S secret\_file] [-T address:port] [-n name] [command [...]]

通过命令行的方式连接至varnishd进行管理操作的工具，指定要连接的varnish实例的方法有两种：  
-n name —— 连接至名称为“name”的实例；  
-T address:port —— 连接至指定套接字上的实例；

其运行模式有两种，当不在命令行中给出要执行的"command"时，其将进入交互式模式；否则，varnishadm将执行指定的"command"并退出。要查看本地启用的缓存，可使用如下命令进行。

# varnishadm -S /etc/varnish/secret -T 127.0.0.1:6082 storage.list