> `varnish`是一个高速 web 加速器，基于反向代理的方式将varnish部署到 http 服务器前面，并将客户端请求的内容缓存到 varnish 服务器上。当客户端再次请求相同的内容时，能够快速交付给客户端，varnish官方地址。此外，开源的缓存解决方案还有 `squid` 。挪威最大的在线报社使用3台varnish缓存替换了12台squid，性能比之前好，varnish作为一个新起之秀，在性能方面有更为优秀的表现。

缓存相关概念

局部性原理

> 程序的局部性原理可分为`空间局部性`和`时间局部性`，所谓空间局部性是指程序在执行过程中涉及到的某个数据存储单元，则在不久之后其附近存储单元中的数据也可能被访问到。时间局部性是指某存储单元中的数据被访问，则在不久之后该数据可能再次被访问到。正是程序执行呈现出局部性的规律使得缓存成为可能，更加有效。客户端请求的web资源，极有可能再次访问该资源，如果依旧从原始服务器请求资源，网络、数据访问延时都有可能造成客户端访问较慢，所以可以将内容缓存到缓存服务器，当客户端请求的时候可以快速的从缓存服务器取得资源。

缓存命中率

> 计算缓存命中率的公式：缓存命中率 = 命中次数 / (命中次数 + 未命中次数)，命中率是一个范围在0-1之间的数值，通常情况下缓存命中率可以很好的衡量缓存系统加速效果的优劣。衡量缓存效果的还可以基于页面数量和页面的字节数进行衡量。并不是缓存命中低，缓存效果就不好。

varnish缓存对象

> 缓存对象指缓存的内容。缓存对象有生命周期，过期缓存会被定期清理。缓存空间耗尽时，缓存程序会使用 LRU(最近最少未使用算法) 算法清除过去缓存。缓存对象分为可缓存和不可缓存的对象，例如不可缓存对象通常有用户私有数据。缓存对象的 key为 URL 的哈

> 希值，值为 URL 所对应的数据。
>
> `LRU Least Recently Used(页面置换算法)`：因为缓存数据是存储在内存中，而可缓存的空间又是及其有限和宝贵的，为了避免缓存将内存空间耗尽，当缓存把内存空间占尽的时候，varnish管理进程会将最近最少使用的内存清除出内存，从而保证可缓存空间不被耗尽。

新鲜度检测机制

> 存在一种可能，当后端服务器的 web 资源发生改版，此时缓存服务器中缓存的内容还是之前陈旧的内容，如果客户端此时发起请求，则得到的内容是旧的内容，这种现象是我们不愿意看到的。所以，需要对缓存的内容进行新鲜度进行检测，简而言之，就是检测缓存的内容所对应的原始服务器上的内容是否发生改变。缓存新鲜度检测机制有：`过期机制`、`条件式请求`。
>
> **expires：**在 HTTP/1.0 版本中引入一个 `expires` 首部。例如响应报文中：`expires: Thu, 09 Nov 2017 04:56:27 GMT` 首部，它表示 Web 服务在响应给客户端时告知客户端此内容可以缓存到什么时候，如果内容过期了，只有在客户端和服务器端验证有效性后才能后响应给客户端。
>
> **Etag：**例如：`ETag:"6eb6-5434e6543b740"`，如果内容被修改了，其 Etag 也会别修改，所以ETag 的作用跟Last-Modified 的作用差不多，主要供 WEB 服务器判断一个对象是否改变了。比如前一次请求某个 html 文件时，获得了其ETag，当再次又请求这个文件时，浏览器就会把先前获得的 ETag 值发送给 WEB 服务器，然后 WEB 服务器会把这个 ETag 跟该文件的当前 ETag 进行对比，然后就知道这个文件有没有改变了。
>
> **条件式请求：**

```
# last-modified
last-modified:Sat, 14 May 2016 05:28:03 GMT
```

> 请求首部，浏览器将此首部发送给服务器并进行比对内容最后比如文件的最后修改时间，动态页面的最后产生时间等等。

```
# If-Modified-Since
If-Modified-Since: Thu, 30 May 2017 07:07:52 GMT
```

> 该请求首部告诉服务器如果客户端传来的最后修改时间与服务器上的一致，则直接响应304 和响应报头即可。当前各浏览器均是使用的该请求首部来向服务器传递保存的 Last-Modified 值。
>
> **Cache-Cntrol:**

请求首部值

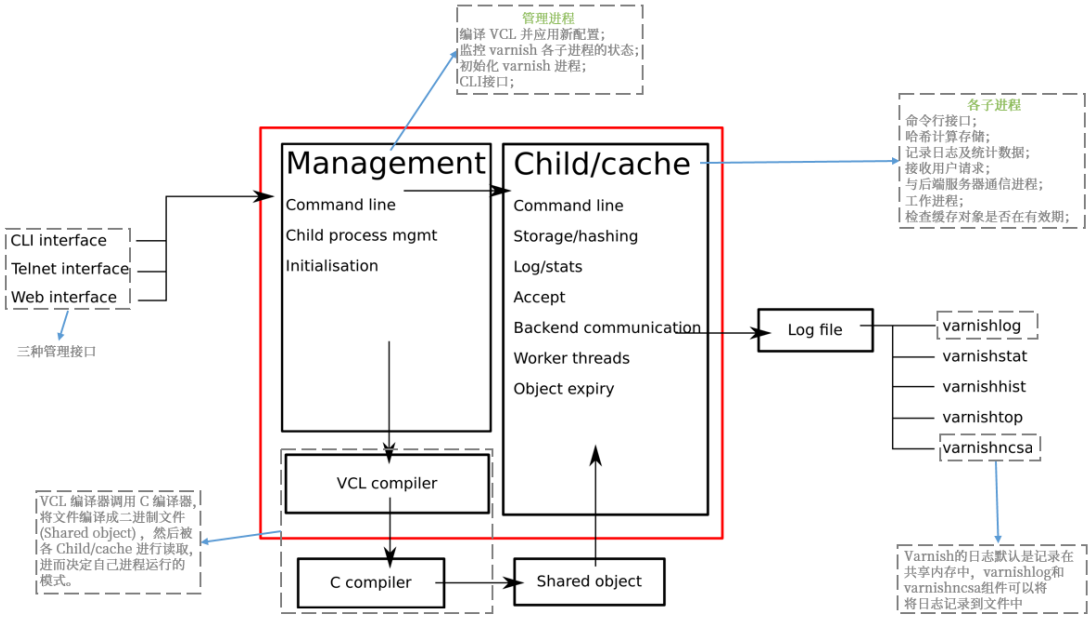
首部值	含义
no-cache	告知代理缓存服务器，不从缓存中获得内容，从原始服务器获取资源
no-store	不缓存请求或响应的任何内容
max-age	只接受 Age 值小于 max-age 值，并且没有过期的对象
max-stale	可以接受过期的对象，但是过期时间必须小于 max-stale 值

> 响应首部值

首部值	含义
public	可向任意方提供的缓存
private	只能用该缓存内容回应先前请求该内容的那个用户
s-maxage	意思和 max-age 类似，但是只用于公有缓存，在公有缓存中使用的时候会覆盖 max-age 的值
no-cache	可以缓存，但是只有在跟WEB服务器验证了其有效后，才能使用缓存
max-age	本响应包含的对象的过期时间

varnish介绍

varnish设计结构



> varnish 类似于 Nginx 程序，分为 `Management` 进程和 `child` 子进程。Management 进程读取 VCL 配置文件并初始化缓存空间以及管理子进程。子进程负责缓存内容、记录日志、接收用户请求、与后端服务器通信、检查缓存等。

varnish工作流程

> varnish 缓存服务器首先是一个代理服务器，代为后端服务器接收客户端发起的请求。当 varnish 接收到客户端发起的请求后，varnish 工作进程首先会查看自己缓存中是否存在客户端要请求的资源是否是可缓存内容，如果不是可缓存内容，则向后端原始服务器发起请求，之后将将内容返回给客户端，如果是可缓存内容，则首先查看缓存是否有客户端请求的资源，如果有则直接返回，如果没有则向后端原始服务器请求资源，然后将资源缓存到本地，同时将资源返回给客户端。

varnish安装配置

varnish安装

```
yum install varnish
```

> varnish 的缓存是基于 key/value 存储在内存中的，这就带来一个问题，就是当缓存的内容很多，时间长之后，会产生 内存碎片，从而会影响到缓存性能。varnish 依赖于 `jemalloc` 程序包，这个程序能够更高效的完成内存分配。

varnish文件列表

```
# 部分
$ rpm -ql varnish
/etc/varnish/default.vcl                        # 配置文件
/etc/varnish/varnish.params                     # varnish进程启动传递的参数，即varnish进程的工作属性
/run/varnish.pid                                # varnish进程pid文件
/usr/bin/varnishadm                             # varnish命令行接口管理工具
/usr/bin/varnishlog                             # 记录日志的服务进程程序
/usr/bin/varnishncsa                            # 记录日志的服务进程程序
/usr/bin/varnishstat                            # 查看varnish状态信息
/usr/bin/varnishtop                             # 动态排序日志信息
/usr/lib/systemd/system/varnish.service         # varnish Unit文件
/usr/lib/systemd/system/varnishlog.service      # varnishlog记录日志Unit文件
/usr/lib/systemd/system/varnishncsa.service     # varnishncsa记录日志Unit文件
```

```
/usr/sbin/varnish_reload_vcl                    # 重新加载配置文件
/usr/sbin/varnishd                              # varnish主程序
/usr/share/doc/varnish-4.0.4/builtin.vcl        # 内建的缓存策略
```

启动服务测试

> 说明：后端使用 Apache 作为 web 服务器，IP地址为172.18.26.2，varnish 版本为 4.0.4 ， IP地址为172.18.26.1。

```
# 在 /etc/varnish/varnish.params 文件中修改varnish进程监听的地址和端口
VARNISH_LISTEN_PORT=80
```

```
# 在 /etc/varnish/varnish.default.vcl 文件中修改后端原始服务器的IP地址和端口
backend default {
    .host = "172.18.26.2";
    .port = "80";
}
# varnish 服务器响应客户端时添加一个自定义首部
sub vcl_deliver {
    if (obj.hits>0) {
        set resp.http.X-cache = "HIT from " + server.ip;
    } else {
        set resp.http.X-cache = "MISS from " + server.ip;
    }
}
# 启动服务
systemctl start varnish.service
```

> 客户端请求测试，看客户端收到的请求报文是否有`X-cache`首部信息，如果从varnish服务器中取得，则会显示。

> `X-cache: HIT From172.18.26.1`

可以传递varnish进程的参数

```
# 传递给varnish的参数可以在/etc/varnish/varnish.params 配置文件中的指定，并在Unit启动文件中指定参数变量
-f VCL_NAME                                  # 指定vcl文件
-a address[:port][,address[:port][...]  # 指明监听地址和端口，可以监听多个地址和端口。默认监听端口是6081、6082
-d                                           # 启动调试模式
-g GROUP_NAME                                # 指定以哪个组的身份运行程序
-g USER_NAME                                 # 指定以哪个用户的身份运行程序
-s [name=]type[,options]                     # 指定缓存存储方式
-T address[:port]                            # 在指定的地址和端口上提供管理界面
-r PARAM_NAME ...                            # 设定可读参数列表
-p param=value                               # 设定额外的配置参数，可以运行时修改。也可以使用 varnishadm 的子命令 param.set thread_pools 4
# 可运行时修改的参数
thread_pools                    # 定义线程池个数，最好等于或小于CPU数量
thread_pool_max                 # 每个线程池最多线程
thread_pool_min                 # 每个线程池最少线程，最大空闲线程数
thread_pool_timeout             # 当线程数量超过thread_pool_min时，空闲时间又超过此值时，释放线程
thread_pool_add_delay           # 创建线程的延长时间
thread_pool_destroy_delay       # 释放线程的超时时长，可缓慢的释放线程
```

varnish存储缓存对象的方式

> 1. `file` 基于文件存储，将所有缓存都放置于一个文件中，不支持持久机制。重启服务，则保存在内存中的缓存元数据丢失。存放到文件系统上，临时有效。示
>   例："file,/data/cache/varnis/bin,2G"
> 2. `malloc`基于内存存储，容易生成内存碎片。示
>   例："malloc,256M"
> 3. `persistent` 基于文件的持久存储

VCL介绍

VCL状态引擎

> VCL (Varnish Configuration Language)，是专门用于配置 varnish 服务器缓存策略及管理缓存的一种语言，其风格类似于 C 语言。当一个新的配置被加载后，由 VCC 进程创建 Manager 进程，最后将 VCL 转换为 C 语言代码，C 语言依赖于 gcc 编译器，由 gcc 将配置编译为一个共享对象，最后被加载到各子进程。VCL 配置的工作流被看做一个有限状态，例如`sub vcl_recv`就是线程刚接收到客户端请求时的状态。VCL 的状态引擎有一下几种：

```
# VCL state Engine
vcl_pipe          # 当varnish接收到自己无法理解的协议请求时，会将请求直接转发后端服务器
vcl_pass          # 到后端服务器取资源
vcl_error         # 当访问的资源没有权限或者错误时，varnish直接返回错误信息
vcl_hash          # 自定义hash生成时的数据来源
vcl_hit           # (缓存命中)从缓存中查找到缓存对象时要执行的操作
vcl_miss          # (缓存未命中)从缓存中没有查找到缓存对象时要执行的操作
vcl_deliver       # 将用户请求的内容响应给客户端时用到的方法
vcl_waiting       # 并发请求过大，等待状态
vcl_purge         # 清理缓存，此例程监控用户是否有清理缓存的请求，之后交给 vcl_synth 例程
```

> VCL state Engine之间存在相关性，其在状态的切换流程：

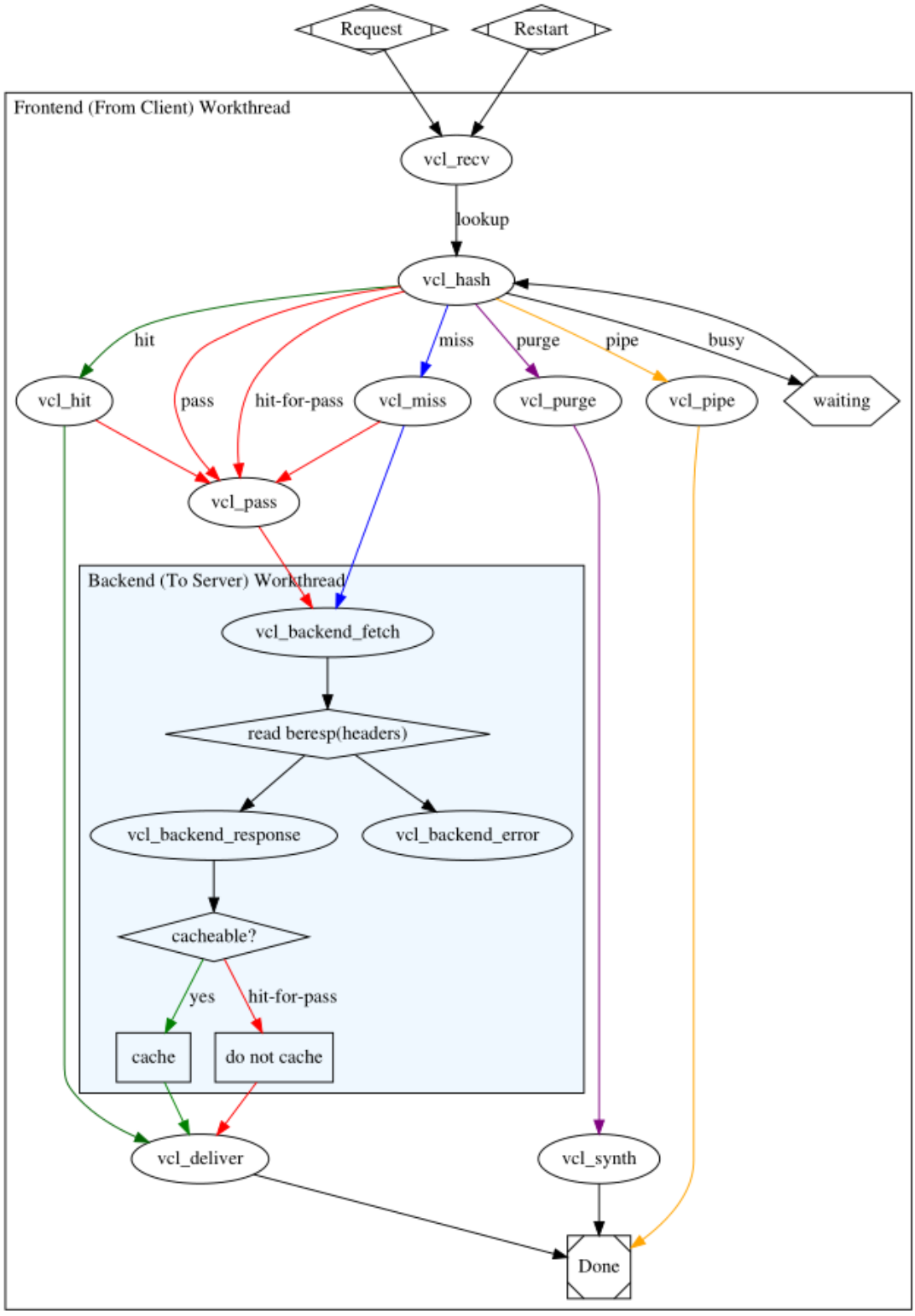


Figure 23: Simplified Version of the Varnish Finite State Machine

> varnish 工作进程接收到客户端请求，首先进程的 VCL 子例程是 `vcl_recv`，如果客户端所请求的内容不可缓存内容，则直接进入 `vcl_pass` 状态，之后向原始服务器请求资源，否则对客户端请求资源的URL取哈希值，进而比对所请求的资源是否在本地缓存区域， 如果本地缓存中有，则进程 `vcl_hit` 状态。在某些场景下，我们需要手动对缓存对象进行修改，例如清除缓存对象，此时可以

> 发起特殊的修改请求，varnish 收到请求后，发现是清除缓存的要求，则进入`vcl_purge`状态，之后交友`vcl_synth`子例程对子例程对缓存对象进行修改。如果 varnish 接收到无法理解的请求，例如一个 PUT请求，由于 varnish 无法理解且不能正确响应，此时varnish只是一个代理服务器的作用，将请求转发给后端的原始服务器，vcl 状态进入`vcl_pipe`。当请求连接 varnish 服务器的并发量很大，varnish 工作进程无法处理时，新的连接状态进入`vcl_watting`。
>
> 通常我们在指定 varnish 的缓存策略时，就要根据状态的切换以及在那个时间点上进行操作。

VCL语法格式

> 1. 注释：`/abc/`、`#abc`、`/*abc*/`
> 2. 定义Subroutines
>
> ```
> sub vcl_SubROUtines_Name {
>     .....
> }
> ```
>
> 3. 条件判断，不支持循循环
>
> ```
> if (CONDTION) {
>
> } else {
>
> }
>
> ```
>
> 4. 变量：`set name=value` `unset name`
> 5. 操作符：`= == ~ ! && ||`
> 6. 支持终止语句，return(action),没有返回值,实现状态引擎的转换

VCL变量 更多参考

> **req**
>
> 客户端请求的报文中首部的值，*表示可以是客户端请求的请求报文中的头部名称。
>
> `req.proto`：表示客户端发起请求的HTTP协议版本
>
> `req.restarts`：客户端发起请求后URL被重写，由此客户端重新请求的次数
>
> `req.http.*`：表示客户端请求的请求报文中的头部名称
>
> **resp**
>
> varnish响应客户端的报文。
>
> `resp.reason`：varnish发送给客户端报文的原因短语
>
> **bereq**
>
> 发给后端服务器的请求。
>
> `bereq.http.HEADERS`：发往后端服务器的请求报文的指定首部
> `bereq.request`：请求方法
> `bereq.url`：请求URL
> `bereq.proto`：使用协议版本
> `bereq.backend`：选择调用后端的那一台服务器
>
> **beresp**
>
> 后端服务器响应。
>
> `bereq.proto` ： 后端服务器响应的协议版本
> `bereq.status`：后端服务器状态
> `bereq.backend.ip`： 后端服务器IP
> `bereq.backend.name`：后端服务器主机名
> `beresp.do_gzip`： 是否压缩后存入缓存
> `beresp.do_gunzip`：是否解压后存入缓存
> `beresp.http.HEADERS` ： 响应报文的首部

> `beresq.reason`：响应报文的原因短语
> `beresq.ttl`：后端服务器响应的内容的余下的生存时长。总共可缓存时长减去传输网络延时时长
>
> **client**
>
> `client.ip`：发请求至varnish主机的客户端IP
>
> **server**
>
> `server.ip`：varnish主机的IP
>
> `server.hostname`：varnish主机的Hostname
>
> **obj**
>
> `obj.hits`：缓存对象从缓存中命中的次数
>
> `obj.ttl`：缓存对象的ttl

varnish中的内置函数 更多参考

> **清理缓存对象**
>
> `ban(expression)`，示例 ban("req.http.host == " + req.http.host + " && req.url == " + req.url);

varnish配置应用

1.特定资源不允许缓存

> 在后端服务器web站点目录下创建 admin 目录，配置此目录下的资源必须向后端服务器请求，而不使用缓存。使用URL匹配的方式。

```
# 编辑配置 varnish 缓存策略文件 default.vcl
sub vcl_recv {
    if (req.url ~ "(?i)^/admin" || req.url ~ "(?i)^/login") {
        return(pass);
    }
    if (req.http.Authorization || req.http.Cookie) {
        return (pass);
    }
    return (hash);
}
# "(?i)"表示对URL不区分大小写匹配
```

> 客户端使用curl命令测试，/admin路径下的资源始终都是从原始服务器请求，而没有缓存。

2.取消私有cookie并设定缓存时长

> 默认情况下带有cookie信息的资源，varnish不会将它缓存。如果某种资源带有cookie标识，但又希望此资源缓存，并缓存较长时间。

```
# 编辑配置 varnish 缓存策略文件 default.vcl
sub vcl_backend_response {
    if (beresp.http.cache-control !~ "s-maxage") {
        if (bereq.url ~ "(?i)\.(png|jpg)$") {
            unset beresp.http.Set-Coolie;
            set beresp.ttl = 3600s;
        }
        if (bereq.url ~ "(?i)\.css$") {
            set beresp.ttl = 300s;
            unset beresp.http.Set-Coolie;
        }
    }
}
```

3.原始服务器记录真实客户端IP

> **X-Forwarded-For（XFF）**是用来识别通过HTTP代理或负载均衡方式连接到Web服务器的客户端最原始的IP地址的HTTP请求头字段。其格式为：`X-Forwarded-For: client1, proxy1, proxy2`，最左边（client1）是最原始客户端的IP地址，代理服务器每成功收到一个请求，就把请求来源IP地址添加到右边。

```
# 编辑配置varnish缓存策略文件 default.vcl
sub vcl_recv {
    if (req.restarts == 0) {
        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;
        }
    }
}
```

> 后端web服务器在记录日志的时候，将接收到的X-Forwarded-For首部的值记录下来。以Apache为例。

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

4.缓存对象修剪 PURGE

> 当某些资源我们希望缓存尽快失效，或者重新刷新某资源的缓存，则需要手动修建缓存对象。为安全，需要添加ACL规则，使得只有特定用户可以执行PURGE操作。

```
# 编辑配置varnish缓存策略文件 default.vcl
acl purgers {
    "172.18.26.4"/16;
}
sub vcl_recv {
    if (req.method == "PURGE") {
        if (!client.ip ~ purgers) {
            return(synth(405,"Purging not allowed for " + client.ip));
        }
        return(purge);
    }
}
```

> 使用curl命令手动修剪缓存对象

```
# 将2560x1600.jpg图片替换为其他内容，此时客户端请求到的还是旧的内容，所以需要修剪缓存对象
curl -X PURGE http://172.18.26.1/images/2560x1600.jpg
```

```
[ root@node3 ~ ]# curl -X PURGE http://172.18.26.1/images/2560x1600.jpg
<!DOCTYPE html>
<html>
  <head>
    <title>200 Purged</title>
  </head>
  <body>
    <h1>Error 200 Purged</h1>
    <p>Purged</p>
    <h3>Guru Meditation:</h3>
    <p>XID: 32777</p>
    <hr>
    <p>Varnish cache server</p>
  </body>
</html>
```

5.缓存对象修剪 ban

> 使用varnish管理工具进行缓存对象修剪。

```
# 格式
ban <field> <operator> <arg>
#示例
ban req.url ~ (?i).(jpg|jpeg)$
```

> 写入配置文件，使用ban(expression)内置函数进行缓存修剪。

```
# 编辑配置varnish缓存策略文件 default.vcl
if (req.method == "BAN") {
    ban("req.http.host == " + req.http.host + " && req.url == " + req.url);
    return(synth(200, "Ban added"));
}
```

6.动静分离

> 实现动静分离，使得缓存服务器从不同的后端服务器取的资源。

```
# 编辑配置varnish缓存策略文件 default.vcl
backend web1 {
    .host = "172.18.26.2";
    .port = "80";
    .probe = {
        .url = "/";
    }
}
backend web2 {
    .host = "172.18.26.3";
    .port = "80";
    .probe = {
        .url = "/";
    }
}
sub vcl_recv {
    if (req.url ~ "(?i)\.(jpg|png|gif)$") {
        set req.backend_hint = web1;
    } else {
        set req.backend_hint = web2;
    }
}
```

7.配置多个后端主机实现负载均衡效果

```
import directors;
backend web1 {
    .host = "172.18.26.2";
    .port = "80";
    .probe = {
        .url = "/";
    }
}
backend web2 {
    .host = "172.18.26.3";
    .port = "80";
    .probe = {
        .url = "/";
    }
}
# vcl状态初始化。定义一个新的集群，使用round_robin()调度方式。此外调度方式还有：hash、random、fallback
sub vcl_init {
    new cluster1 = directors.round_robin();
    cluster1.add_backend(web1);
    cluster1.add_backend(web2);
}
sub vcl_recv {
    set req.backend_hint = cluster1.backend();
}
```

8.健康状态检测

```
# 定义健康状态检测机制
probe healthcheck {                 # healthcheck为检测名称 {定义healthcheck}
    .url = "/";                     # 定义健康检查的页面，表示对那个页面进行检测
    .interval = 6s;                 # 探测请求的发送周期，默认为5秒；
```

```
    .expected_response = 200;    # 期望检测对检测页面请求时的返回状态码，默认为200
    .timeout = 0.3s;             # 每次探测请求的过期时长
    .window = 8;                 # 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
    .threshold = 3;              # 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在健康运行
    .initial = 3;                # 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;
}
backend web1 {
    .host = "...";
    .port = "...";
    .probe = healthcheck;          # 是否对后端主机进行健康状态检测
    .max_connections N;            # 并发连接最大请求数
    .connect_timeout = 0.5s;       # 连接超时时长
    .first_byte_timeout = 20s;     # 第一个字节传输超时时长
    .between_bytes_timeout = 5s; # 两个字节之间传输的超时时长
    .max_connections = 50;         # 最大连接数
}
```

> 手动调整健康状态

```
backend.set_health BACKEND_NAME sick     # 标记为维护模式
backend.set_health BACKEND_NAME auto     # 由探测结果决定后端主机是否健康
backend.set_health BACKEND_NAME heakthy # 直接标记为健康状态
```

varnish管理工具

varnishadm

> 管理varnishd进程的工具，通过127.0.0.1:6082端口连接。也可以交互式方式给varnishd进程传递参数。

```
# 语法格式
varnishadm [-n ident] [-t timeout] [-S secretfile] -T [address]:port command [...] -n is mutually exlusive with -S and -T
# 示例
varnishadm -S /etc/varnish/secret -T 127.0.0.1:6082
# 交互式子命令
help                    # 帮助信息
200
help [<command>]
ping [<timestamp>]  # 探测varnish
quit                    # 退出
banner
status                  # 查看varnish进程状态。子进程是否处于正常状态
start                   # 启动子进程
stop                    # 停止子进程
vcl.load <configname> <filename>     # 装载编译指定VCL文件为配置文件
vcl.inline <configname> <quoted_VCLstring>
vcl.use <configname>            # 指明使用哪一个VCL文件
vcl.discard <configname>    # 指明删除哪一个VCL文件
vcl.list                        # 列出所有可用的VCL文件。boot表示启动时加载的文件，数字表示文件编号
param.show [-l] [<param>]   # 显示运行时参数
param.set <param> <value>   # 配置运行时参数
panic.show                      # 查看恐慌线程列表
panic.clear                     # 清理恐慌线程
storage.list                    # 存储列表
vcl.show [-v] <configname>
```

```
backend.list [<backend_expression>]
# 查看后端服务器列表
backend.set_health <backend_expression> <state>
# 手动设定后端服务器的状态
ban <field> <operator> <arg> [&& <field> <oper> <arg>]...   # 修剪缓存对象
ban.list
```

varnishlog

> 查看varnish共享内存中的日志信息，仅查看新生成的日志。

varnishncsa

> 另一种风格的日志信息。

varnishstat

> 查看varnishd状态信息。

```
# 语法格式
varnishstat [-1] [-x] [-j] [-f field_list] [-l] [-n varnish_name] [-N filename] [-V] [-w delay]
# 示例
varnishstat -l   # 查看选项
varnishstat -1 -f MAIN.cache_hit    # 查看缓存命中的次数
varnishstat -1 -f MAIN.cache_hit    # 查看缓存未命中的次数
```

varnishtop

> 动态排序日志信息。

```
# 选项
-1        # 一次性显示排序后的数据。默认为动态显示
-i        # 显示指定标签的值的显示。多个标签值使用逗号 "," 隔开
-i        # 显示指定标签的值的显示。多个标签值使用逗号 "," 隔开
-I        # 显示指定标签的值的显示。多个标签值使用逗号 "," 隔开，支持正则表达式
-x        # 排除指定标签的值的显示
-X        # 排除指定标签的值的显示，支持正则表达式
```