私塾在线《高级软件架构师实战培训阶段一》

——跟着cc学架构系列精品教程

本部分课程概览

- 根据实际的应用需要，学习要用到的Varnish的知识，以快速上手、理解并掌握Varnish
- 一：Varnish简介
  包括：是什么、能干什么、特点
- 二：Varnish安装和基本使用
  包括：通过源码安装、基本的配置示例、启动、运行、关闭等
- 三：VCL基础
  包括：VCL基本语法、backend、director、负载均衡、后端服务器健康检查、ACL访问控制、Grace模式、Saint模式、常用函数、常用的HTTP头等
- 四：VCL进阶
  包括：VCL子程序、VCL变量、常见VCL应用片断等
- 五：管理Varnish
  包括：telnet进入CLI的方式、Varnishadm的方式、Varnishadm命令

本部分课程概览

- 六：系统掌握Varnishd命令
  包括：Varnishd命令、运行选项、运行期参数等

- 七：查看Varnish运行日志
  包括：Varnishlog命令、Varnishtop命令、Varnishhist命令、Varnishsizes命令、Varnishnsca命令和Varnishreplay命令；理解Varnish的共享内存

- 八：监控、分析并优化Vanish
  包括：规划Varnish的缓存大小、Varnishstat统计信息、提高Varnish的命中率、Varnish的性能优化

Varnish简介

- Varnish是什么

  Varnish是一款开源的、高性能的HTTP加速器和反向代理服务器
- Varnish能干什么

  最主要的功能就是：通过缓存来实现Web访问加速
- Varnish特点

  主要基于内存或者是虚拟内存进行缓存，性能好

  支持设置精确的缓存时间

  VCL配置管理比较灵活

  后端服务器的负载均衡和健康检查

  局部支持ESI

  URL地址重写

  优雅的处理后端服务器宕机的问题

  32位机器上缓存文件大小为最大2GB

Varnish安装

- 源码安装

演示环境：CentOS6.5

1：需要gcc，系统自带了，没有的话，需要先安装

2：需要pcre，这个前面讲Nginx已安装了

3：需要libedit-dev，安装命令：yum install libedit-dev*

4：去https://www.varnish-cache.org/ 下载，然后进行解压安装，示例如下：

（1）先解压源码包，然后进入到这个包里面

（2）安装命令示例如下：

第一步：因为安装varnish需要pcre，因此先设置一下路径：

```
export PKG_CONFIG_PATH=/usr/local/pcre/lib/pkgconfig
```

第二步：

```
./configure --prefix=/usr/common/varnish
```

第三步：

配置后就依次 make ， make install

安装过后，如果从外面访问不了，多半是被防火墙挡住了，可以关闭掉防火墙：

```
/sbin/service iptables stop
```

Varnish基本运行

- 运行前的准备

  把配置文件default.vcl上传到 etc/varnish目录里面去

- 运行的基本命令示例

  ./varnishd -f /usr/common/varnish/etc/varnish/default.vcl -s malloc,32M -T 127.0.0.1:2000 -a 0.0.0.0:1111

其中：1：-f 指定要运行的配置文件

2： -s malloc,32M ： -s 选项用来确定varnish使用的存储类型和存储容量，这里使用的是malloc类型（malloc是一个C函数，用于分配内存空间）

3：-T 127.0.0.1:2000 ： 指定varnish的管理ip和端口

4： -a 0.0.0.0:1111 ：指定varnish对外提供web服务的ip和端口

接下来就可以测试一下了：

http://192.168.0.103:1111/arch1/goods/toList ，这是访问varnish的

http://192.168.0.103:8080/arch1/goods/toList ，这是直接访问Tomcat的

- 关闭varnish

  到valish/sbin的路径下，运行 pkill varnished

VCL-1

- VCL简介

  VCL（Varnish Configuration Language）：Varnish配置语言，语法简单，功能强大，类似于c、perl。主要用来配置如何处理请求和内容的缓存策略。

  VCL在执行时，会转换成二进制代码。

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

- 基本语法介绍

  1：用花括号做界定符，使用分号表示声明结束。注释用//、#、/* */

  2：赋值（=）、比较（==）、和一些布尔值（！、&&、||），!(取反)等类似c语法

  3：支持正则表达式，ACL匹配使用~ 操作

  4：不同于C的地方，反斜杠（\）在VCL中没有特殊的含义。只是用来匹配URLs

  5：VCL没有用户定义的变量，只能给backend、request、document这些对象的变量赋值，大部分是手工输入的，而且给这些变量分配值的时候，必须有一个VCL兼容的单位

  6：VCL有if，但是没有循环。

  7：可以使用set来给request的header添加值，unset 或 remove 来删除某个header

VCL-2

- 申明backend

一个backend申明创建和初始化一个backend目标：

```
backend sishuok {
      .host = "www.sishuok.com";
      .port = “8080";
}
```

一个请求可以选择一个Backend：

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

还可以给它设置很多的参数，如下所示：

VCL-3

```
backend sishuok {
        .host = "www.sishuok.com";
        .port = “8080";
        .connect_timeout = 1s;
        .first_byte_timeout = 5s;
        .between_bytes_timeout = 2s;
        .max_connections=1000;
}
```

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

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

VCL-4

- Director：是backend的逻辑分组或backend的集群。主要有随机、循环和DNS几种Director，不同类型的Director具有不同的算法来选择backend。比如随机的Director示例如下：

```
director b2 random {
        .retries = 5;
        {
                .backend = b1;//引用已经存在的backend
                .weight  = 7;
        }
        {
                .backend  = { //或者是直接在这里定义backend
                        .host = "fs2";
                }
                .weight  = 3;
        }
}
```

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

.weight表示这个后端的权重

VCL-5

- 随机的Director又分成三种，分别是：random、client、hash，他们采用同样的随机分发算法，只是种子数值不同，种子数分别采用随机数、客户端id，或者是缓存的hash（典型如url）。
- 对于client director

  你可以通过设置VCL的变量client.identity来区分客户端，值可以从session cookie 或其它相似的值来获取
- 对于hash director

  默认使用URL的hash值，可以通过 req.hash 获取到

- round-robin director

  它没有什么选项，就是一次循环使用backend，第一个请求用第一个backend，第二个请求用第二个，以此类推。

  如果某个backend出了问题，它会继续尝试下一个，理论上它要尝试完所有的backend，都不好用的话，才会出错。

VCL-6

- DNS director有两种不同的方法来选择后端，一种是random或者round-robin；另一种是使用.list（list的方式不支持ipv6）：

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

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

VCL-7

n fallback director：选择第一个健康的backend，示例：

```
director b3 fallback {
    { .backend = www1; }
    { .backend = www2; } // 第一个不好用，才会到这里
    { .backend = www3; } // 前两个都不好用，才会到这里  }
```

n probe（后端探针）：探测后端，确定他们是否健康，返回的状态用req.backend.healthy核对：

```
backend sishuok {
    .host = "www.sishuok.com";
    .port =  “8080";
    .probe = {       .url = "/test.jpg";
          .timeout = 0.3 s;
          .window = 8; //要检查后端服务器的次数
          .threshold = 3; //.window里面要有多少polls成功就认为后端是健康的
          .initial = 3; //当varnish启动的时候，要确保多少个probe正常
    } }
```

当然，也可以把 probe从backend中拿出来单独定义，形如：

```
backend sishuok{……
    .probe=p1;
}
probe p1{……}
```

VCL-8

- 可能用到的参数：
  - .url：访问backend的路径，缺省是”/”
  - .request：设置详细的请求头，是一些字符串
  - .window：要检查后端服务器的次数，默认是8
  - .threshold：.window里面要有多少polls成功就认为后端是健康的，默认是3
  - .initial：当varnish启动的时候，要确保多少个probe正常，默认和threshold一样
  - .expected_response：期望的response code，默认是200
  - .interval：定义probe多久检查一次后端，默认是5秒
  - .timeout：定义probe的过期时间，默认是2秒
- 也可以指定原始的http请求，形如：

```
backend sishuok {
    .host = "www.sishuok.com";
    .port = “8080";
    .probe = {
          .request ="GET / HTTP/1.1 “
                    "Host: www.foo.bar “
                    "Connection: close";
    }}
```

VCL-9

- ACLs ：访问控制列表，示例如下：

```
acl local {
    "localhost";
    "192.0.2.0"/24;
    ! "192.0.2.23";
}
```

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

下面是一个匹配的示例：

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

Grace模式

- GRACE模式

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

如果您的服务每秒有数千万的点击率，那么这个队列是庞大的，没有用户喜欢等待服务器响应。为了解决这个问题，可以指示varnish去保持缓存的对象超过他们的TTL（就是该过期的，先别删除），并且去提供旧的内容给正在等待的请求。

为了提供旧的内容，首先我们必须有内容去提供。使用以下VCL，以使varnish保持所有对象超出了他们的TTL30分钟。

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

这样，varnish还不会提供旧对象。为了启用varnish去提供旧对象，我们必须在请求上开启它。下面表示，我们接收15s的旧对象：

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

你可能想知道，为什么，如果我们无法提供这些对象，我们在缓存中保持这些对象30分钟？如果你开启健康检查，你可以检查后端是否出问题。如果出问题了，我们可以提供长点时间的旧内容。

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

所以，总结下，Grace模式解决了两个问题：

1：通过提供旧的内容，避免请求扎堆。

2：如果后端坏了，提供旧的内容。

Saint模式

- Saint模式

神圣模式可以让你抛弃一个后端服务器的某个页面，并尝试从其他服务器获取，或提供缓存中的旧内容。让我们看看如何在VCL中开启：

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

设置beresp.saintmode为10秒时，varnish会不请求该服务器10秒。或多或少可以算是一个黑名单。restart被执行时，如果我们有其他后端可以提供该内容，varnish会请求它们。当没有其他后端可用，varnish就会提供缓存中的旧内容。

VCL常用的函数

- 在VCL里面，可以使用如下这些内置函数：

hash_data(str)：

增加一个散列值，默认hash_data() 是调用request的host和url

regsub(str，regex，sub)：

用sub来替换指定的目标

regsuball(str，regex，sub)：

用sub替换所有发现的目标

ban_url(regex)：

禁用缓存中url匹配regex的所有对象， ban_url(regex)预计在4.0会去掉，建议使用ban(expression)

ban(expression)：

禁用缓存中匹配表达式的所有对象，这是一种清空缓存中某些无效内容的方法

Vanish常用的HTTP头

- Cache-Control：指定了缓存如何处理内容。varnish关心max-age参数，并用它来计算对象的TTL。“Cache-Control:no-cache”是被忽略的。
- Age：varnish添加了一个Age头信息，以指示在Varnish中该对象被保持了多久。你可以通过varnishlog像下面那样抓出Age：varnishlog -i TxHeader -I ^Age
- Pragma：一个HTTP 1.0服务器可能会发送”Pragma:no-cache”。Varnish忽略这种头信息。在VCL中你可以很方便的增加对这种头信息的支持，在vcl_fetch中：

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

- Authorization：varnish看到授权头信息时，它会pass该请求。你也可以unset这个头信息
- Cookies：varnish不会缓存来自后端的具有Set-Cookie头信息的对象。同样，如果客户端发送了一个Cookie头信息，varnish将绕过缓存，直接发给后端。
- Vary： Vary头信息是web服务器发送的，代表什么引起了HTTP对象的变化。可以通过Accept-Encoding这样的头信息弄明白。当服务器发出”Vary:Accept-Encoding”，它等于告诉varnish，需要对每个来自客户端的不同的Accept-Encoding缓存不同的版本。所以，如果客户端只接收gzip编码。varnish就不会提供deflate编码的页面版本。

  如果Accept-Encoding字段含有很多不同的编码，比如浏览器这样发送：

  Accept-Encodign: gzip,deflate 另一个这样发送:

  Accept-Encoding: deflate,gzip 因为Accept-Encoding头信息不同，varnish将保存两种不同的请求页面。规范Accept-Encoding头信息将确保你的不同请求的缓存尽可能的少，后面有个例子。

VCL的子程序-1

- 一个子程序就是一串可读和可用的代码，子程序在VCL中没有参数，也没有返回值。示例如下：

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

- 调用一个子程序，使用子程序的关键字名字，如下所示：call pipe_if_local;
- 有很多默认子程序和varnish的工作流程相关，这些子程序会检查和操作http头文件和各种各样的请求，决定哪个、哪些请求被使用，如果这些子程序没有被定义，或者没有完成预定的处理而被终止，控制权将被转交给系统默认的子程序。它们是：

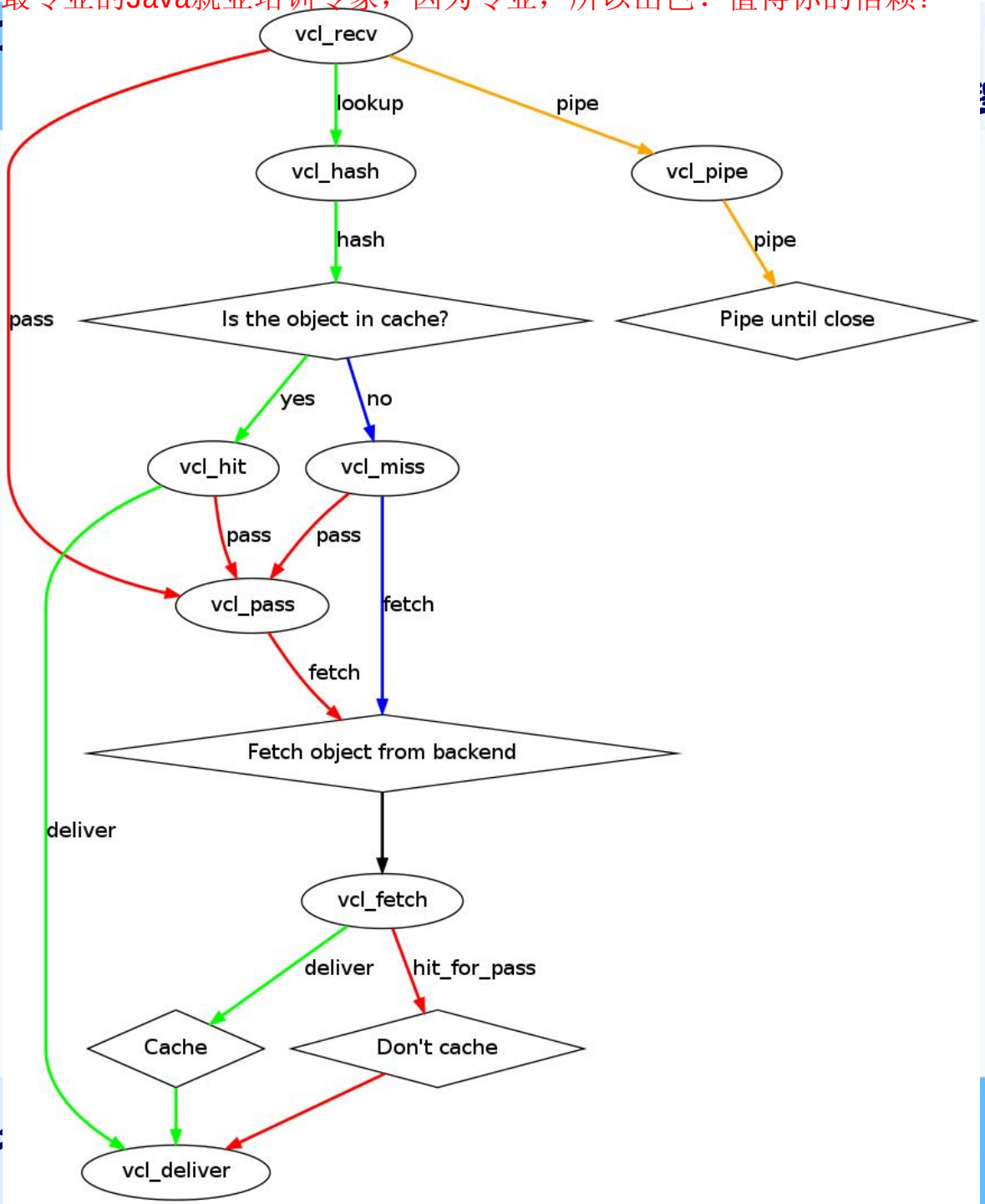
1：vcl_init

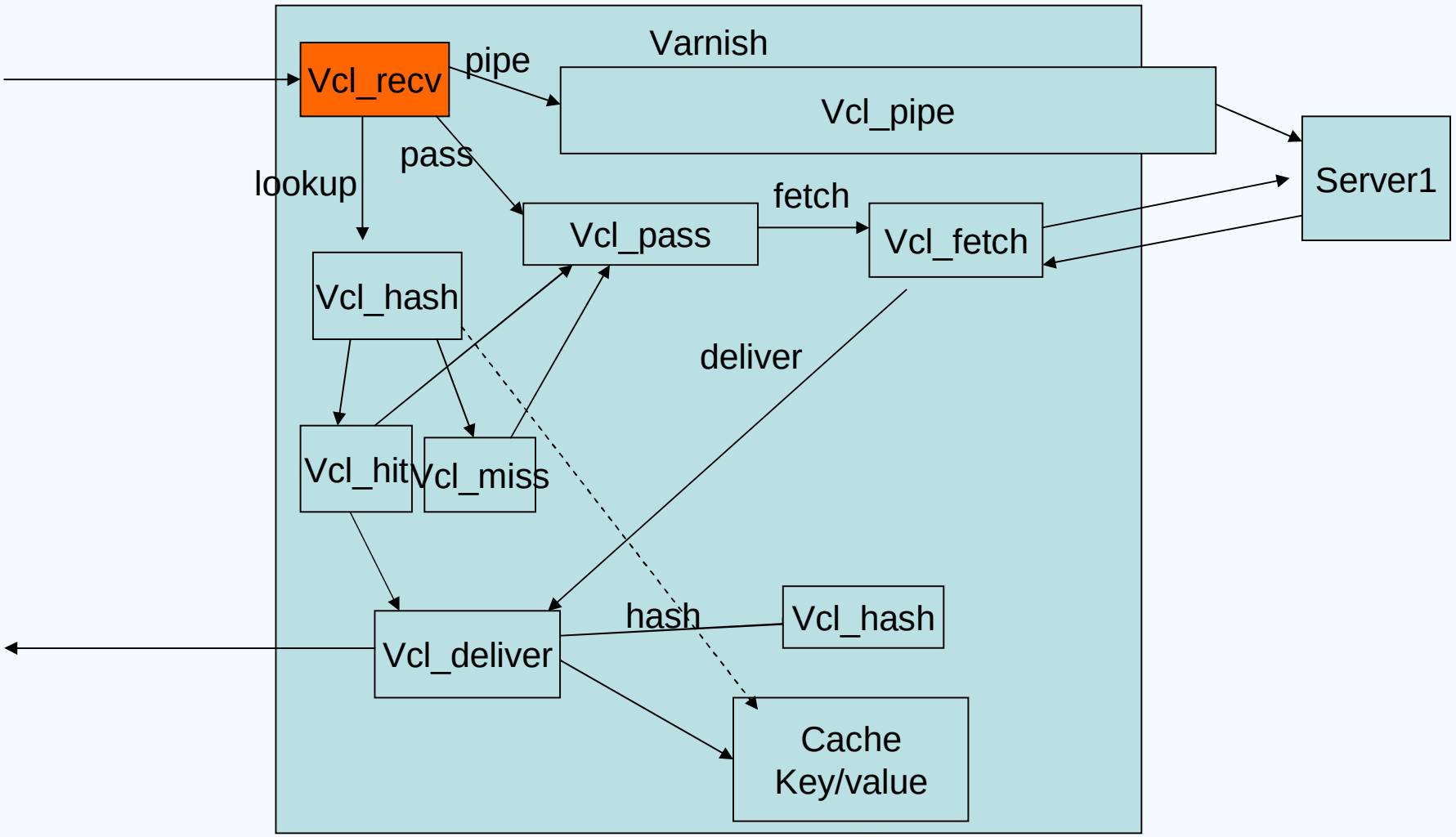
当VCL加载时调用，之后加载客户请求。一般用于初始化VMOD模块。

返回值有：ok

表示正常返回值，返回OK后VCL加载。

- VCL的Request流





VCL的子程序-3

- Actions，VCL中主要有如下动作：

  pass：当一个请求被pass后，这个请求将通过varnish转发到后端服务器，该请求不会被缓存，后续的请求仍然通过Varnish处理。pass可以放在vcl_recv 和vcl_fetch中。

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

  pipe：pipe和pass相似，都要访问后端服务器，不过当进入pipe模式后，在此连接未关闭前，后续的所有请求都直接发到后端服务器，不经过Varnish的处理。

  deliver：请求的目标被缓存，然后发送给客户端

  hit_for_pass：表示直接从后台获取数据，会创建一个hit_for_pass的对象，该对象的TTL值将会被设置成beresp.ttl的当前值。用来控制vcl_deliver如何处理当前的请求，后续的请求会直接vcl_pass，可在vcl_fetch中用

  fetch：从后端服务器获取请求目标，控制权转交给vcl_fetch。

  hash：进入Hash模式

  restart：重启本次事务，重新返回给vcl_recv，如果重启次数超过了max_restarts报错

  ok：表示正常

  error：表示错误

VCL的子程序-4

- 在VCL中，有3个重要的数据结构：

req：

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

beresp：

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

obj：

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

VCL的子程序-5

2：vcl_recv

在请求的开始被调用，在接收、解析后，决定是否响应请求，怎么响应，使用哪个后台服务器。在vcl_recv中，可以修改请求，比如可以修改cookies，添加或者删除请求的头信息。注意vcl_recv中只有请求的目标。vcl_recv子程序以下面的关键字结束：error code [reason] 、pass、pipe、lookup

3：vcl_pipe

请求进入pipe模式的时候被调用，在这个模式，请求会被passed到后端服务器，在连接关闭前，无论是这个客户端还是对应的后端服务器的数据，都会进入pass模式。vcl_pipe子程序以下面的关键字结束：error code [reason]、pipe

4：vcl_pass

请求进入pass模式的时候被调用，在这个模式，请求会被passed到后端服务器，后端服务器的应答会被passed给客户端，但是不会被缓存。相同客户端的随后的请求正常处理。vcl_pass子程序以下面的关键字结束：

error code [reason]、pass、restart

VCL的子程序-6

5：vcl_hash

使用req.hash += req.http.Cookie 或者HTTP头文件包含的cookie生成hash字符串。vcl_hash将以下面的关键字结束：hash

6：vcl_hit

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

error code [reason]、pass、deliver、restart

7：vcl_miss

当需要的内容没有在缓存中命中的时候被调用，决定是否尝试到后端服务器查找目标，从哪个后端服务器查找目标，vcl_miss子程序以下面的关键字结束：error code [reason] 、pass、fetch

8：vcl_fetch

在一个文件成功从后台获取后被调用，通常他的任务就是改变response headers，触发ESI进程，在请求失败的时候轮询其他服务器。在vcl_fetch中一样的包含请求的对象，还有返回对象beresp，它将会包含后端服务器的返回信息。以下面的关键字结束：error code [reason]、 hit_for_pass、deliver、restart

VCL的子程序-7

9：vcl_deliver

当一个没有被cached内容交付给客户端的时候被调用，vcl_deliver子程序以下面关键字结束：deliver、restart

10：vcl_error

当hit错误或者是发生内部错误的时候。以下面关键字结束： deliver、restart

11：vcl_fini

当销毁VCL程序的时候调用，

return值：ok

表示正常销毁VCL程序

VCL的变量-1

n 由于子程序没有参数，子进程必须的信息通过全局变量来处理。

以下是到处都可用的变量：

now：当前时间

下面的变量在backend申明中有效：

.host：一个backend的主机名或者IP地址

.port：一个backend的服务名字或者端口号

下面的变量在处理请求时有效：

client.ip：客户端IP

client.identity：客户的id，用在负载均衡的时候的client director

server.hostname：server的主机名

server.identity：server的身份，使用-i 参数设置，如果－i参数没有传递给varnishd，server.identity将给varnishd实例设置名字。

server.ip：客户端连接上socket，接收到的IP地址

server.port：客户端连接上socket，接收到的端口号

req.request：请求类型，例如”GET”,”HEAD”

VCL的变量-2

req.proto：客户端的Http协议
req.url：请求的URL
req.backend：使用哪个后端服务器为这个请求提供服务
req.backend.healthy：后端服务器是否健康
req.http.header：对应的HTTP头
req.hash_always_miss：强制本请求的缓存失效
req.hash_ignore_busy：当lookup缓存的时候，忽略busy的对象
req.can_gzip：设置能使用gzip
req.restarts：设置最大的重启次数
req.esi：设置是否支持ESI，今后会改变，建议不要使用
req.esi_level：设置ESI的level
req.grace：设置对象被保持的时间
req.xid：请求的唯一id

VCL的变量-3

下面这些变量在访问backend的时候用

bereq.request：请求的类型(如 "GET", "HEAD")

bereq.url：请求的url

bereq.proto：请求的协议

bereq.http.header：请求的HTTP header

bereq.connect_timeout：等待后端服务器响应的时间

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

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

下面这些变量在从backend取回，但还没有进入缓存的时候使用，也就是vcl_fetch变量

beresp.do_stream：对象会直接返回给客户端，不会在varnish中缓存。在Varnish3里面，这些对象将会被标记为busy

beresp.do_esi：是否进行ESI处理

beresp.do_gzip：是否在存储前Gzip压缩

beresp.do_gunzip：是否在存储前解压缩

beresp.http.header：HTTP header

beresp.proto：HTTP的协议

VCL的变量-4

beresp.status：HTTP的状态码
beresp.response：服务端返回的状态消息
beresp.ttl：对象保存的时间
beresp.grace：对象grace保存的时间
beresp.saintmode：saint模式持续的时间
beresp.backend.name：response的backend的名字
beresp.backend.ip：response的backend的ip
beresp.backend.port：response的backend的端口
beresp.storage：强制Varnish保存这个对象

下面这些变量在请求目标被成功的从后端服务器或者缓存中获得后有效

obj.proto：返回请求目标的HTTP版本
obj.status ：服务器返回的HTTP状态码
obj.response ：服务器返回的HTTP状态信息
obj.ttl：目标的剩余生存时间，以秒为单位。
obj.lastuse：最后一个请求后，过去的时间，以秒为单位。
obj.hits：大概的delivered的次数，如果为0，表明缓存出错。
obj.grace：对象grace的存活时间
obj.http.header：Http header

VCL的变量-5

下面这些变量在目标hash key以后有效

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

下面这些变量在准备回应客户端时使用

resp.proto：准备响应的HTTP协议版本

resp.status：返回客户端的HTTP状态码

resp.response：返回客户端的HTTP状态信息

resp.http.header：通信的HTTP头

使用SET关键字，把值分配给变量：

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

可以使用remove关键字把HTTP头彻底的删除：

```
sub vcl_fetch {
        remove obj.http.Set-Cookie;
}
```

常见VCL应用片断-1

- 为不同的设备设置不同的header参数：

```
sub vcl_recv {
    if (req.http.User-Agent ~ "iPad" ||
          req.http.User-Agent ~ "iPhone" ||
          req.http.User-Agent ~ "Android") {
          set req.http.X-Device = "mobile";
    } else {
          set req.http.X-Device = "desktop";
    }
}
```

- 想要取消访问/images的request的cookie：

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

常见VCL应用片断-2

n 通过ACL来控制能访问的ip地址

```
acl local {
    "localhost";
    "192.168.1.0"/24; /* and everyone on the local network */
    ! "192.168.1.23"; /* except for the dialin router */
}
sub vcl_recv {
  if (req.request == "PURGE") {
    if (client.ip ~ local) {
       return(lookup);
    }    } }
sub vcl_hit {
   if (req.request == "PURGE") {
     set obj.ttl = 0s;
     error 200 "Purged.";
    } }
sub vcl_miss {
  if (req.request == "PURGE") {
    error 404 "Not in cache.";
  }
}
```

常见VCL应用片断-3

- 修改从后台服务器返回的对象的ttl：

```
sub vcl_fetch {
   if (req.url ~ "\.(png|gif|jpg)$") {
     unset beresp.http.set-cookie;
     set beresp.ttl = 1h;
  }
}
```

- 设置客户端发送的accept-encoding头只有gzip和default两种编码，gzip优先

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

常见VCL应用片断-4

n 简单的图片防盗链：

```
if (req.http.referer ~ "http://.*") {
    if (  !(req.http.referer ~ "http://.*sishuok\.com"
          || req.http.referer ~ "http://.*google\.com"
          || req.http.referer ~ "http://.*google\.cn"
    )) {
          set req.http.host = "www.sishuok.com";
          set req.url = "/static/images/logo.gif";
    }
    return (lookup);
}
```

管理Varnish-1

- 在启动Varnish的时候，已经通过-T的参数来指定了管理Varnish的ip和端口，现在就可以连接这个ip和端口来进行Varnish的管理，有两种连接方式：
  第一种：telnet ip port 的方式，会进入一个纯文本的命令行管理界面
  第二种：varnishadm -T ip:port的方式，进入varnish的命令行管理界面
- 两种方式都是进入Varnish Command Line Interface，简称CLI，在CLI里面可以控制和改变大多数Varnish运行的参数和配置，而无须中断Varnish的服务。
- CLI主要能完成如下的功能：
  配置：能上传、修改和删除VCL文件
  参数：能查看和修改各种参数
  清除缓存：可以清除Varnish中的缓存内容
  进程管理：可以启动或者停止缓存子进程
- CLI中可用的命令：

backend.list：列出定义中的backend，包括它们的健康状态

backend.set_health matcher state：为某个backend设置健康状态，当你想要把某个backend从使用序列中移出的时候，这个命令很有用

管理Varnish-2

ban field operator argument [&& field operator argument [...]]：使得匹配ban表达式的内容从缓存中清除。

- 一个ban表达式包含一到多个条件，一个条件由一个字段、一个操作符、一个参数构成，多个条件之间可以用“&&”来表示 and的关系
- 字段可以是任意的VCL变量，如：req.url，req.http.host or obj.http.set-cookie等
- 操作符有：==表示等、~匹配正则表达式、>、<、！等
- 参数可以是一个用双引号引起来的字符串，也可以是正则表达式、或者数字。数字后面开可以跟“KB”，“MB”，“GB” 或“TB”等。
- 示例如下：

  清除请求url完全匹配“/news”的缓存内容： ban req.url == "/news“

  清除请求url不能以.ogg结尾，同时这对象的大小不能大于10M 的缓存内容

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

  清除host为example.com或者www.example.com，同时从backend获得set-cookie头里面包含userid=1663 ， 的缓存内容

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

管理Varnish-3

ban.list：内容ban列表。

1：如果缓存的内容比ban旧，那么这个缓存内容会被标示成banned，不能再使用这个缓存内容，需要从backend获取内容。

2：如果ban的表达式比所有的缓存对象都旧的话，它将从ban的list里面去掉

3：ban.list输出的格式如下：

0x7fea4fcb0580 1303835108.618863 131G req.http.host ~ www.myhost.com && req.url ~ /some/url

第一个字段是ban的地址，第二个字段是时间戳，第三个字段表示有多少个对象受此ban的影响（通常在debug时才有效），G表示这个ban已经Gone，也就不再有效了

后面的内容才是真正的ban的内容。

ban.url regexp ：要ban的url，匹配这个规则的url，缓存立即失效，注意一点：在这个url里面host是会被忽略的。

help [command]：显示命令的帮助，不写命令就显示所有的命令

param.set：设置param的值

管理Varnish-4

param.show [-l] [param]：显示param以及他们的值。-l的话，会带着命令的简短描述

ping [timestamp]：ping Varnish的缓存进程，保持连接是活动的

quit：退出CLI管理

start：如果Varnish的缓存进程没有启动的话，启动它

status：检查Varnish的缓存进程的状态

stop：停止Varnish的缓存进程

storage.list：列表显示定义的storage backends

vcl.discard configname：废弃某个配置，注意，如果这个配置的引用不为0的话，简单的说就是已经使用了，这个命令无效

vcl.inline configname vcl：使用VCL来创建一个新的配置

vcl.list：列出可用的配置，以及参照使用他们的数量

vcl.load configname filename：创建一个新的配置，配置的名字是configname，内容是filename指定的文件内容

vcl.show configname：显示配置的具体内容

vcl.use configname：使用哪一个配置

Varnishadm命令

- 命令语法：

  varnishadm [-t timeout] [-S secret_file] [-T address:port] [-n name] [command [...]]

  -t timeout：等待一个操作完成的时间，单位秒

  -S secret_file：确定一个认证的安全文件

  -T address：port：连接到管理接口的地址和端口，在启动Varnish时指定的

  -n name：连接到管理接口的名字， 在启动Varnish时指定的

- 后面还可以直接跟要执行的命令，如果不跟命令，就会进入CLI的界面

Varnishd命令-1

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

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

- 选项说明如下：

-a address：[:port][,address]

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

-b host[:port] ：指定后端服务器的地址和接口，如果没有接口，默认是8080

-C ：编译VCL代码成C语言，然后退出，指定VCL文件用-f参数

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

-F ：在前端运行

-f config ：使用指定的VCL配置文件代替系统默认的

-g group ： 指定Varnishd子进程使用的户组

Varnishd命令-2

-h type[,options] ： 指定hash算法
-i identity ： 指定Varnishd server的身份
-l shmlogsize ： 指定shmlogfile的大小，单位可以使用‘k’和‘m’，默认是80M，不要指定比80M小的值
-n name ： 为这个实例指定一名字
-P file ： 指定pidfile，用于保存PID信息
-p param=value ： 设定指定参数的值
-S file ： 访问管理端口用到的安全认证文件的路径
-s [name=]type[,options] ： 使用指定的存储后端，可以多次使用此选项指定不同的存储类型
-T address[:port] ： 提供一个管理接口的地址和端口
-t ttl ： 给缓存中的内容指定最小的ttl
-u user ： 指定运行varnishd子进程的用户
-V ： 显示Varnishd的版本，然后退出
-w min[,max[,timeout]] ： 指定线程最小和大空闲时间，这是一个设置thread_pool_min、thread_pool_max和thread_pool_timeout的捷径。
如果只有一个值被指定，那么thread_pool_min和thread_pool_max都是用这 个值。
Thread_pool_timeout会失效

Varnishd命令-3

- 可用的Hash算法如下：

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

  classic[,buckets]：一个标准的hash table，默认使用这个

  critbit：一个自适应的树结构，相比传统的B-tree，critbit-tree几乎不用锁，性能更好

- 可用的Storage Type如下：

Malloc[,size]

malloc是基于内存的存储。Size参数指定分配给Varnish的最大内存，默认单位是byte，你可以指定单位，如：K、M、G、T，默认没有限制。

file[,path[,size[,granularity]]]

采用文件来存储对象，然后使用mmap来把文件映射到内存，这是Varnish缺省的存储方式。Path指定存储的路径和文件名，默认是/tmp。Size指定文件的最大尺寸，默认是byte，同样可用K、M、G、T等单位，还可以用%来指定使用系统空闲空间的百分比，默认是50%。

如果文件已经存在，varnish会缩小或放大backing文件到指定的size。注意：如果Varnishd需要创建或者扩大一个文件，它不会去预分配需要增加的空间，而之前又没有设置好空间的话，可能会产生碎片，影响性能。在分配文件前使用dd命令来创建文件，可以尽量减少碎片。

granularity 这个参数指定了间隔的尺寸，默认是字节，可以指定单位，但不能用%。默认的间隔尺寸是VM的page大小，如果你有太多的小文件，这个值可以小一点。

文件的性能通常取决于设备的是写入速度，还有使用上的寻找时间。

Varnishd命令-4

persistent,path,size

这种类型还处于试验性质，类似于文件存储的类型，它能更安全的保障对象能正确存储，就算是varnish是正常或者非正常关闭的情况。这里就不多讲了。

- Transient Storage

如果你把一个Storage的name设置成Transient，它将用来存储瞬时对象，缺省的Varnish将会使用无限制的malloc来存储它们。

- 运行期参数，就是在管理界面，通过param.show看到的参数

运行参数有速记标志，避免重复出现，该标记的含义：

experimental

对这个参数，我们没有固定的值来说明好不好，欢迎观察和反馈这个值

delayed

这个值可以在不工作的时候改变，但是不会立即生效

restart ：工作进程会被停止，重新启动

reload ：VCL程序会被重新装载

注意：在32位系统上，有一些默认值，比如：workspace_client (=16k), thread_pool_workspace (=16k), http_resp_size (=8k), http_req_size (=12k), gzip_stack_buffer (=4k) , thread_pool_stack (=64k) ，可以减少这些值以保持虚拟空间

Varnishd命令-5

acceptor_sleep_decay

Default: 0.900 Flags: experimental

如果我们的文件描述，或者工作线程等资源耗尽，接收器在两次间隔中会休眠，这个参数减少成功接收的休眠时间（0.9 = reduce by 10%）

acceptor_sleep_incr

Units: s Default: 0.001 Flags: experimental

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

acceptor_sleep_max

Units: s Default: 0.050 Flags: experimental

如果我们的文件描述，或者工作线程等资源耗尽，接收器在两次接收间隔中会休眠，这个参数设置最大的休眠时间

auto_restart

Units: bool Default: on

如果子进程宕了，自动重启

ban_dups

Units: bool Default: on

发现并去掉重复的bans

Varnishd命令-6

ban_lurker_sleep

Units: s Default: 0.01

设置ban lurker线程在成功执行一次ban后的休息时间。默认是1秒，设置为0将禁用ban lurker。

between_bytes_timeout

Units: s Default: 60

设置在接收数据时，两个字节间的超时时间，值为0，表示不会超时。这个参数不能用在pipe模式中

cc_command

Default: exec gcc -std=gnu99 -pthread -fpic -shared -Wl,-x -o %o %s Flags: must_reload

编译C源码的参数，%s是源文件的名字，%o是输出文件名字

cli_buffer

Units: bytes Default: 8192

CLI输入的缓冲区大小。如果使用很大的vcl文件的话，需要加大这个值。注意要使用-p参数使其生效

cli_timeout

Units: seconds Default: 10

管理员对CLI请求的超时时间

clock_skew

Units: s Default: 10

设置后段服务器和valish之间，多少时差是可以接受的

Varnishd命令-7

connect_timeout

Units: s Default: 0.7

设置连接后端服务器默认的超时时间，vcl的配置可以覆盖这个值。

critbit_cooloff

Units: s Default: 180.0

设置critbit hasher在cooloff列表保存deleted objheads的时间

default_grace

Units: seconds Default: 10 Flags: delayed

设置grace的缺省时间，varnish将会在对象过期后延迟递交，好让其他线程做一个新的拷贝。

default_keep

Units: seconds Default: 0 Flags: delayed

设置保存一个无用对象的时间。这意味着对象从缓存中删除的时间=ttl+grace+keep

default_ttl

Units: seconds Default: 120

如果backend和vcl都没有给对象分配ttl的话，这个设置将生效。已经在缓存中存在的对象，在它们重新从后台获取数据前不受影响，强制他们生效，可以使用"ban.url “

diag_bitmap

Units: bitmap Default: 0

设置Bitmap控制诊断码，具体的可以从文档上查看

Varnishd命令-8

```
expiry_sleep
    Units: seconds    Default: 1
    设置expiry线程的休息时间
fetch_chunksize
    Units: kilobytes    Default: 128    Flags: experimental
    设置fetcher使用的缺省chunksize，这个值应该比多数对象大。
fetch_maxchunksize
    Units: kilobytes    Default: 262144    Flags: experimental
    分配给存储的最大chunksize，分配过大会引起延迟和碎片。
first_byte_timeout
    Units: s    Default: 60
    从后端获取第一个数据的时间。我们只等待这么长时间，超时就放弃，0表示永不放弃，vcl的配置可以
    覆盖这个值，这个值在pipe模式无效。
group
    Default: magic    Flags: must_restart
    使用哪个没有特权的组来运行此进程。
gzip_level
    Default: 6    Gzip compression level: 0=debug, 1=fast, 9=best
    Gzip compression level: 0=debug, 1=fast, 9=best
gzip_memlevel
    Default: 8
    Gzip内存level 1=slow/least, 9=fast/most compression. Memory impact is 1=1k, 2=2k, ...
    9=256k.
```

Varnishd命令-9

gzip_stack_buffer
    Units: Bytes   Default: 32768   Flags: experimental
    Gzip缓存区的大小
gzip_tmp_space
    Default: 0   Flags: experimental
    在哪儿给gzip/gunzip分配临时空间，0－malloc；1-session workspace；2-thread workspace
    如果你有很多gzip/gunzip活动,建议使用workspace以减少内存耗费，要知道gzip需要256+KB，gunzip
    需要32+KB(64+KB if ESI processing)
gzip_window
    Default: 15
    Gzip窗口大小8=least, 15=most compression. Memory impact is 8=1k, 9=2k, ... 15=128k.
http_gzip_support
    Units: bool   Default: on   Flags: experimental
    开启gzip支持，Varnish将会在存储到缓存前压缩对象。
http_max_hdr
    Units: header lines   Default: 64
    可以处理的最大的Http Header。
http_range_support
    Units: bool   Default: on
    开启支持HTTP Range headers，以支持多线程断点续传。
http_req_hdr_len
    Units: bytes   Default: 8192
    最大能接受的客户端请求头的大小。

Varnishd命令-10

```
http_req_size
    Units: bytes    Default: 32768
    最大能处理的客户端请求的大小。
http_resp_hdr_len
    Units: bytes    Default: 8192
    最大能处理的从backend返回的响应头的大小。
http_resp_size
    Units: bytes    Default: 32768
    最大能处理的从backend返回的响应的大小。
idle_send_timeout
    Units: seconds    Default: 60    Flags: delayed
    等待发送数据的时间，如果超时还是没有数据发送，session会关闭。
listen_address
    Default: :80    Flags: must_restart
    监听的地址，可接收的表达：host, host:port, :port
listen_depth
    Units: connections    Default: 1024    Flags: must_restart
    监听队列的深度
log_hashstring
    Units: bool    Default: on
    是否记录这hash内容到共享内存日志
```

Varnishd命令-11

```
log_local_address
    Units: bool    Default: off
    日志是否记录本地ip的tcp连接
lru_interval
    Units: seconds    Default: 2    Flags: experimental
    目标对象移到LRU列表前的Grace时长
max_esi_depth
    Units: levels    Default: 5
    esi:include进程的最大深度
max_restarts
    Units: restarts    Default: 4
    一个请求的最大重试次数。
nuke_limit
    Units: allocations    Default: 50    Flags: experimental
    在空间中保存对象body的最大对象数量
pcre_match_limit
    Default: 10000
    内部调用pcre_exec()的次数限制
pcre_match_limit_recursion
    Default: 10000
    内部递归调用pcre_exec()的次数限制
```

Varnishd命令-12

```
ping_interval
    Units: seconds    Default: 3    Flags: must_restart
    子进程ping主进程的时间间隔，0表示禁止ping
pipe_timeout
    Units: seconds    Default: 60
    PIPE会话的空闲超时时间，如果超时还没有数据发送的话，session会关闭
prefer_ipv6
    Units: bool    Default: off
    如果后端支持ipv4和ipv6，那么偏好使用ipv6
queue_max
    Units: %    Default: 100    Flags: experimental
    允许的队列长度，用百分比表示，设置的是 请求队列与worker进程的百分比
rush_exponent
    Units: requests per request    Default: 3    Flags: experimental
    为完成一个请求对象准备多少parked request
saintmode_threshold
    Units: objects    Default: 10    Flags: experimental
    Saint模式在超时前可以容纳的对象数目，0表示禁用Saint模式
send_timeout
    Units: seconds    Default: 600    Flags: delayed
    发送的超时时间
```

Varnishd命令-13

```
sess_timeout
    Units: seconds    Default: 5
    每个session保持的空闲时间
sess_workspace
    Units: bytes    Default: 65536    Flags: delayed
    分配各每个HTTP协议的session的空间大小，最小是1024 bytes
session_linger
    Units: ms    Default: 50    Flags: experimental
    一个session的linger（逗留，慢慢消失）的时间
session_max
    Units: sessions    Default: 100000
    session的最大数量
shm_reclen
    Units: bytes    Default: 255
    SHM日志的最大数量，最大是65535 bytes
shm_workspace
    Units: bytes    Default: 8192    Flags: delayed
    分配给worker线程中shm日志的空间，最小是4096 bytes
Shortlived
    Units: s    Default: 10.0
    对象最小的存活时间，如果分配各对象的TTL比这个值小，就保存在瞬时存储里面
```

Varnishd命令-14

```
syslog_cli_traffic
    Units: bool    Default: on
    记录CLI的syslog(LOG_INFO)
thread_pool_add_delay
    Units: milliseconds    Default: 2
    创建新线程前等待的时间，设置太长，会造成工作线程不足，太短会造成工作线程堆积
thread_pool_add_threshold
    Units: requests    Default: 2    Flags: experimental
    创建工作线程的阀值，设置太小，会造成工作线程过量，太大会造成线程不足
thread_pool_fail_delay
    Units: milliseconds    Default: 200    Flags: experimental
    线程池在一个线程失败，创建新线程前的等待时间
thread_pool_max
    Units: threads    Default: 500    Flags: delayed, experimental
    每个线程池能容纳的最大线程数
thread_pool_min
    Units: threads    Default: 5    Flags: delayed, experimental
    每个线程池最小的线程数，最小值为2
thread_pool_purge_delay
    Units: milliseconds    Default: 1000    Flags: delayed, experimental
    在purge线程间等待的时间，最小值为100
```

Varnishd命令-15

```
thread_pool_stack
    Units: bytes    Default: -1    Flags: experimental
    Worker thread stack size
thread_pool_timeout
    Units: seconds    Default: 300    Flags: delayed, experimental
    线程池中的线程数小于thread_pool_min，在关闭线程池前等待的时间，最小是1秒
thread_pool_workspace
    Units: bytes    Default: 65536    Flags: delayed
    分配各工作线程的workspace大小，最小是1024 bytes
thread_pools
    Units: pools    Default: 2    Flags: delayed, experimental
    线程池的数量
thread_stats_rate
    Units: requests    Default: 10    Flags: experimental
    处理多少个线程，然后统计一次
User
    Default: magic    Flags: must_restart
    运行进程的用户，通常和group一起配置
vcc_err_unref
    Units: bool    Default: on
    未引用vcl对象的错误结果
```

Varnishd命令-16

vcl_dir
    Default: /usr/local/etc/varnish
    缺省的VCL文件路径和名字
vcl_trace
    Units: bool   Default: off
    开启跟踪VCL执行情况。
vmod_dir
    Default: /usr/local/lib/varnish/vmods
    定义VCL modules的路径

Varnish的Log-1

- Varnish将日志记录到共享内存片段，而不是记录到一个普通文件中。当记录到内存片段的最后处，会再从头开始记，覆写老数据。这比记录到文件要快的多，不需要磁盘空间。
- Varnishlog是一个用来查看Varnish日志的程序

  1：启动varnishlog，会看到主进程的基本信息

  2：然后重新刷新浏览器访问页面，会看到访问的日志信息

  3：格式说明：第一列可以是任意的数字，它代表具体的请求。数字相同，表示他们是同属于一个HTTP事务的。第二列是日志信息的标签。所有的日志条目都是用一个标签去标记，该标签代表何种行为被记录。以Rx开头的标签代表varnish正在接受数据，Tx代表正在发送数据。

  第三列表示数据的是来自或者要发送给客户（c），另外，还有为b的情况，代表数据来自或要发送给后端（b）。第四列是被记录的数据。
- 日志过滤：可以使用varnishlog去过滤，基本的选项如下：

  -b 只显示varnish和后端服务器之间通信的记录条。当你想优化缓存命中率的时候，非常有用。

  -c 和-b类似，只是针对与客户端的通信情况。

  -i *tag* 只有显示带有特定标签的行。”varnishlog -i SessionOpen”将只显示新会话的情况。注意标签是大小写敏感的。

  -I 通过正则表达式过滤数据，并显示匹配行。”varnishlog -c -I RxHeader -I Cookie”，将显示所有来自客户端的cookie头信息。

  -o 根据请求id，将记录条目分组，如果要写到一个文件里面，使用-w选项

Varnish的Log-2

- Varnishlog还有如下选项：

  -a 当把日志写入文件时，采用追加的方式，而不是覆盖
  -C 匹配正则表达式的时候，忽略大小写
  -D 以进程方式运行
  -d 启动时处理旧的日志，通常valishlog只会在进程写入日志后启动
  -k num 只显示开头的num个日志记录
  -n 指定varnish实例的名字，用来获取日志，默认是主机名
  -P file 记录PID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 num 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，而不是显示他们，如果没有-a参数的话，就会覆盖文件。如果在写文件的时候，接收到sighup的信号，他会创建一个新的文件
  -X regex 排除匹配正则表达式的日志
  -x tag 排除匹配tag的日志
  如果-o选项被指定，需要使用正则表达式和tag来制定需要的日志

Varnish的Log-3

- 当前定义的tag：

  Backend、BackendClose、BackendOpen、BackendReuse、BackendXID、CLI、ClientAddr、Debug、Error、ExpBan、ExpKill、ExpPick、Hit、HitPass、HttpError、HttpGarbage、Length、ObjHeader、ObjLostHeader、ObjProtocol、ObjRequest、ObjResponse、ObjStatus、ObjURL、ReqEnd、ReqStart、RxHeader、RxLostHeader、RxProtocol、RxRequest、RxResponse、RxStatus、RxURL、SessionClose、SessionOpen、StatAddr、StatSess、TTL、TxHeader、TxLostHeader、TxProtocol、TxRequest、TxResponse、TxStatus、TxURL、VCL_acl、VCL_call、VCL_return、VCL_trace、WorkThread

- 示例：

  输出日志到一个文件：

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

  读取一个日志文件，然后显示首页的请求

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

Varnishtop

- varnishtop工具读取共享内存日志，并且显示一个持续更新的最常见的记录条的列表。
  下面的参数可以使用：
  -1 代替连续不断的更新和显示，只显示一次然后退出。暗示：-d
  -b 包含指定后端服务器的日志，如果没有使用-b或-c，那么varnishtop担当这两种角色
  -C 使用正则表达式的时候忽略大小写
  -c 包含指定客户端的日志，如果没有使用-b或-c，varnishtop但当这两种角色
  -d 启动的时候使用旧的日志记录，通常varnishtop只读取启动以后生成的日志
  -f 只显示日志的第一列
  -I regex匹配正则表达式的日志，如果没有使用-i或者-I，那么所有的日志都会匹配
  -I tag匹配指定的tag，如果没有使用-i或者-I，那么所有的日志都会被匹配
  -n 指定varnish实例的名字，用来获取日志，如果没有指定，默认使用主机名
  -r file读入日志文件，代替共享内存
  -V 显示版本号，然后退出
  -X regex排除匹配表达式的日志
  -x tag排除匹配tag的日志
- 下面这个例子显示和更新收到的URL：varnishtop -i RxURL
- 下面的例子显示连续不断的更新用户使用的用户代理：

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

Varnishhist和Varnishsizes

- varnishhist工具读取共享内存日志，并且提供不断更新的直方图，用以展示它们处理的最近的N个请求的分布。N值和垂直刻度在左上角显示。水平刻度是对数的。命中用管道符号（“|”）标记，未命中使用哈希符号（“#”）标记。

  下面的选项是可用的：

  -b 分析指定后端服务器的日志
  -C 忽略正则表达式的大小写
  -c 分析指定客户端的日志
  -d 在启动过程中处理旧的日志，通常只会在进程写入日志后启动
  -I regex匹配正则表达式的日志，如果没有使用-i或者-I，那么所有的日志都会匹配
  -I tag匹配指定的tag，如果没有使用-i或者-I，那么所有的日志都会被匹配
  -n 指定varnish实例的名字，用来获取日志，如果没有指定，默认使用主机名
  -r file读入日志文件，代替共享内存
  -V 显示版本号，然后退出
  -w delay等待更新的延迟时间，默认是1秒
  -X regex导入匹配表达式的日志
  -x tag导入匹配tag的日志
- varnishsizes和varnishhist相似，但它一次主请求只算一次，而varnishhist是主请求和连带请求分开计算次数

Varnishnsca

- NSCA格式：美国国家超级计算技术应用中心（NSCA）公用格式是常用的标准日志格式，有很多日志分析工具可以对NSCA格式进行分析处理

- Varnishnsca 就是Varnish提供的，用来按照Apache/NSCA格式显示日志，这个命令的选项基本类似于Varnishlog，下面是比较有特色的选项：

  -f　在日志输出中用X-Forwarded-For HTTP header来替换client.ip

  -F　指定日志的格式，当前是：%h %l %u %t "%r" %s %b "%{Referer}i" "%{User-agent}i“

  支持转义字符 \n 和 \t 。具体支持的格式请参看官方文档。

  -m tag:regex　仅仅罗列出tag匹配正则表达式的记录。多个-m选项之间的关系是and的关系

Varnishreplay

- Varnishreplay工具会从日志文件中读取内容，然后重新填充到缓存里面，相当于一个回放的功能。可以用来恢复缓存或做各种测试。
- 参数如下：

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

  -D　打开debug模式。

  -r file　使用文件里的语法分析日志，这个参数是强制的。
- 应用场景

  1：缓存数据迁移

  将新varnish的backend设定为原来的varnish，基于varnishreplay拷贝缓存数据

  2：原来的varnish宕机，可以使用这个命令把缓存数据恢复回来

  分析varnishlog，得到缓存索引（uri），向后端预请求uri。

Varnish的共享内存

- Varnish使用共享内存进行日志记录和统计，因为它是更快和更有效的。但棘手的是他不定期写日志文件，对于Varnish的共享内存，你需要了解：
  1：当你使用append模式打开一个文件，操作系统可以保证写不会覆盖文件中的现有数据
  2：对于共享内存日志，我们从内核得不到任何帮助，writer需要保证不会相互影响，他们还需要保证读数据是可行和安全的
  3：在varnish的worker过程中，使用互斥锁来保证一致性，包括日志记录和统计的计数器。对于只读的数据是不用加锁的。
  4：当varnishd启动的时候，如果它发现存在一个共享内存的文件，而且可以安全的读取到master_pid数据，它就会检查进程是否正在运行，如果没有运行，将会报错

- 分配：共享内存文件内的section是动态分配的，比如添加一个新的backend的时候。改变会反映到分配链表上，开始“alloc_seq” header的值是0，当改变后，它会是另外一个值
- 释放：当共享内存文件内的section是freed，它的类将会变成“cool”状态至少10秒，以让程序来检查alloc_seq header的值，以及类的变化

规划Varnish的缓存大小

- 给Varnish选择多少内存，是个很艰巨的问题，需要考虑以下事情：

1：应用的热门数据集有多大？

对一个门户或者新闻站来说，这个数据集可能就只是首页和它相关内容的大小。这里包括的两部分，一部分是只首页本身的文字图片内容，另一部分是首页会链接到的页面或对象（比如图片），这个很容易理解，首页的内容是最可能被点击的，命中率也会很高。

2：产生一个对象的花费有多大？

有时候，如果从后端返回并不太消耗资源，同时你的内存又有限的话，就应该缓存一部分图片，而不是去缓存所有图片。

3：使用varnishstat或其他工具监控n_lru_nuked计数器。

如果你有很多LRU活动的话，那么你的缓存正因空间限制在清除对象，此时你就要考虑增加缓存大小了。

缓存任何对象都会携带保存在实际存储区域之外的开销。所以，即便你指定-s malloc,16G，varnish可能实际使用了两倍。Varnish中每个对象的花销大概是1k。所以，如果在你的缓存中有很多小对象的话，花销是非常大的。

Varnishstat统计信息

- Varnishstat工具显示一个运行的varnished实例的相关统计数据
- 参数如下：

  -1 只显示一次就退出

  -f 使用逗号分隔字段列表来显示，使用“^”开始排除列表。

  -l 监听有效的列使用-f参数

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

  -V 显示版本号，然后退出

  -w delay 刷新间隔时间，默认1s

  -x 显示xml格式

  -j 显示json格式
- 显示中每列的含义，从左到右：

  1：值 2：从最后一秒更新以来的每秒的一个平均值，或者一个不能计算的周期

  3：从进程开始到现在每秒的平均值，或者是一个不能计算的周期。

  4：描述

  当使用-1选项，输出列的含义,从左到右：

  1：特征名字 2：值

  3：从进程开始到现在每秒的平均值，或者是一个不能计算的周期。

  4：描述

提高Varnish的命中率-1

- Varnish的默认缓存策略是偏向保守的（可以通过配置改变）

  它默认只缓存get请求和HEAD请求，不缓存带有Cookie和认证信息的请求，也不会缓存带有Set-Cookie或者有变化的头信息的响应。

  Varnish也会检查请求和响应中的Cache-Control头信息，这个头信息中会包含一些选项来控制缓存行为。当Cache-control中Max-age的控制和默认策略冲突时，varnish不会单纯的根据Cache-control信息就改变自己的缓存行为。

  例如：Cache-Control: max-age=n，n为数字，如果varnish收到web服务器的响应中包含max-age，varnish会以此值设定缓存的过期时间（单位：秒），否则varnish将会设置为参数配置的时间，默认为120秒。
- 提高Varnish命中率的根本方法，就是仔细规划请求和应答，并自定义缓存策略，通过VCL来配置自己想要缓存的内容，并主动设置对象的ttl，尽量不去依赖Http header
- 当然，如果使用默认策略的话，就需要好好的跟踪和分析Http header了
- 有一个提高命中率的简单方法，就是尽量加大ttl值，当然要在合理范围
- 查看日志，分析经常访问后端服务器的url，有一些常用的命令来帮助你：
  比如： varnishtop -i txurl 就可以看到请求后端的url

提高Varnish的命中率-2

- 通过日志辅助分析

varnish里内置了日志模块，可以在vcl文件最上边引用std库，以便输出日志：

```
import std;
```

然后在需要输出日志的地方，使用 std.log 即可。

比如你想跟踪哪些连接没有命中缓存，可以在vcl_miss函数中这样写：

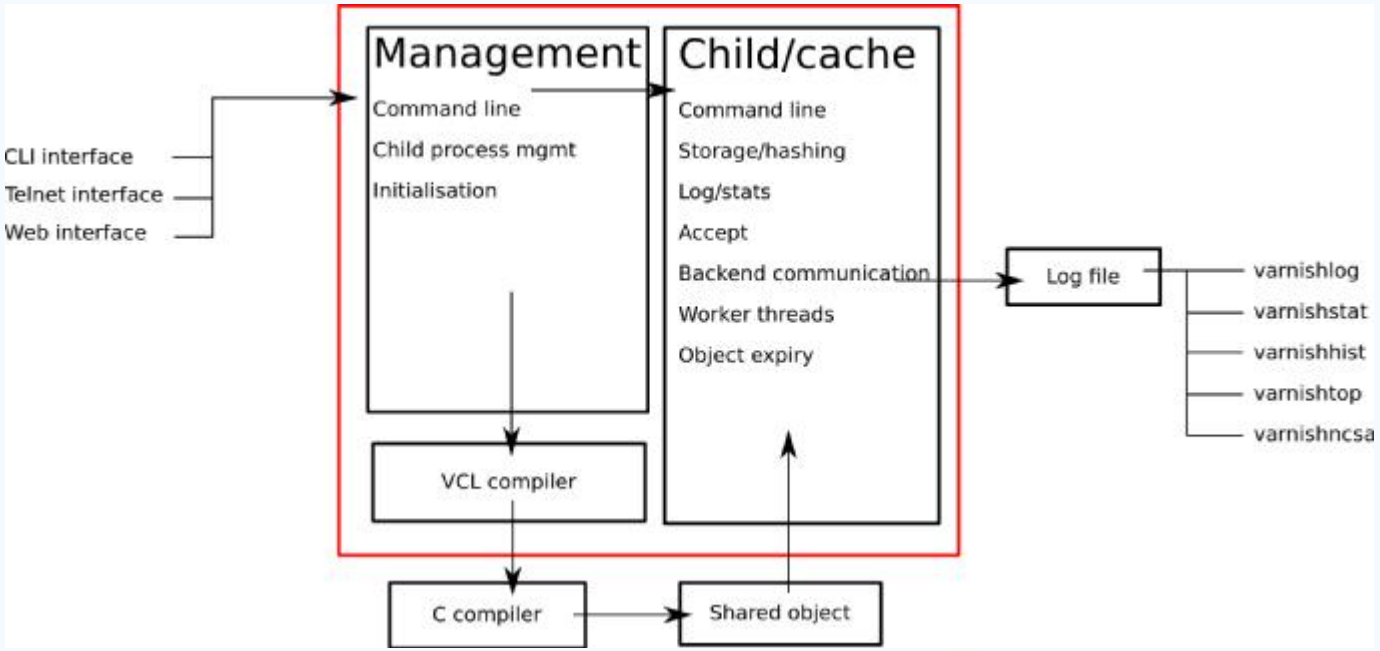
```
sub vcl_miss {
   std.log("url miss! the url=" + req.url);
   return (fetch);
}
```

启动varnish 后，通过 varnishlog工具跟踪打印出的日志

```
varnishlog -I log
```

Varnish的性能调优-1

- Varnish的性能调优分成两个部分，一个是硬件、操作系统和网络部分的优化；另外一个，也是最重要的一个，就是VCL的调优。
- 要进行硬件、操作系统和网络部分的优化，了解Varnish的进程和线程架构是有必要的，他们能帮助你更好的去调整优化，以及整合应用系统。
- Varnish的进程架构图如下：



Varnish的性能调优-2

- 管理进程(The management process)

  Varnish主要有两个进程，管理进程和子进程，管理进程负责：管理配置的变更（包括VCL和参数）、编译VCL、监控Varnish运行、初始化Varnish，以及提供命令行接口等。

  管理进程会每隔几秒钟检查一下子进程，如果发现子进程挂了，会kill掉然后重新启动一个。这些操作都会记录在日志里面，以利于你检查错误。

- 子进程(The child process)

  子进程包括几个不同类型的线程，包括但不限于：

  1：Acceptor线程：接受新的连接并代表它们

  2：Worker线程：一个会话一个线程，通常会使用数百个Worker线程

  3：Expiry线程：负责从缓存中清除旧的内容

- 工作区(Workspace)

  Varnish使用Workspace来减少多个线程间对于内存的竞争。Varnish有多个工作区，最重要的是session workspace，它通常用来操作会话数据。比如：修改www.sishuok.com到sishuok.com，以减少重复缓存。

  就算你的Session工作区有5G，使用了1000个线程，但实际使用的内存容量不会这么多，虚拟内存会是5个G，但是实际的内存是根据你实际的使用来的，内存控制器和操作系统会帮你管理内存，这个不用担心

Varnish的性能调优-3

- 和系统的通讯，子进程会使用共享内存日志访问文件系统，这意味着，如果一个线程需要记录日志，它只需要获得锁，然后写入内存，然后释放锁就可以了。另外每个线程都有一份日志数据的缓存，以减少锁的竞争
- 日志数据通常为80M，分成两个部分，第一个部分是计数器，第二个部分是请求数据。你可以使用日志工具来查看共享内存日志，因为日志记录并不会是原始的格式。
- 对于储存类型，如果不需要永久保持缓存的话，建议使用malloc，如果要永久保存，或者是要缓存的内容超过了物理内存的大小，那么使用file。

  另外一个要注意：缓存对象会有额外的开销，以保持对缓存的追踪，大概一个对象需要1k的额外开销，这也意味着，最终使用的内存会比你通过-s指定的内存大小要大一些。

- 通常对硬件、操作系统和网络部分的优化，主要就是相关参数的优化。参数调优的方法：通常是在CLI界面去调整，然后测试，如果ok的话，就把它们配置到配置文件里面去

Varnish的性能调优-4

- 线程模式

子进程是Varnish真正产生奇迹的地方，它包含一系列的线程来执行不同的任务，下面罗列几个常见的：

cache-worker：每个连接一个，负责处理请求

cache-main：一个，负责启动

ban lurker：一个，负责禁用的处理

acceptor：一个，负责接受新的连接

epoll/kqueue：可配置，缺省是2个，管理线程池

expire：一个，负责删除过期内容

backend poll：一个backend poll一个线程，用于健康检查

通常我们只需要配置cache-worker线程数，其他的线程可以不用管。

调整Varnish的时候，需要考虑预期的流量，线程模型允许你使用多个线程池，但时间和经验表明，只要你有2个线程池就够了，加入更多也不会提高性能。一些旧的资料会建议你一个cpu核运行一个线程池，这个已经过时了，现在只要2个就够了。

线程池相关的参数，前面已经都讲过了，最重要的是thread_pool_min和thread_pool_max。通常保持最少在500-1000个线程都是合适的，具体的可以根据varnishstat来查看n_wrk_queued，根据具体情况来配置

Varnish的性能调优-5

- 线程growth时间

  Varnish能支持数千个线程同时运行，但并不是所有的操作系统内核都能支撑，因此使用thread_pool_add_delay来保证每个线程间有一点延迟，现在已经不重要了，操作系统都比较成熟了，一般从20ms到2s

- 系统级参数

  随着Varnish越来越成熟，越来越少的参数需要调整了。sess_workspace是一个要调整的重要参数，它设置的是从客户端传入的Http 头的workspace大小：

  1：通常这个值从缺省的65536字节到10M

  2：要注意，它是虚拟的内存，不是实际使用的内存

- 另外一些可调优的参数就是各种时间，如：

  connect_timeout、first_byte_timeout、between_bytes_timeout、send_timeout、sess_timeout、cli_timeout

  通常情况下，默认的数值能满足大多数应用的需要，但你还是需要结合实际的应用进行调整。比如connect_timeout，默认是0.7秒，如果Varnish和backend是通过远程来连接和访问，这个时间可能就需要延长。

Varnish的性能调优-6

- 优化linx内核参数

内核参数是用户和系统内核之间交互的一个接口，通过这个接口，用户可以在系统运行的同时动态的更新内核配置，而这些内核参数是通过Linux Proc文件系统存在的，因此，可以通过对Proc文件系统进行调整，达到性能优化的目的。

以下参数是官方给出的一个配置，内容如下：

```
net.ipv4.ip_local_port_range = 1024 65536
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
net.ipv4.tcp_fin_timeout = 30
net.core.netdev_max_backlog = 30000。
net.ipv4.tcp_no_metrics_save=1
net.core.somaxconn = 262144
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 2
```

将以上内容添加到/etc/sysctl.conf文件中，然后执行如下命令，让设置生效：

```
sysctl  -p
```

Varnish的性能调优-7

- 对上面的每个选项含义解释如下：

net.ipv4.ip_local_port_range：指定外部连接的端口范围，默认32768到61000

net.core.rmem_max：该文件指定了接收套接字缓冲区大小的最大值，单位是字节

net.core.wmem_max：该文件指定了发送套接字缓冲区大小的最大值，单位是字节

net.ipv4.tcp_rmem：此参数与net.ipv4.tcp_wmem都是用来优化TCP接收/发送缓冲区，包含三个整数值：min，default，max

tcp_rmem：min表示为TCP socket预留用于接收缓冲的最小内存数量，default为TCP socket预留用于接收缓冲的缺省内存数量，max用于TCP socket接收缓冲的内存最大值

tcp_wmem：min表示为TCP socket预留用于发送缓冲的内存最小值，default为TCP socket预留用于发送缓冲的缺省内存值，max用于TCP socket发送缓冲的内存最大值

net.ipv4.tcp_fin_timeout：用来减少处于FIN-WAIT-2连接状态的时间，使系统可以处理更多的连接。值为整数，单位为秒

举一个例子：在一个tcp会话过程中，在会话结束时，A首先向B发送一个fin包，在获得B的ack确认包后，A就进入FIN WAIT2状态等待B的fin包，然后给B发ack确认包。此参数就是用来设置A进入FIN WAIT2状态等待对方fin包的超时时间。如果时间到了仍未收到对方的fin包就主动释放该会话

Varnish的性能调优-8

net.core.netdev_max_backlog：该参数表示在每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数量

net.ipv4.tcp_syncookie：该文件表示是否打开SYN Cookie功能，tcp_syncookies是一个开关，该功能有助于保护服务器免受SyncFlood攻击。默认为0，这里设置为1

net.ipv4.tcp_max_orphans：表示系统中最多有多少TCP套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤儿连接就会复位并打输出警告信息。这个限制仅仅是为了防止简单的DoS攻击。此值不能太小。这里设置为262144

net.ipv4.tcp_max_syn_backlog：表示SYN队列的长度，预设为1024，这里设置队列长度为262144，以容纳更多等待连接

net.ipv4.tcp_synack_retries：这个参数用于设置内核放弃连接之前发送SYN+ACK包的数量。

net.ipv4.tcp_syn_retries：此参数表示在内核放弃建立连接之前发送SYN包的数量。