Nginx 性能优化实践

主讲：鲁班
时间：2019/8/30

概要：

1. Nginx 反向代理与负载均衡
2. Nginx 实现高速缓存
3. Nginx 性能参数调优

一、Nginx 反向代理实现

知识点：

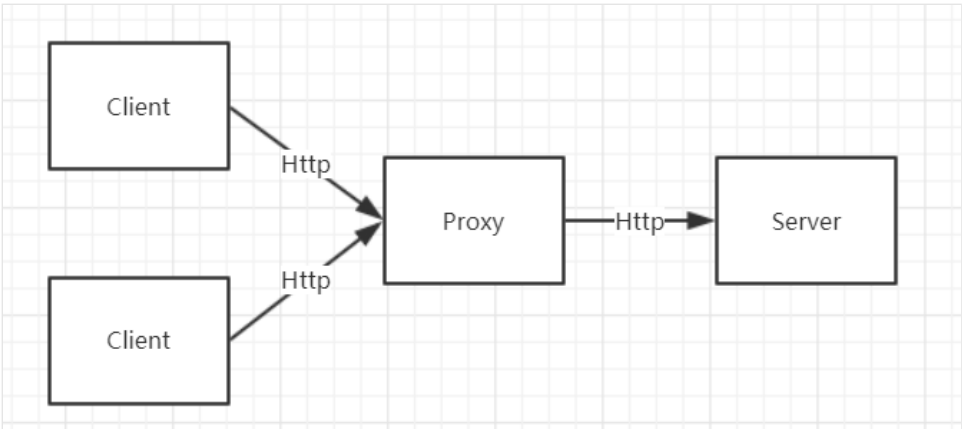
1. 反向代理基本配置
2. 负载均衡配置与参数解析
3. 负载均衡算法详解

1. 反向代理基本配置

提问：什么是反向代理其与正向代理有什么区别?

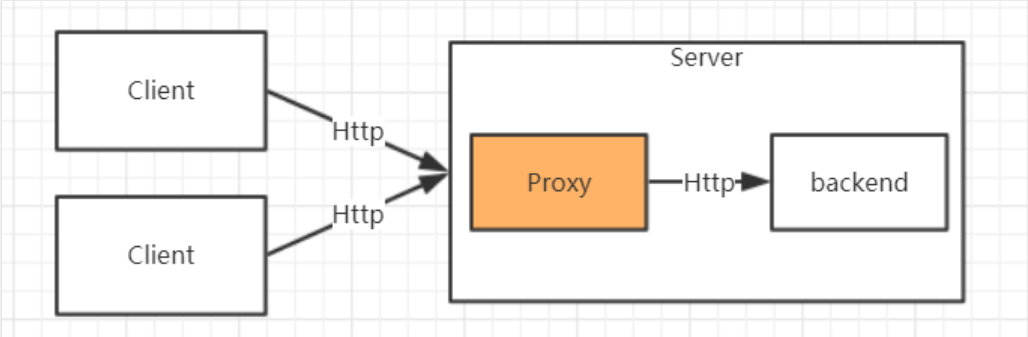
正向代理的概念：

正向代理是指客户端与目标服务器之间增加一个代理服务器，客户端直接访问代理服务器，在由代理服务器访问目标服务器并返回客户端并返回 。这个过程当中客户端需要知道代理服务器地址，并配置连接。



反向代理的概念：

反向代理是指 客户端访问目标服务器，在目标服务内部有一个统一接入网关将请求转发至后端真正处理的服务器并返回结果。这个过程当中客户端不需要知道代理服务器地址，代理对客户端而言是透明的。



反向代理与正向代理的区别

	正向代理	反向代理
代理服务器位置	客户端与服务都能连接的们位置	目标服务器内部
主要作用	屏蔽客户端IP、集中式缓存、解决客户端不能直连服务端的问题。	屏蔽服务端内部实现、负载均衡、缓存。
应用场景	爬虫、翻墙、maven 的nexus 服务	Nginx 、Apache负载均衡应用

Nginx代理基本配置

Nginx 代理只需要配置 location 中配置proxy_pass 属性即可。其指向代理的服务器地址。

```
# 正向代理到baidu 服务
location = /baidu.html {
          proxy_pass http://www.baidu.com;
}
```

```
# 反向代理至 本机的8010服务
location /luban/ {
     proxy_pass http://127.0.0.1:8010;
}
```

代理相关参数：

```
proxy_pass            # 代理服务
proxy_redirect off;   # 是否允许重定向
proxy_set_header Host $host; # 传 header 参数至后端服务
proxy_set_header X-Forwarded-For $remote_addr; # 设置request header 即客户端IP 地址
proxy_connect_timeout 90; # 连接代理服务超时时间
proxy_send_timeout 90; # 请求发送最大时间
proxy_read_timeout 90;  # 读取最大时间
proxy_buffer_size 4k;
proxy_buffers 4 32k;
proxy_busy_buffers_size 64k;
proxy_temp_file_write_size 64k;
```

2.负载均衡配置与参数解析

通过proxy_pass 可以把请求代理至后端服务，但是为了实现更高的负载及性能， 我们的后端服务通常是多个， 这个是时候可以通过upstream 模块实现负载均衡。

演示upstream 的实现。

```
upstream backend {
   server 127.0.0.1:8010 weight=1;
   server 127.0.0.1:8080 weight=2;
}
location / {
          proxy_pass http://backend;
}
```

upstream 相关参数:

- **service** 反向服务地址 加端口
- **weight** 权重
- **max_fails** 失败多少次 认为主机已挂掉则，踢出
- **fail_timeout** 踢出后重新探测时间
- **backup** 备用服务
- **max_conns** 允许最大连接数
- **slow_start** 当节点恢复，不立即加入,而是等待 slow_start 后加入服务对列。

3.upstream 负载均衡算法介绍

- **ll+weight**： 轮询加权重（默认）
- **ip_hash** : 基于Hash 计算 ,用于保持session 一至性
- **url_hash**: 静态资源缓存,节约存储，加快速度（第三方）
- **least_conn** ： 最少链接（第三方）
- **least_time** ： 最小的响应时间,计算节点平均响应时间，然后取响应最快的那个，分配更高权重（第三方）

二、Nginx 高速缓存

知识点:

1. 缓存案例分析
2. Nginx 静态缓存基本配置
3. 缓存更新

1、案例分析:

某电商平台商品详情页需要实现 700+ QPS，如何着手去做?

1. 首先为分析一下一个商品详情页有哪些信息

从中得出 商品详情页依懒了

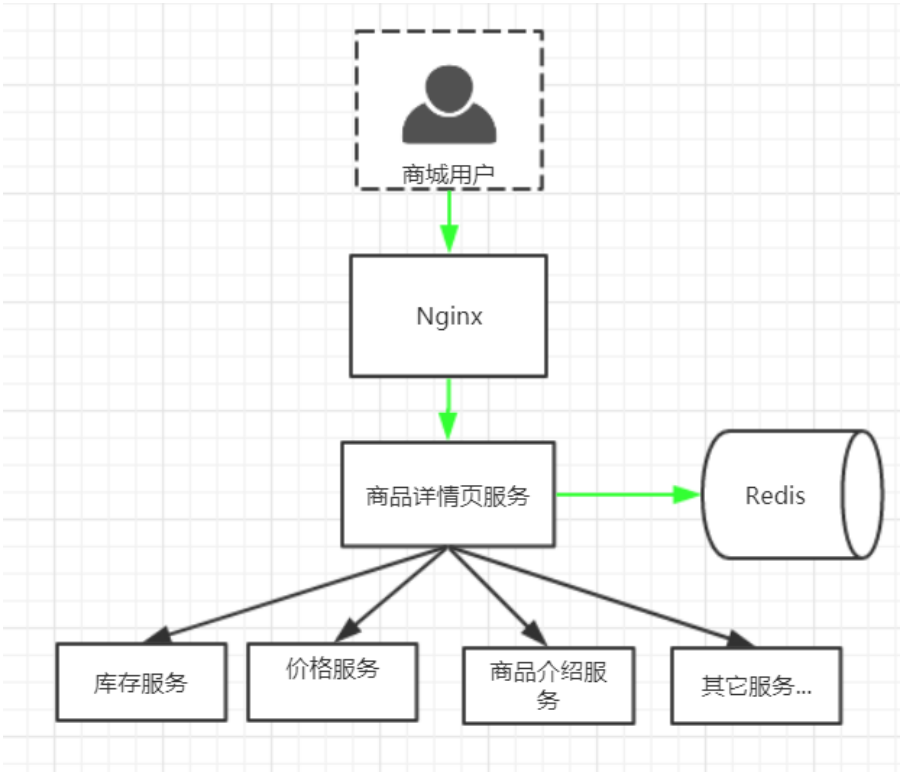
对于商品详情页涉及了如下主要服务：

- 商品详情页HTML页面渲染
- 价格服务
- 促销服务
- 库存状态/配送至服务
- 广告词服务
- 预售/秒杀服务
- 评价服务
- 试用服务
- 推荐服务
- 商品介绍服务
- 各品类相关的一些特殊服务

解决方案：

1. 采用Ajax 动态加载 价格、广告、库存等服务
2. 采用key value 缓存详情页主体html。

方案架构：

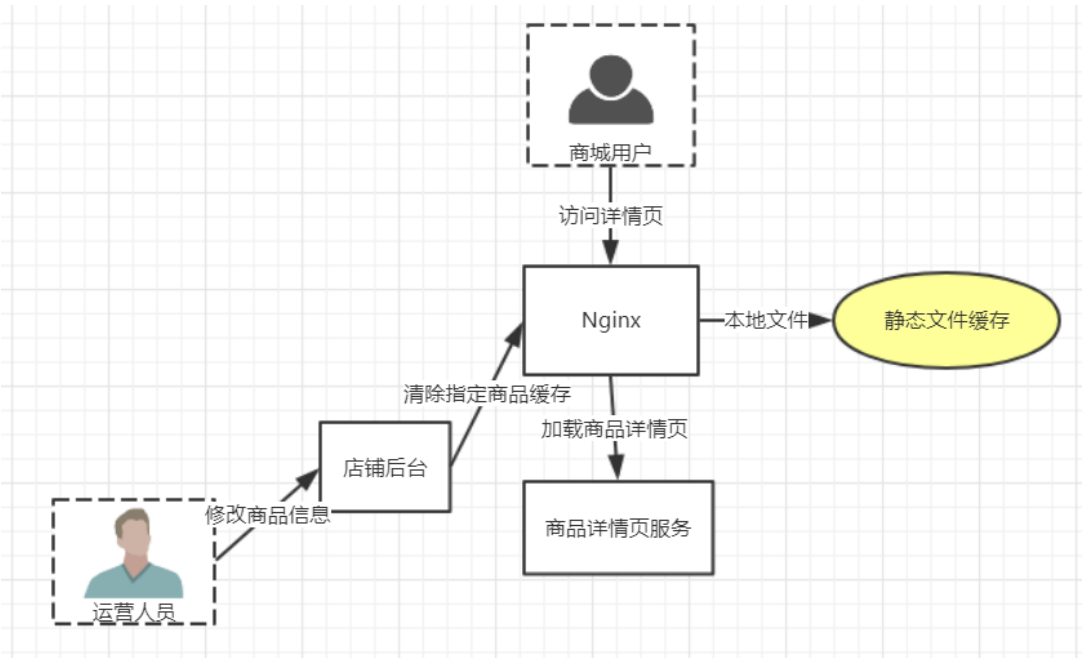


问题：

当达到500QPS 的时候很难继续压测上去。

分析原因：一个详情页html 主体达平均150 kb 那么在500QPS 已接近千M局域网宽带极限。必须减少内网通信。

基于Nginx 静态缓存的解决方案：



2.Nginx 静态缓存基本配置

一、在http元素下添加缓存区声明。

```
#proxy_cache_path 缓存路径
#levels 缓存层级及目录位数
#keys_zone 缓存区内存大小
#inactive 有效期
#max_size 硬盘大小
proxy_cache_path /data/nginx/cache_luban levels=1:2
keys_zone=cache_luban:500m inactive=20d max_size=1g;
```

二、为指定location 设定缓存策略。

```
# 指定缓存区
proxy_cache cache_luban;
#以全路径md5值做做为Key
proxy_cache_key $host$uri$is_args$args;
#对不同的HTTP状态码设置不同的缓存时间
proxy_cache_valid 200 304 12h;

```

演示缓存生效过程

- ☐ 配置声明缓存路径
- ☐ 为location 配置缓存策略
- ☐ 重启nginx（修改了）
- ☐ 查看缓存目录生成

缓存参数详细说明

父元素	名称	描述
http	proxy_cache_path	指定缓存区的根路径
	levels	缓存目录层级最高三层，每层1~2个字符表示。如1:1:2 表示三层。
	keys_zone	缓存块名称 及内存块大小。如 cache_item:500m 。表示声明一个名为cache_item 大小为500m。超出大小后最早的数据将会被清除。
	inactive	最长闲置时间 如:10d 如果一个数据被闲置10天将会被清除
	max_size	缓存区硬盘最大值。超出闲置数据将会被清除
location	proxy_cache	指定缓存区，对应keys_zone 中设置的值
	proxy_cache_key	通过参数拼装缓存key 如：$host$uriis_argsargs 则会以全路径md5值做做为Key
	proxy_cache_valid	为不同的状态码设置缓存有效期

3.缓存的清除：

该功能可以采用第三方模块 ngx_cache_purge 实现。

为nginx 添加 ngx_cache_purge 模块

```
#下载ngx_cache_purge 模块包 ,这里nginx 版本为1.6.2 purge 对应2.0版
wget http://labs.frickle.com/files/ngx_cache_purge-2.3.tar.gz
```

```
#查看已安装模块
./sbin/nginx -V
#进入nginx安装包目录 重新安装 --add-module为模块解压的全路径
./configure --prefix=/root/svr/nginx --with-http_stub_status_module --with-http_ssl_module --add-module=/root/svr/nginx/models/ngx_cache_purge-2.0
#重新编译
make
#拷贝 安装目录/objs/nginx 文件用于替换原nginx 文件
#检测查看安装是否成功
nginx -t
```

清除配置：

```
location ~ /clear(/.*) {
  #允许访问的IP
   allow            127.0.0.1;
   allow            192.168.0.193;
   #禁止访问的IP
   deny             all;
   #配置清除指定缓存区和路径(与proxy_cache_key一至)
   proxy_cache_purge    cache_item $host$1$is_args$args;
}
```

配置好以后 直接访问 ：

```
# 访问生成缓存文件
http://www.luban.com/?a=1
# 清除生成的缓存,如果指定缓存不存在 则会报404 错误。
http://www.luban.com/clear/?a=1
```

三、Nginx 性能参数调优

---

worker_processes number;
每个worker进程都是单线程的进程，它们会调用各个模块以实现多种多样的功能。如果这些模块确认不会出现阻塞式的调用，那么，有多少CPU内核就应该配置多少个进程；反之，如果有可能出现阻塞式调用，那么需要配置稍多一些的worker进程。例如，如果业务方面会致使用户请求大量读取本地磁盘上的静态资源文件，而且服务器上的内存较小，以至于大部分的请求访问静态资源文件时都必须读取磁盘（磁头的寻址是缓慢的），而不是内存中的磁盘缓存，那么磁盘I/O调用可能会阻塞住worker进程少量时间，进而导致服务整体性能下降。

每个worker 进程的最大连接数
语法：worker_connections number;
默认：worker_connections 1024

worker_cpu_affinity cpumask[cpumask……]

绑定Nginx worker进程到指定的CPU内核

为什么要绑定worker进程到指定的CPU内核呢？假定每一个worker进程都是非常繁忙的，如果多个worker进程都在抢同一个CPU，那么这就会出现同步问题。反之，如果每一个worker进程都独享一个CPU，就在内核的调度策略上实现了完全的并发。

例如，如果有4颗CPU内核，就可以进行如下配置：

worker_processes 4;

worker_cpu_affinity 1000 0100 0010 0001;

注意 worker_cpu_affinity配置仅对Linux操作系统有效。

Nginx worker 进程优先级设置

语法：worker_priority nice;

默认：worker_priority 0;

优先级由静态优先级和内核根据进程执行情况所做的动态调整（目前只有±5的调整）共同决定。nice值是进程的静态优先级，它的取值范围是－20～+19，－20是最高优先级，+19是最低优先级。因此，如果用户希望Nginx占有更多的系统资源，那么可以把nice值配置得更小一些，但不建议比内核进程的nice值（通常为－5）还要小

Nginx worker进程可以打开的最大句柄描述符个数

语法： worker_rlimit_nofile limit;

默认：空

更改worker进程的最大打开文件数限制。如果没设置的话，这个值为操作系统的限制。设置后你的操作系统和Nginx可以处理比“ulimit -a”更多的文件，所以把这个值设高，这样nginx就不会有“too many open files”问题了。

是否打开accept锁

语法：accept_mutex[on|off]

默认：accept_mutext on;

accept_mutex是Nginx的负载均衡锁，当某一个worker进程建立的连接数量达到worker_connections配置的最大连接数的7/8时，会大大地减小该worker进程试图建立新TCP连接的机会，accept锁默认是打开的，如果关闭它，那么建立TCP连接的耗时会更短，但worker进程之间的负载会非常不均衡，因此不建议关闭它。

使用accept锁后到真正建立连接之间的延迟时间

语法：accept_mutex_delay Nms;

默认：accept_mutex_delay 500ms;

在使用accept锁后，同一时间只有一个worker进程能够取到accept锁。这个accept锁不是堵塞锁，如果取不到会立刻返回。如果只有一个worker进程试图取锁而没有取到，他至少要等待accept_mutex_delay定义的时间才能再次试图取锁。