参考：<http://blog.csdn.net/yinwenjie/article/details/46742661>

# [架构设计：负载均衡层设计方案（3）——Nginx进阶](http://blog.csdn.net/yinwenjie/article/details/46742661)

上篇文章《**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "_blank)**设计：负载均衡层设计方案（2）——Nginx安装》（[http://blog.csdn.net/yinwenjie/article/details/46620711](http://blog.csdn.net/yinwenjie/article/details/46620711" \t "_blank)），我们介绍了Nginx的核心设计思想、基本安装和使用。本来准备继续介绍Nginx的几个使用特性，但是奈何博文篇幅太长，只有将一篇文章拆成两篇。本文我们将承接上文，继续讲解Nginx的实用特性，包括gzip功能、rewirte功能和一个第三方的节点监测模块。本文我们还将提到Taobao团队对Nginx的深度改造Tengine。

1、Nginx继续进阶

1.1、gzip

nginx对返回给浏览器的http response body是可以进行压缩的。虽然需要消耗一点cpu和内存资源，但是想到100KB的数据量可以压缩到60KB甚至更小进行传输，是否有一定的吸引力？这里我的建议是，不要为了节约成本将业务服务和负载层服务放在一台物理服务器上，这样做既影响性能又增加了运维难度。http返回数据进行压缩的功能在很多场景下都实用：

* 如果浏览器使用的是3G/4G网络，那么流量对于用户来说就是money。
* 压缩可节约服务器机房的对外带宽，为更多用户服务。按照目前的市场价良好的机房带宽资源的一般在200RMB/Mbps，而服务器方案的压力往往也来自于机房带宽。
* 主要注意的是，不是Nginx开启了gzip功能，HTTP响应的数据就一定会被压缩（me：http设置header会自动在传输的时候对数据进行压缩），除了满足Nginx设置的“需要压缩的http格式”以外，客户端（浏览器）也需要支持gzip（不然它怎么解压呢），一个好消息是，目前大多数浏览器和API都支持http压缩。

我们首先来讲解Nginx中的gzip的设置参数，然后我们讲解当开启压缩功能后，HTTP的交互过程和过程中关键的几个属性。我们首先来看看Nginx中开启gzip的属性（gzip的设置放置在HTTP主配置区域）：

#开启gzip压缩服务，   
gzip on;

#gzip压缩是要申请临时内存空间的，假设前提是压缩后大小是小于等于压缩前的。例如，如果原始文件大小为10K，那么它超过了8K，所以分配的内存是8 \* 2 = 16K;再例如，原始文件大小为18K，很明显16K也是不够的，那么按照 8 \* 2 \* 2 = 32K的大小申请内存。如果没有设置，默认值是申请跟原始数据相同大小的内存空间去存储gzip压缩结果。   
gzip\_buffers 2 8k;

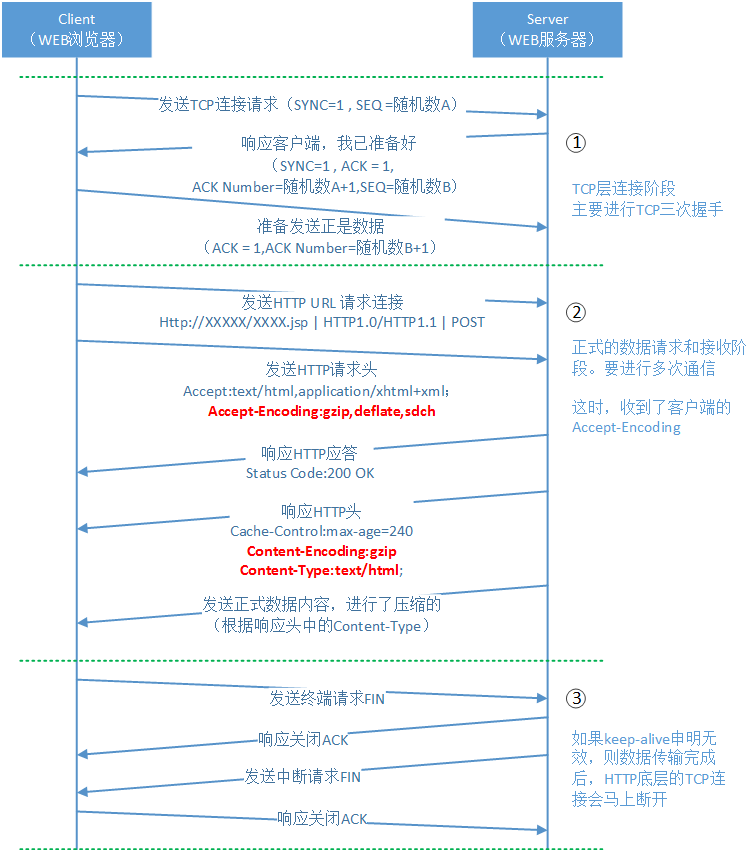
#进行压缩的原始文件的最小大小值，也就是说如果原始文件小于5K，那么就不会进行压缩了   
gzip\_min\_length 5K;

#gzip压缩基于的http协议版本，默认就是HTTP 1.1   
gzip\_http\_version 1.1;

# gzip压缩级别1-9，级别越高压缩率越大，压缩时间也就越长，消耗CPU越高   
gzip\_comp\_level 5;

#需要进行gzip压缩的Content-Type的Header的类型（me：设置的mime类型）。建议js、text、css、xml、json都要进行压缩；图片就没必要了，gif、jpge文件已经压缩得很好了，就算再压，效果也不好，而且还耗费cpu。   
gzip\_types text/HTML text/plain application/x-[**JavaScript**](http://lib.csdn.net/base/javascript) text/css application/xml;

设置完成后，重启nginx，即可生效。下面我们来看看浏览器和服务器进行gzip压缩的请求和处理返回过程（实际上在我的《标准Web系统的架构分层》文章中，已经有所提及）：



* 整个请求过程来看，开启gzip和不开启gip功能，其http的请求和返回过程是一致的，不同的是参数。这个可以看看我的另外一篇文章《标准Web系统的架构分层》[http://blog.csdn.net/yinwenjie/article/details/46480485](http://blog.csdn.net/yinwenjie/article/details/46480485" \t "_blank)
* 当开启HTTP的gzip功能时，客户端发出http请求时，会通过headers中的Accept-Encoding属性告诉服务器“我支持gzip解压，解压格式（**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**）deflate,sdch为：”。Accept-Encoding:gzip,deflate,sdch
* 注意，不是request说自己支持解压，Nginx返回response数据的时候就一定会压缩。这还要看本次Nginx返回数据的格式是什么，如果返回数据的原始数据格式，和设置的gzip\_types相符合，这时Nginx才会进行压缩。
* Nginx返回response headers时，如果数据被压缩了，就会在Content-Encoding属性中标示gzip，表示接下来返回的response content是经过压缩的；并且在Content-Type属性中表示数据的原始格式。
* 最后返回经过压缩的response content给客户端，客户端再进行解压。这里注意一下，在客户端发送的headers里面，有一个deflate,sdch。这是两种压缩算法，如果读者感兴趣，可以查查相关的资料（我建议查查，了解哈弗曼压缩算法对扩展自己的架构思路很有帮助）

1.2、rewrite

本小结内容，假定读者了解正则表达式。如果您不清楚正则表达式，请首先Google或者百度，正则表达式不在我们讨论的范围内。

Nginx的强大在于其对URL请求的重写（重定位）。Nginx的rewrite功能依赖于PCRE Lib，请一定在Nginx编译安装时，安装Pcre lib。请参见我的上一篇文章《架构设计：负载均衡层设计方案（2）——Nginx安装》[http://blog.csdn.net/yinwenjie/article/details/46620711](http://blog.csdn.net/yinwenjie/article/details/46620711" \t "_blank)

我们先从讲解rewrite的关键语法，然后出示几个示例，由示例进行讲解。先来说一下Nginx中几个关键的语法：

**正则表达式匹配：**

* ~ 区分大小写进行正则表达式匹配
* ~\* 不区分大小写进行正则表达式匹配
* !~ 区分大小写进行正则表达式不匹配
* !~\* 不区分大小写进行正则表达式不匹配

举例说明:

示例1：location ~\* \.(jpg|gif|png|ioc|jpeg)$

location是Nginx中的关键字，代表当前的URL请求值（me：请求的路径）。

以上表达式表示对URL进行不区分大小写的匹配，一旦URL以jpg或gif或png或ioc或jpeg结尾时，匹配成功。

示例2：$var1 ~ ^(\d+)$

var1是Nginx中使用set关键字定义的变量，以上语句代表var1和一个整数进行匹配。

**Nginx中的全局变量：**   
从上面的各个实例中，我们已经发现Nginx是支持变量的，Nginx还内置了一些全局变量，这里列举一些比较重要的全局变量：

* $content\_length: 获取request中header部分的“Content\_Length”值。
* $content\_type: 获取request中header部分的“Content\_type”值。
* $request\_method: 请求方式，常用的有两种请求方式：POST、GET
* $remote\_addr: 发送请求的客户端ip
* $remote\_port: 发送请求的客户端端口
* $request\_uri: 含有参数的完整的初始URI
* $server\_addr: request到达的server的ip。
* $server\_port: 请求到达的服务器的端口号。

**rewrite语法**

rewrite regex replacement flag

#regex：表示当前匹配的正则表达式。只有$url大小写相关匹配regex正则表达式，这个$url才会被rewrite进行重定向。

#replacement：重定向目标规则。这个目标规则支持动态变量绑定，这个问题下文马上用示例来讲。

#flag：重定向规则。

**rewrite中的Flag关键字**

* redirect：通知客户端重定向到rewrtie后面的地址（302）。
* permanent：通知客户端永久重定向到rewrtie后面的地址（303）。
* last：将rewrite后的地址重新在server标签执行。
* break：将rewrite后地址重新在当前的location标签执行。

实际上针对客户端来说，其效果是一样的，都是由客户端重新发起http请求，请求地址重新定位到replacement规则的URL地址；这里关键要讲解最常用的last和break两个关键字：

所有的rewrite语句都是要在server中的location中书写的，如下：

server {

。。。。。。

。。。。。。

location ... {

if(...) {

rewirte regex replacement flag;

}

rewirte regex replacement flag;

}

}

那么，break关键字说明重写的replacement地址在当前location的区域马上执行。

last关键字说明重写的replacement地址在当前server所有的location中重新再做匹配。

下面我们结合rewrite关键字和rewrite flag关键字给出典型的示例进行讲解：

示例1：

location ~\* ^/(.+)/(.+)\.(jpg|gif|png|jpeg)$ {

rewrite ^/orderinfo/(.+)\.(jpg|gif|png|jpeg)$ /img/$1.$2 break;

root /cephclient;

}

location在不进行大小写区分的情况下利用正则表达式对$url进行匹配。当匹配成功后进行rewrite重定位。

rewrite进行重写url的规则是：regex表达式第一个括号中的内容对应$1，regex表达式第二个括号中的内容对应$2，以此类推。

这样重定位的意义就很明确了：将任何目录下的文件名重定位到img目录下的对应文件名，

并且马上在这个location中（注意是Nginx，而不是客户端）执行这个重写后的URL定位。

示例2：

server {

。。。。

。。。。

location ~\* ^/orderinfo/(.+)\.(jpg|gif|png|jpeg)$ {

rewrite ^/orderinfo/(.+)\.(.+)$ /img/$1.$2 last;

}

location / {

root /cephclient;

}

}

在server中，有两个location位置，当url需要访问orderinfo目录下的某一个图片时，rewrite将重写这个url，

并且重新带入这个url到server执行，这样“location /”这个location就会执行了，并找到图片存储的目录。

1.3、健康检查模块

在本小节我们介绍一个用于Nginx对后端UpStream集群节点健康状态检查的第三方模块：nginx\_upstream\_check\_module（<https://github.com/yaoweibin/nginx_upstream_check_module>）。这个模块有资料介绍是TaoBao团队开发的，但是我在GitHua上试图求证时并没有找到直接证据。

这里需要说明的是，目前有很多Nginx模块实现Nginx对后端集群节点的健康监测，不止nginx\_upstream\_check\_module。Nginx官方有一个模块healthcheck\_nginx\_upstreams也可以实现对后端节点的健康监测（[https://github.com/cep21/healthcheck\_nginx\_upstreams](https://github.com/cep21/healthcheck_nginx_upstreams" \t "_blank)有详细的安装和使用介绍）

我们回到对nginx\_upstream\_check\_module的讲解，**要使用这个第三方模块首先您需要进行下载，然后通过patch命令将补丁打入您原有的Nginx源码中，并且重新进行编译安装**。下面我们来重点讲解一下这个模块的安装和使用。

**下载nginx\_upstream\_check\_module模块：**

wget https://codeload.github.com/yaoweibin/nginx\_upstream\_check\_module/zip/master

您也可以直接到GitHua上进行下载，还一个在linux系统上使用git命令进行下载。

**解压安装，并补丁打入Nginx源码**

# unzip ./nginx\_upstream\_check\_module-master.zip

注意是将补丁打入Nginx源码，不是Nginx的安装路径：

# cd ./nginx-1.6.2

# patch -p1 < ../nginx\_upstream\_check\_module-master/check\_1.5.12+.patch

如果补丁安装成功，您将看到以下的提示信息：

patching file src/http/modules/ngx\_http\_upstream\_ip\_hash\_module.c

patching file src/http/modules/ngx\_http\_upstream\_least\_conn\_module.c

patching file src/http/ngx\_http\_upstream\_round\_robin.c

patching file src/http/ngx\_http\_upstream\_round\_robin.h

这里请注意：在nginx\_upstream\_check\_module官网的安装说明中，有一个打补丁的注意事项：

If you use nginx-1.2.1 or nginx-1.3.0, the nginx upstream round robin

module changed greatly. You should use the patch named

'check\_1.2.1.patch'.

If you use nginx-1.2.2+ or nginx-1.3.1+, It added the upstream

least\_conn module. You should use the patch named 'check\_1.2.2+.patch'.

If you use nginx-1.2.6+ or nginx-1.3.9+, It adjusted the round robin

module. You should use the patch named 'check\_1.2.6+.patch'.

If you use nginx-1.5.12+, You should use the patch named

'check\_1.5.12+.patch'.

If you use nginx-1.7.2+, You should use the patch named

'check\_1.7.2+.patch'.

这里我们的Nginx的版本是1.6.2，那么就应该打入check\_1.5.12+.patch这个补丁

**重新编译安装Nginx：**

注意重新编译Nginx，要使用add-module参数将这个第三方模块安装进去：

# ./configure --prefix=/usr/nginx-1.6.2/ --add-module=../nginx\_upstream\_check\_module-master/

# make && make install

通过以上的步骤，第三方的nginx\_upstream\_check\_module模块就在Nginx中准备好了。接下来我们讲解一下如何使用这个模块。首先看一下upstream的配置信息：

upstream cluster {

# simple round-robin

server 192.168.0.1:80;

server 192.168.0.2:80;

check interval=5000 rise=1 fall=3 timeout=4000;

#check interval=3000 rise=2 fall=5 timeout=1000 type=ssl\_hello;

#check interval=3000 rise=2 fall=5 timeout=1000 type=http;

#check\_http\_send "HEAD / HTTP/1.0\r\n\r\n";

#check\_http\_expect\_alive http\_2xx http\_3xx;

}

上面的代码中，check部分就是调用nginx\_upstream\_check\_module模块的语法：

check interval=milliseconds [fall=count] [rise=count]

[timeout=milliseconds] [default\_down=true|false]

[type=tcp|http|ssl\_hello|mysql|ajp|fastcgi]

interval：必要参数，检查请求的间隔时间。

fall：当检查失败次数超过了fall，这个服务节点就变成down状态。

rise：当检查成功的次数超过了rise，这个服务节点又会变成up状态。

timeout：请求超时时间，超过等待时间后，这次检查就算失败。

default\_down：后端服务器的初始状态。默认情况下，检查功能在Nginx启动的时候将会把所有后端节点的状态置为down，检查成功后，在置为up。

type：这是检查通信的协议类型，默认为http。以上类型是检查功能所支持的所有协议类型。

check\_http\_send http\_packet

http\_packet的默认格式为："GET / HTTP/1.0\r\n\r\n"

check\_http\_send设置，这个设置描述了检查模块在每次检查时，向后端节点发送什么样的信息

check\_http\_expect\_alive [ http\_2xx | http\_3xx | http\_4xx | http\_5xx ]

这些状态代码表示服务器的HTTP响应上是OK的，后端节点是可用的。默认情况的设置是：http\_2xx | http\_3xx

当您根据您的配置要求完成检查模块的配置后，请首先使用nginx -t 命令监测配置文件是否可用，然后在用nginx -s reload重启nginx。

1.4、不得不提的tengine

Tengine是由淘宝网发起的Web服务器项目。它在Nginx的基础上，针对大访问量网站的需求，添加了很多高级功能和特性。Tengine的性能和稳定性已经在大型的网站如淘宝网，天猫商城等得到了很好的检验。它的最终目标是打造一个高效、稳定、安全、易用的Web平台（[http://tengine.taobao.org/](http://tengine.taobao.org/" \t "_blank)）。

您应该懂了，我建议您根据业务的实际情况，适时在生产环境引入Tengine。但在本博客发布时，Tengine的2.X版本还不稳定，所以建议实用1.5.2的稳定版本。**请记住Tengine就是经过升级改造后的Nginx**。

2、后文介绍

花了两篇文章的功夫，终于将我想给大家讲解的nginx的实用特性讲完了，但是nginx远远不止这些特性。后面有时间我们会再回到Nginx，重点讲解针对Nginx的脚本开发，我们还会讲解Nginx和Lua的集成。但是为了不扰乱这个系列博文的时间安排，下篇文章我们将开始介绍LVS技术，争取用一篇文章的篇幅讲清楚LVS核心设计思想、单节点安装和使用。再下篇文章我们介绍Keepalived技术，以及keepalived和LVS、Nginx分别进行集成，敬请关注。