Nginx 负载均衡

原文地址：http://nginx.com/resources/admin-guide/load-balancer/ (http://nginx.com/resources/admin-guide/load-balancer/)

Nginx Load Balancing
nginx 负载均衡

This section describes how to use NGINX and NGINX Plus as a load balancer (http://nginx.com/solutions/load-balancing/).
本章将讨论如何使用Nginx和Nginx加做负载均衡 (http://nginx.com/solutions/load-balancing/)器。

In This Section
本章包括

- Load balancing overview (http://nginx.com/resources/admin-guide/load-balancer/#overview)
- 负载均衡概览
- Proxying traffic to a group of servers (http://nginx.com/resources/admin-guide/load-balancer/#proxy_pass)
- 代理流量到服务器组
- Choosing a load balancing method (http://nginx.com/resources/admin-guide/load-balancer/#method)
- 选择一个负载均衡方法
- Server weights (http://nginx.com/resources/admin-guide/load-balancer/#weight)
- 服务器权值
- Server slow start (http://nginx.com/resources/admin-guide/load-balancer/#slow_start)
- 服务器慢启动
- Enabling session persistence (http://nginx.com/resources/admin-guide/load-balancer/#sticky)
- 开启会话持久
- Limiting the number of connections (http://nginx.com/resources/admin-guide/load-balancer/#maxconns)
- 限制连接数目
- Passive health monitoring (http://nginx.com/resources/admin-guide/load-balancer/#health_passive)
- 被动的健康检测
- Active health monitoring (http://nginx.com/resources/admin-guide/load-balancer/#health_active)
- 主动的健康检测
- Sharing data with multiple worker processes (http://nginx.com/resources/admin-guide/load-balancer/#zone)
- 多工作进程共享数据
- Configuring load balancing using DNS (http://nginx.com/resources/admin-guide/load-balancer/#resolve)
- 使用DNS配置负载均衡
- Runtime reconfiguration (http://nginx.com/resources/admin-guide/load-balancer/#upstream_conf)
- 实时重配置

Overview
概览

Load balancing across multiple application instances is a commonly used technique for optimizing resource utilization, maximizing throughput, reducing latency, and ensuring fault-tolerant configurations.
通常多个应用实例来做负载均衡是优化资源利用、最大化吞吐量、减少延迟、确保容错配置。
NGINX can be used as a very efficient HTTP load balancer in different deployment scenarios (http://nginx.com/blog/nginx-load-balance-deployment-models/).
Nginx可以通过不同的部署方案 (http://nginx.com/blog/nginx-load-balance-deployment-models/)作为非常有效的HTTP负载均衡器。
Proxying Traffic to a Group of Servers
代理流量到服务器组
To start using NGINX with a group of servers, first, you need to define the group with the upstream (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream) directive. The directive is placed in the **http** context.

要开始在一组服务器里使用Nginx，首先，你得用 upstream (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream) 指令来定义一个组。这个指令放在http块里。
Servers in the group are configured using the server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive (not to be confused with the "server" block that defines a virtual server running on NGINX). For example, the following configuration defines a group named "backend" and consists of three server configurations (which may resolve in more than three actual servers):
分组里的服务器通过指令server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server)配置（不要跟"server"块搞混淆了，那是用于定义虚拟主机的）。例如，下面这个配置定义了一个由三个服务器配置（也可能更多）组成的名为"backend"的分组。

[plain]

```
http {
    upstream backend {
        server backend1.example.com weight=5;
        server backend2.example.com;
        server 192.0.0.1 backup;
    }
}
```

To pass requests to a server group, the name of the group is specified in the proxy_pass (http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass)directive (or fastcgi_pass (http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html#fastcgi_pass), memcached_pass (http://nginx.org/en/docs/http/ngx_http_memcached_module.html#memcached_pass), uwsgi_pass (http://nginx.org/en/docs/http/ngx_http_uwsgi_module.html#uwsgi_pass), scgi_pass (http://nginx.org/en/docs/http/ngx_http_scgi_module.html#scgi_pass) depending on the protocol). In the next example, a virtual server running on NGINX passes all requests to the "backend" server group defined in the previous example:
要发送请求给一个服务器组，使用 proxy_pass (http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_pass) 指令来指定分组名（或者是根据协议使用fastcgi_pass (http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html#fastcgi_pass), memcached_pass (http://nginx.org/en/docs/http/ngx_http_memcached_module.html#memcached_pass), uwsgi_pass (http://nginx.org/en/docs/http/ngx_http_uwsgi_module.html#uwsgi_pass), scgi_pass (http://nginx.org/en/docs/http/ngx_http_scgi_module.html#scgi_pass) ）。在下个例子中，一个跑在Nginx上的虚拟服务器发送所有请求到前一个例子中定义的"backend"服务器组。

[plain]

```
server {
    location / {
        proxy_pass http://backend;
    }
}
```

The following example sums up the two examples above and shows proxying requests to the "backend" server group, where the server group consists of three servers, two of them run two instanses of the same application while one is a backup server, NGINX applies HTTP load balancing to distribute the requests:
下面这个例子综合了上述两个示例，将请求代理到"backend"服务器组，"backend"服务器组由三个服务器组成，其中两个运行相同的应用实例，一个为备用服务器，Nginx应用HTTP负载均衡来分发请求：

[plain]

```
http {
    upstream backend {
        server backend1.example.com;
        server backend2.example.com;
        server 192.0.0.1 backup;
    }
    server {
        location / {
            proxy_pass http://backend;
        }
    }
}
```

Choosing a Load Balancing Method
选择一个负载均衡方式

NGINX supports four load balancing methods:
Nginx支持四个负载均衡方法：
1、The round-robin method: requests are distributed evenly across the servers with server weights (http://nginx.com/resources/admin-guide/load-balancer/#weight) taken into consideration. This method is used by default:

1、轮循法：请求根据服务器权重 (http://nginx.com/resources/admin-guide/load-balancer/#weight)均衡地分布到各服务器，这个方法为默认方法：

[plain]

```
upstream backend {
  server backend1.example.com;
  server backend2.example.com;
}
```

2、The least_conn (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#least_conn) method: a request is sent to the server with the least number of active connections with server
 weights (http://nginx.com/resources/admin-guide/load-balancer/#weight) taken into consideration:

2、least_conn (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#least_conn)法：请求被发送到激活连接数最少的服务器，服务器权重也为选择因素：

[plain]

```
upstream backend {
    least_conn;

    server backend1.example.com;
    server backend2.example.com;
}
```

3、The ip_hash (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#ip_hash) method: the server to which a request is sent is determined from the client IP address. In this case, either the first three octets of IPv4 address or the whole IPv6 address are used to calculate the hash value.

3、ip 哈希法：请求发送到哪个服务器取决于客户端的IP地址。这种情况下，使用IPv4 地址的前3个字节或者IPv6的整个地址用来计算哈希值。

The method guarantees that requests from the same address get to the same server unless it is not available.
这个方法能保证来自同一个地址的请求送到同一个服务器除非这个服务器不可用了。

[plain]

```
upstream backend {
    ip_hash;

    server backend1.example.com;
    server backend2.example.com;
}
```

4、The generic hash (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#hash) method: the server to which a request is sent is determined from a user-defined key which may be a text, variable, or their combination. For example, the key may be a source IP and port, or URI:

4、通用哈希法：请求发送到哪个服务器取决于一个用户端定义的关键词，如文本，变量或两者组合。例如，这个关键词可以是来源IP和端口，或者URI：

[plain]

```
upstream backend {
    hash $request_uri consistent;

    server backend1.example.com;
    server backend2.example.com;
}
```

The optional consistent parameter of the hash (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#hash) directive enables ketama (http://www.last.fm/user/RJ/journal/2007/04/10/392555/) consistent hash load balancing. Requests will be evenly distributed across all upstream servers on the user-defined hashed key value. If an upstream server is added to or removed from an upstream group, only few keys will be remapped which will minimize cache misses in case of load balancing cache servers and other applications that accumulate state.

hash (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#hash) 指令的可选的consistent参数开启ketama (http://www.last.fm/user/RJ/journal/2007/04/10/392555/)一致性哈希负载均衡。请求将会通过用户端定义的关键词的哈希值均衡地分发到上游分组，只有少数几个关键词会被重映射以最小化负载均衡服务器和其他应用积累状态导致的缓存缺失。
If a method other than the default one is used, the corresponding directive (least_conn,ip_hash or hash) should be specified inside the upstream before the server directives.
如果要使用非默认的方法，相应的指令（least_conn,ip_has 或者 hash）必须放在upstream块里的server指令之前。
If one of the servers needs to be temporarily removed, it can be marked with the down (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#down) parameter in order to preserve the current hashing of client IP addresses. Requests that were to be processed by this server are automatically sent to the next server in the group:
如果某个服务器需要临时移除，可以使用down (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#down)参数来标记它，当前正在哈希的客户端IP地址得以保存。本该这个服务器处理的请求会自动地发送给本组中的下一个服务器：

[plain]

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
    server backend3.example.com down;
}
```

Server Weights
服务器权重

By default, NGINX distributes requests among the servers in the group according to their weights using the round robin algorithm. The weight (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#weight) parameter of the server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive sets the weight of a server, by default, it is 1:
默认，Nginx根据服务器组里服务器的权重使用轮循算法为分发请求。server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) 指令的weight (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#weight)参数用来设置服务器的权重，此值缺省为1：

[plain]

```
upstream backend {
    server backend1.example.com weight=5;
    server backend2.example.com;
    server 192.0.0.1 backup;
}
```

In the example, the first server has weight 5, the other two servers have the default weight (=1), but one of them is marked as a backup server and does not normally receive any requests. So of every six requests, five requests will be sent to the first server and one request will be sent to the second server.
本例中，第一个服务器权重为5，另外两个采用默认值1，但是其中一个被标记为备用服务器所以通常情况下不会接收请求。所以每6个请求中，5个发送到第一个服务器1个发送到第二个服务器。

Server Slow Start
服务器慢启动

The server slow start feature prevents a recently recovered server from being overwhelmed by connections, which may timeout and cause the server to be marked as failed again.
服务器慢启动功能能防止刚刚恢复的服务器被大量连接超载导致服务器超时又被标为失败。
In NGINX, slow start allows an upstream server to gradually recover its weight from zero to its nominal value after it bas been recovered or became available. This can be done with the slow_start (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#slow_start) parameter of the server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive:
在Nginx中，慢启动能让上流服务器在完全恢复并可用之后逐渐把它的权重从0恢复到设置的值。这个或能 可以通过server指令的slow_start (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#slow_start)参数实现：

```
[plain]
upstream backend {
    server backend1.example.com slow_start=30s;
    server backend2.example.com;
    server 192.0.0.1 backup;
}
```

The time value sets the time for the server will recover its weight.
这个时间值设置了服务器恢复权重的时间。
Note that if there is only a single server in a group, max_fails (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#max_fails), fail_timeout (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#fail_timeout) and slow_start (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#slow_start) parameters will be ignored and this server will never be considered unavailable.
注意如果分组里只有一个服务器， max_fails (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#max_fails), fail_timeout (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#fail_timeout) 和 slow_start (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#slow_start)参数会被忽略，并且这个服务器永远不会被判定为不可用。

Enabling Session Persistence
开启会话持久性

Session persistence means that NGINX identifies user sessions and routes the requests from this session to the same upstream server.
会话持久意思是Nginx认证用户会话并且把来自该会话的请求路由到相同的上游服务器。
NGINX supports three session persistence methods. The methods are set with the sticky (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky) directive.
Nginx支持三个会话持久方法，这三个方法使用sticky (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky)指令设置。
1、The sticky cookie (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky_cookie) method. With this method, NGINX adds a session cookie to the first response from the upstream group and identifies the server which has sent the response. When a client issues next request, it will contain the cookie value and NGINX will route the request to the same upstream server:

1、sticky cookie (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky_cookie) 方法。

使用这个方法，Nginx添加一个会话cookie给第一个来自这个上游分组的响应，并且识别谁发送一个响应的服务器。当一个客户端发出下一个请求，请求会带着这个cookie值，Nginx将把这个请求路由到相同的上游服务器：

```
[plain]
upstream backend {
    server backend1.example.com;
    server backend2.example.com;

    sticky cookie srv_id expires=1h domain=.example.com path=/;
}
```

In the example, the srv_id parameter sets the name of the cookie which will be set or inspected. The optional expires parameter sets the time for the browser to keep the cookie. The optional domain parameter defines a domain for which the cookie is set. The optional path parameter defines the path for which the cookie is set. This is the simplest session persistence method.
本例中，srv_id 参数设置用来设置或者检测的cookie名称。可选参数expires指定浏览器保存这个cookie的时长。可选参数domain定义cookie设置到的域名。可选参数path定义cookie的保存路径。这是最简单的会话持久方法。
2、The sticky route (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky_route) method. With this method, NGINX will assign a “route” to the client when it receives the first request. All subsequent requests will be compared with the route (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#route) parameter of the server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive to identify the server where requests will be proxied. The route information is taken from either cookie, or URI.

2、sticky route (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky_route)方法。

通过这个方法，Nginx在第一次接收到请求的时候给这个客户端分配一个“路由”。所有接下来的请求都会和server指令的route (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#route)参数对比找出请求将被代理到的服务器。路由信息可以从cookie或URI中获取。

[plain]

```
upstream backend {
    server backend1.example.com route=a;
    server backend2.example.com route=b;

    sticky route $route_cookie $route_uri;
}
```

3、The cookie learn (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky_learn) method. With this method, NGINX first finds session identifiers by inspecting requests and responses. Then NGINX “learns” which upstream server corresponds to which session identifier. Generally, these identifiers are passed in a HTTP cookie. If a request contains a session identifier already “learned”, NGINX will forward the request to the corresponding server:

3、cookie learn (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#sticky_learn) 方法。

Nginx首先通过检查请求和响应来查找会话ID。接着Nginx就知道哪个上游服务器对应哪个会话ID。通常，这些ID是通过HTTP cookie传递的。如果一个请求包含一个已经识别过的会话ID，Nginx直接发送这个请求给相应的服务器：

[plain]

```
upstream backend {
  server backend1.example.com;
  server backend2.example.com;

  sticky learn
      create=$upstream_cookie_sessionid
      lookup=$cookie_sessionid
      zone=client_sessions:1m
      timeout=1h;
}
```

In the example, one of the upstream servers creates a session by setting the cookie “SESSIONID” in the response.
本例中，其中一个上游服务器通过在响应中设置cookie “SESSIONID” 来创建一个会话。
The obligatory parameter create specifies a variable that indicates how a new session is created. In our example, new sessions are created from the cookie “SESSIONID” sent by the upstream server.
必填参数create指定一个变量表示一个新的会话是如何创建的。在我们的示例中，新的会话是通过上游服务器发送的cookie “SESSIONID”创建的。
The obligatory parameter lookup specifies how to search for existing sessions. In our example, existing sessions are searched in the cookie “SESSIONID” sent by the client.

必填参数lookup指定如何搜索存在的会话。示例中，存在的会话在客户端发送的cookie “SESSIONID”中搜索。
The obligatory parameter zone specifies a shared memory zone where all information about sticky sessions is kept. In our example, the zone is named client_sessions and has the size of 1 megabyte.
必填参数zone指定所有会话信息保存的共享内存空间。我们这个例子中，空间名为client_sessions，大小为1M。
This is a more sophisticated session persistence method as it does not require keeping any cookies on the client side: all info is kept server-side in the shared memory zone.
这是一个更复杂的会话持久方法，因为他不需要在客户端保存任何cookie：所有信息保存在服务端的共享内存空间里。

Limiting the Number of Connections
限制连接数目

With NGINX Plus, it is possible to maintain the desired number of connections by setting the connection limit with the max_conns (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#max_conns) parameter.
使用Nginx加，可以通过max_conns (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#max_conns)参数设置连接限制维持所需的连接数量。
If the max_conns limit has been reached, the request can be placed into the queue for its further processing provided that the queue (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#queue) directive is specified. The directive sets the maximum number of requests that can be simultaneously in the queue:
如果max_conns (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#max_conns)设置的限制达到了，请求可以被放入队列进行queue (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#queue)指令提供的进一步的处理。queue (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#queue)指令设置了可以同时放入队列的最大请求数：

[plain]

```
upstream backend {
    server backend1.example.com  max_conns=3;
    server backend2.example.com;

    queue 100 timeout=70;
}
```

If the queue is filled up with requests or the upstream server cannot be selected during the timeout specified in the optional timeout parameter, the client will receive an error.
如果队列排满或者在可选参数timeout设置的时间内无法选择上游服务器，客户端将接到一个错误。
Note that the max_conns limit will be ignored if there are idle keepalive (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#keepalive) connections opened in other worker processes (http://nginx.org/en/docs/ngx_core_module.html#worker_processes). As a result, the total number of connections to the server may exceed the max_conns value in a configuration where the memory is shared with multiple worker processes (http://nginx.com/resources/admin-guide/load-balancer/#zone).
注意如果闲置的keepalive (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#keepalive)连接在另一个worker processes (http://nginx.org/en/docs/ngx_core_module.html#worker_processes)中打开了max_conns限制会被忽略。导致的结果是，在多个工作进程共享内存 (http://nginx.com/resources/admin-guide/load-balancer/#zone)的配置中，连接的总数可能会超过max_conns的值。

Passive Health Monitoring
被动式健康检测

When NGINX considers a server unavailable, it temporarily stops sending requests to that server until it is considered active again. The following parameters of the server directive configure the conditions to consider a server unavailable:
当Nginx认为一台服务器不可用，就暂时性的停止发送请求到这台服务器直到它再次被判定为可用。下面这些server指令的参数用来配置判断一台服务器不可用的条件：

- The fail_timeout parameter sets the time during which the specified number of failed attempts should happen and still consider the server unavailable. In other words, the server is unavailable for the interval set by fail_timeout.
- fail_timeout 参数设置了在一定时间内，如果尝试失败的次数达到规定的数目，一直认为该服务器不可用。换句话说，服务器在fail_timeout设置的时间间隔内不可用。
- The max_fails parameter sets the number of failed attempts that should happen during the specified time to still consider the server unavailable.
- max_fails参数设置在规定时间内发生的失败尝试的次数来判定服务器不可用。

The default values are 10 seconds and one attempt. So if NGINX fails to send a request to some server or does not receive a response from this server at least once, it immediately considers the server unavailable for 10 seconds. The following example shows how to set these parameters:
默认值为10秒钟一次尝试。就说如果Nginx发送请求到一个服务器或者从一个服务器失败至少一次，这个服务器直接被认为不可用10秒钟。下例展示如何设置这些参数。

[plain]
```
upstream backend {
    server backend1.example.com;
    server backend2.example.com max_fails=3 fail_timeout=30s;
    server backend3.example.com max_fails=2;
}
```

Next are some more sophisticated features for tracking server availability, which are available in the commercial subscription (http://nginx.com/products/) of NGINX.
下面是一些更复杂的跟踪服务器可用性功能，这些功能在Nginx的商业订购 (http://nginx.com/products/)中。

Active Health Monitoring
主动的健康检测

Periodically sending special requests to each server and checking for a response that satisfies certain conditions can monitor the availability of servers.
定期地发送一些特别的请求到每个服务器，检测一个满足一些条件的响应来监视服务器的可用性。
To enable this type of health monitoring in your nginx.conf file the location that passes requests to the group should include the health_check (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#health_check) directive. In addition, the server group should also be dynamically configurable with the zone (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#zone) directive:
要开启这类健康检测，在你的nginx.conf文件中发送请求到这个分组的location块里面必须包含health_check (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#health_check)指令。另外，服务器组必须通过zone (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#zone)指令进行动态配置：

[plain]
```
http {
    upstream backend {
        zone backend 64k;

        server backend1.example.com;
        server backend2.example.com;
        server backend3.example.com;
        server backend4.example.com;
    }

    server {
        location / {
            proxy_pass http://backend;
            health_check;
        }
    }
}
```

This configuration defines a server group and a virtual server with a single location that passes all requests to a server group. It also enables health monitoring with default parameters. In this case, every five seconds NGINX will send the “/” requests to each server in the backend group. If any communication error or timeout occurs (or a proxied server responds with the status code other than 2xx or 3xx) the health check will fail for this proxied server. Any server that fails a health check will be considered unhealthy, and NGINX will stop sending client requests to it until it once again passes a health check.
这个配置定义了一个服务器组，一个只有一个location块的虚拟机，这个location里的所有请求都将发送到这个服务器组。这个location还开启了参数为默认值的健康监视。这样子，每过五秒，Nginx会发送”/‘请求到每个backend组里的服务器，如果哪个对话错误或者超时（或者哪个代理服务器响应里返回的状态码不是2或3开头的）那么这个代理服务器健康检测失败。任何一个健康检测为失败的服务器被认为是不健康的，Nginx将停止发送客户端请求到这台服务器直到它再次被检测为健康。
The zone directive defines a memory zone that is shared among worker processes and is used to store the configuration of the server group. This enables (http://nginx.org/en/docs/http/load_balancing.html#shared) the worker processes to use the same set of counters to keep track of responses from the servers in the group. The zone directive also makes the group dynamically configurable (http://nginx.org/en/docs/http/load_balancing.html#runtime_upstream).

zone指令定义了一个在工作进程中共享的内存空间，该内存空间用于存储服务器组的配置。这可以让所有工作进程使用同一个计数器来跟踪这个分组里的服务器的响应。zone同样能让分组支持动态配置 (http://nginx.org/en/docs/http/load_balancing.html#runtime_upstream)。
This behavior can be overridden using the parameters of the health_check directive:
这个行为可以通过health_check指令的参数覆盖：

[plain]

```
location / {
    proxy_pass http://backend;
    health_check interval=10 fails=3 passes=2;
}
```

Here, the duration between two consecutive health checks has been increased to 10 seconds using the interval parameter. In addition, a server will be considered unhealthy after 3 consecutive failed health checks by setting the fails parameter to 3. Finally, using the passes parameter, we have made it so that a server needs to pass 2 consecutive checks to be considered healthy again.
此间，两次连续的健康检测的时间间隔被通过interval参数增加到10秒。另外 ，通过设置fails参数为3，一个服务器在连续3次健康检测失败后被认为不健康。最后，使用passes参数，我们让一个服务器连续两次检测为健康才被重新认为是健康的。
It is possible to set a specific URI to request in a health check. Use the uri parameter for this purpose:
要在健康检测请求中指定URI，使用uri参数 ：

[plain]

```
location / {
    proxy_pass http://backend;
    health_check uri=/some/path;
}
```

The provided URI will be appended to the server domain name or IP address specified for the server in the upstream directive. For example, for the first server in the backend group declared above, a health check request will have the http://backend1.example.com/some/path URI.
提供的URI将添加到在upstream指令中server指令指定的服务器域名或者IP地址后面。例如，上述backend分组中第一个server，健康检测的请求URI为"http://backend1.example.com/some/path”。
Finally, it is possible to set custom conditions that a healthy response should satisfy. The conditions are specified in the match (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#match) block, which is defined in the match parameter of the health_check directive.
最后，也可以自定义满足健康检测的条件。这些条件在match (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#match)块里规定，match (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#match)块在health_check指令的match参数里定义。

[plain]

```
http {
    ...

    match server_ok {
        status 200-399;
        body !~ "maintenance mode";
    }

    server {
        ...

        location / {
            proxy_pass http://backend;
            health_check match=server_ok;
        }
    }
}
```

Here a health check is passed if the response has the status in the range from 200 to 399, and its body does not match the provided regular expression.
这里如果一个响应状态码在200到399之间，并且响应体里没有匹配上提供的正则表达式，那健康检测就算通过了。
The match directive allows NGINX to check the status, header fields, and the body of a response. Using this directive it is possible to verify whether the status is in the specified range, whether a response includes a header, or whether the header or body matches a regular expression. The match directive can contain one status condition, one body condition, and multiple header conditions. To correspond to the match block, the response must satisfy all of the conditions specified within it.

match指令可以让Nginx检测一个响应的状态码，头字段，以及主体内容。通过这个指令可以实现，确认状态码是否在规定的范围内，一个响应是否包含一个头，或者响应头和响应体是否匹配一个正则式。match指令可以包含一个状态码条件，一个响应体条件和多个响应头条件。要想符合match块，响应必须满足里面定义的所有条件。
For example, the following match directive looks for responses that have the status code 200, contain the “Content-Type” header with text/html exact value, and have the body that includes the text “Welcome to nginx!”:
例如，下面这个match指令查找这样的响应，状态码为200，包含准确值为text/html的”Content-type”头，还有响应体要包含文本“Welcome to nginx!”：

[plain]

```
match welcome {
    status 200;
    header Content-Type = text/html;
    body ~ "Welcome to nginx!";
}
```

In the following example using a !, conditions match responses that have the status **other** than 301, 302, 303, and 307 and **do not** include the “Refresh” header field.
下面这个例子使用了一个!，条件为匹配拥有301,302,303,307以外状态码和不包含”Refresh”头字段的响应。

[plain]

```
match not_redirect {
    status ! 301-303 307;
    header ! Refresh;
}
```

Health checks can also be enabled for non-HTTP protocols, such as FastCGI, SCGI, uWSGI and memcached.
健康检测也可以在非HTTP协议里启用，例如FastCGI,SCGI,uWSGI和memecached。

Sharing Data with Multiple Worker Processes
多进程共享数据

If the upstream directive **does not** include the zone (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#zone) directive, each worker process keeps its own copy of the server group configuration and maintains its own set of related counters. The counters include the current number of connections to each server in the group and the number of failed attempts to pass a request to a server. As a result, the server group configuration isn’t changeable.
如果upstream指令不包含zone指令，每个工作进程保存自己分组配置的拷贝，并且维护自己相关的一系统计算器。这些计数器包含了组里每个服务器当前的连接数和尝试发送一个请求到一个服务器的失败次数。其结果是，服务组的配置是不能改变的。
If the upstream directive **does** include the zone (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#zone) directive, the configuration of the server group is placed in a memory area shared among all worker processes. This scenario is dynamically configurable, because the worker processes access the same copy of the group configuration and utilize the same related counters.
如果upstream指令包含了zone (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#zone)指令，服务器分组的配置放在一个所有工作进程共享的内存区域里，这个方案是可动态配置的，因为所有工作进程访问同一个分组配置的拷贝并且使用同一个相关计数器。
The zone directive is mandatory for health checks (http://nginx.org/en/docs/http/load_balancing.html#health_monitor) and runtime modification (http://nginx.org/en/docs/http/load_balancing.html#runtime_upstream) of the server group. However, other features of the server groups can benefit from the use of this directive as well.
对于服务器组的健康检测 (http://nginx.org/en/docs/http/load_balancing.html#health_monitor)和运行时修改 (http://nginx.org/en/docs/http/load_balancing.html#runtime_upstream)，zone指令是强制的。而且，服务器分组的其他功能也可以从该指令的使用中受益。
For example, if the configuration of a group is not shared, each worker process maintains its own counter for failed attempts to pass a request to a server (see the max_fails parameter). In this case, each request gets to only one worker process. When the worker process that is selected to process a request fails to transmit the request to a server, other worker processes don’t know anything about it. While some worker process can consider a server unavailable, others may still send requests to this server. For a server to be definitively considered unavailable, max_fails multiplied by the number of workers processes of failed attempts should happen within the timeframe set by fail_timeout. On the other hand, the zone directive guarantees the expected behavior.
例如，如果一个分组的配置不是共享的，每个工作进程维护自己的一个请求到一个服务器的失败尝试次数（看max_fails参数）计数器。这样的话，每个请求只到一个工作进程。当原本该处理这个请求的工作进程传送请求到服务器失败，其他工作进程根本不知道。当一个工作进程认为一个服务器不可用的时候，其他工作进程仍然向该服务器发送请求。一个服务器被判定为不可用的条件就变成，在fail_timeout设置的时间内失败尝试的次数（为

max_fails设置的值）乘上工作进程的数目。换个方面说，zone 指令能保证预期的行为。
The least_conn load balancing method might not work as expected without the zone directive, at least on small loads. This method of load balancing passes a request to the server with the least number of active connections. Again, if the configuration of the group is not shared, each worker process uses its own counter for the number of connections. And if one worker process passes by a request to a server, the other worker process can also pass a request to the same server. However, you can increase the number of requests to reduce this effect. On high loads requests are distributed among worker processes evenly, and the least_conn load balancing method works as expected.
如果没有zone指令，least_conn 负载均衡方法也可能达不到预期的效果，至少在小负载上。这个负载均衡方法把请求发送给活跃连接数最小的服务器。又来了，如果一个分组的配置不是共享的，每个工作进程使用自己连接数的计数器。而且如果一个工作进程传送一个请求到一个服务器，其他工作进程也会发送请求到相同的服务器。然而，你可以增加请求数量来降低这个影响。在高负载下，请求会平均地分布到各个工作进程，least_conn 负载均衡方法就如预期的工作了。

Setting the Size for the Zone
设置zone的大小

There are no exact settings due to quite different usage patterns. Each feature, such as sticky cookie/route/learn load balancing, health checks, or re-resolving will affect the zone size.
没有什么是准确的配置因为不同的使用方案差异十分之大。每一个功能，例如sticky cookie/route/learn 负载均衡方法，健康检测，或者重解析都会影响zone的大小。
For example, the 256 Kb zone with the “sticky_route” session persistence method and a single health check can hold up to:
例如，256Kb 大的空间使用sticky_route会话持久方式和单健康检测能够支持：

- 128 servers (adding a single peer by specifying IP:port);
- 128个服务器（使用IP:port方式添加单个）；
- 88 servers (adding a single peer by specifying hostname:port, hostname resolves to single IP);
- 88个服务器（使用主机名:端口指定单个方式，其中，一个主机名解析为一个IP）；
- 12 servers (adding multiple peers by specifying hostname:port, hostname resolves to many IPs).
- 12个服务器（指定主机名:端口添加多个，主机名对应多个IP）。

Configuring Load Balancing Using DNS
使用DNS配置负载均衡

The configuration of a server group can be modified at run time using DNS.
这种配置可以在运行时使用DNS修改服务器组。
NGINX Plus can monitor changes of IP addresses that correspond to a domain name of the server and automatically apply these changes to NGINX without its restart. This can be done with the resolver (http://nginx.org/en/docs/http/ngx_http_core_module.html#resolver) directive which must be specified in the http block, and the resolve (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#resolve) parameter of the server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive in a server group:
Nginx加能够监测服务器的域名对应的IP地址的变化，并且在不重启Nginx的情况下应用这些变化。这功能可以通过resolver (http://nginx.org/en/docs/http/ngx_http_core_module.html#resolver)指令实现，该指令必须放在http块里，还有服务器组里server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server)指令的resolve (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#resolve)参数必须指定。

[plain]

```
http {
    resolver 10.0.0.1 valid=300s ipv6=off;
    resolver_timeout 10s;

    server {
        location / {
            proxy_pass http://backend;
        }
    }

    upstream backend {
        zone backend 32k;
        least_conn;
        ...
        server backend1.example.com resolve;
        server backend2.example.com resolve;
    }
}
```

In the example, the resolve (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#resolve) parameter of the server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) directive will preiodically re-resolve “backend1.example.com” and “backend2.example.com” servers into IP addresses. By default, NGINX re-resolves DNS records basing on their TTL, but the TTL value can be overridden with the valid parameter of the resolver (http://nginx.org/en/docs/http/ngx_http_core_module.html#resolver) directive, in our example it is 5 minutes.
本例中，server (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#server) 指令的resolve (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#resolve)参数会定期的重解析 “backend1.example.com” 和 “backend2.example.com”两个服务器的IP地址。默认，Nginx基于DNS和TTL重解析DNS记录，而TTL的被能够resolver (http://nginx.org/en/docs/http/ngx_http_core_module.html#resolver)指令中valid参数覆盖。本例为5分钟。
The optional ipv6=off parameter allows resolving only to IPv4 addresses, though both IPv4 and IPv6 resolving is supported.
可选参数 ipv6=off 设置只解析IPv4地址，尽管IPv4和IPv6的解析都是支持的。
If a domain name resolves to several IP addresses, the addresses will be saved to the upstream configuration and load-balanced. In our example, the servers will be load balanced according to the least_conn (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#least_conn) load balancing method. If one or more IP addresses has been changed or added/removed, then the servers will be re-balanced.
如果一个域名解析为多个IP地址，这些地址会保存到上游和负载均衡的配置。在我们的例子中，这些服务器会通过least_conn (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#least_conn)负载求均衡方法进程负载均衡。如果一个或多个IP地址被修改了添加了或者删除了，这些服务器会被再均衡。

Runtime Reconfiguration
实时重配置

The configuration of a server group can be modified at run time using the special HTTP interface. A configuration command can be used to view all servers or a specific server in a group, modify parameter for a specific server, and add or remove servers.
一个服务器组的配置可以通过使用特殊的HTTP接口实时修改。一个配置命令可以用来查看一组服务器或组内其中一个服务器，修改某个服务器参数，添加或者删除服务器。
To reconfigure at runtime:
实时重配置：

- Specify the zone (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#zone) directive in the upstream directive that configures a group. The zone directive configures a zone in the shared memory. The configuration of the server group will be kept in this zone, so all worker processes will use the same configuration.
- 在配置分组的upstream指令中指定zone指令。zone指令在共享内存中配置一个空间。服务器组的配置将被保存在这个空间里，因此所有的工作进程使用相同的配置。
- Place the upstream_conf (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream_conf) directive in a separate location. The access to this location should be restricted (http://nginx.org/en/docs/http/ngx_http_core_module.html#satisfy).
- 在单个的location块里使用upstream_conf (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream_conf)指令，对这个location的访问必须是受限的 (http://nginx.org/en/docs/http/ngx_http_core_module.html#satisfy)。

An example of this configuration:
示例配置：

```
[plain]
http {
    ...
    # Configuration of the server group
    upstream appservers {
        zone appservers 64k;

        server appserv1.example.com      weight=5;
        server appserv2.example.com:8080 fail_timeout=5s;

        server reserve1.example.com:8080 backup;
        server reserve2.example.com:8080 backup;
    }

    server {
        # Location that proxies requests to the group
        location / {
            proxy_pass http://appservers;
            health_check;
        }

        # Location for configuration requests
        location /upstream_conf {
            upstream_conf;
            allow 127.0.0.1;
            deny all;
        }
    }
}
```

Here, the access to the second location is allowed only from the 127.0.0.1 address. Access from all other IP addresses is denied.
这里，第二个location只允许来自 127.0.0.1地址的访问，来自其他地址的访问都被拒绝。
To pass a configuration command to NGINX, send an HTTP request by any means. The request should have an appropriate URI to get into the location that includes upstream_conf. The request should include the upstream argument set to the name of the server group. For example, to view all backup servers (marked with backup) in the group, send
要发送一个配置命令给Nginx，通过任何方式发送一个HTTP请求。这个请求应该有一个能够进入包含了upstream_conf的location的URI，还应该包含指定这个服务器组名的参数upstream。例如，要查看该组内所有备份服务器（标记为backup），发送
http://127.0.0.1/upstream_conf?upstream=appservers&backup=
To add a new server to the group, send a request with add and server arguments:
要添加一个服务器到组时，发送一个请求带着add和server参数：
http://127.0.0.1/upstream_conf?add=&upstream=appservers&server=appserv3.example.com:8080&weight=2&max_fails=3
To remove a server, send a request with the remove command and the id argument identifying the server:
要删除一个服务器，发送一个带着remove命令和能找到服务器的ID参数的请求：
http://127.0.0.1/upstream_conf?remove=&upstream=appservers&id=2
To modify a parameter of a specific server, send a request with the id argument identifying the server and the parameter:
要修改某一个服务器的参数，发送请求带着要修改的服务器的id参数来标记服务器和要修改的参数。
http://127.0.0.1/upstream_conf?upstream=appservers&id=2&down=
See the reference (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream_conf) for more examples.
更点信息见引用 (http://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream_conf)。