Nginx-upstream模块

Nginx反向代理之 upstream 模块

upstream模块的内容应放于 nginx.conf 配置的 http{} 标签内，其默认的调度算法是rr （轮循 round-robin）

ngx_http_upstream_module模块官方文档

upstream 模块内部 server 标签参数说明

server标签	参数说明
server 192.168.3.101:80	负载均衡后面的 RS 配置，可以是 IP 或域名，如果端口不写，默认是80端口。高并发场景下，IP可换成域名，通过DNS做负载均衡。
weight=1	代表服务器的权重，默认值为1。权重数字越大表示接受的请求比例越大
max_fails=1	Nginx 尝试连接后端主机失败的次数，这个数值是配合 proxy_next_upstream、fastcgi_next_upstream和 memcached_nex_upstream这三个参数来使用的，当 Nginx接收后端服务器返回这三个参数定义的状态码时，会将这个请求转给正常工作的后端服务器，列如 404、502、503。max_fails 的默认值是1；企业场景：建议2-3次。京东1次，蓝汛10次，根据业务需求去配置。
backup	热备配置（RS节点的高可用），当前面激活的 RS 都失败后会自动启用热备 RS。这标志着这个服务器作为备份服务器，若主服务器全部宕机了，就会向他转发请求；注意，当负载调度算法为ip_hash时，后端服务器在负载均衡调度中的状态不能是 weight 和backup。
fail_timeout=10s	在 max_fails 定义的失败次数后，距离下次检查的间隔时间，默认是10s；如果 max_fails 是5，它就检查5次，如果 5 次都是502。那么，它就会根据fail_timeout的值，等待 10s 再去检查，还是只检查一次，如果持续502，在不重行加载 nginx 配置的情况下，每个10s 都只检测一次。常规业务 2-3秒比较合理，比如京东 3秒，蓝汛 3秒，可根据业务需求去配置。
down	这标志着服务器永远不可用，这个参数可配合 ip_hash使用

提示：以上的参数和专业的 haproxy 参数类似，但不如 haproxy 的参数易懂。

upstream 模块调度算法

调度算法一般分为两类：

第一类为静态调度算法，即负载均衡器根据自身设定的规则进行分配，不需要考虑后端节点服务器的情。例如：rr、wrr、ip_hash等都属于静态调度算法。

第二类为动态调度算法，即负载均衡器会根据后端节点的当前状态来决定是否分发请求，例如：连接数少的有限获得请求，响应时间短的优先获得请求。例如：least_conn、fair 等都属于动态调度算法。

常见的调度算法：

1）rr 轮循 （round robin 默认调度算法，静态调度算法）

按客户端请求顺序把客户端的请求逐一分配到不同的后端节点服务器，这相当于 LVS 中的 rr 算法，如果后端节点服务器宕机（默认情况下nginx 只检测80端口）。宕机的服务器会自动从节点服务器池中剔除，以便客户端的用户访问不受影响。新的请求会分配给正产的服务器。

```
upstream myapp1 {
    server 192.168.3.103;
    server 192.168.3.104;
}
```

2）wrr （weight 权重轮循，静态调度算法）

在 rr 轮循算法的基础上加上权重，即为权重轮循算法，当使用该算法时，权重和用户访问成正比，权重值越大，被转发的请求也就越多。可以根据服务器的配置和性能指定权重值大小，有效解决新旧服务器性能不均带来的请求分配问题。

```
upstream myapp1 {
    server 192.168.3.103 weight=1;
```

```
    server 192.168.3.104 weight=2;
}
```

3）ip_hash（静态调度算法）

每个请求按客户端 IP 的 hash 结果分配，当新的请求到达时，先将其客户端IP通过哈希算法哈希出一个值，在随后的客户端请求中，客户 IP 的哈希值只要相同，就会被分配至同一台服务器，该调度算法可以解决动态网页的 session 共享问题，但有时会导致请求分配不均，即无法保证 1:1 的负载均衡，因为在国内大多数公司都是 NAT 上网模式，多个客户端会对应一个外部 IP，所以，这些客户端都会被分配到同一节点服务器，从而导致请求分配不均。LVS 负载均衡的 -P 参数、keepalived 配置里的 persistence_timeout 50 参数都类似这个 Nginx 里的 ip_hash 参数，其功能均为解决动态网页的 session 共享问题。注意：当负载调度算法为 ip_hash时，后端服务器在负载均衡调度中的状态不能有 weight 和 backup ，即使有也不会生效。

```
upstream myapp1 {
    ip_hash;
    server 192.168.3.103;
    server 192.168.3.104;
}
```

4）fair（动态调度算法）

fair 调度算法会根据后端节点服务器的响应时间来分配请求，响应时间端的优先分配。这是更加智能的调度算法。此种算法可以依据页面大小和加载时间长短只能地进行负载均衡，也就是根据后端服务器的响应时间来分配请求，响应时间短的优先分配。Nginx 本身是不支持 fair 调度算法的，如果需要使用这种调度算法，必须下载 Nginx 的相关模块 upstream_fair。

```
upstream myapp1 {
    fair;
    server 192.168.3.103;
    server 192.168.3.104;
}
```

5）least_conn （动态调度算法）

least_conn 调度算法会根据后端节点的连接数来决定分配情况，哪个机器连接数少就分发。

```
upstream myapp1 {
    least_conn;
    server 192.168.3.103;
    server 192.168.3.104;
}
```

6）url_hash 算法

url_hash 按访问 URL 的 hash 结果来分配请求，使每个 URL 定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率命中率。（多用于后端服务器为缓存时的场景下）Nginx 本身是不支持 rul_hash的，如果需要使用这种调度算法，必须安装 Nginx 的hash 模块软件包。

```
upstream myapp1 {
    server 192.168.3.103;
    server 192.168.3.104;
    hash $request_uri;
    hash_method crc32;
}
```