负载均衡：

当一台服务器单位时间内的访问量越大，服务器压力就越大，大到超过自身承受能力时，服务器就会崩溃。为了避免服务器崩溃，可以通过负载均衡的方式来分担服务器压力。
可以建立很多服务器，组成一个服务器集群，当用户访问网站时，先访问一个中间服务器，让这个中间服务器在集群中选择一个压力较小的服务器，然后将该访问请求引入该服务器，这样保证服务器集群中的每个服务器压力趋于平衡，分担了服务器压力，避免了服务器崩溃的情况。

负载的几种常用方式
1.轮询（默认）
每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除

```
upstream backServer {
    server 192.168.0.14
    server 192.168.0.15
}
```

2.weight
指定轮询几率，weight和访问比率，用于后端服务器性能不均的情况。

```
upstream backserver{
   server 192.168.0.14 weight = 3;
   server 192.168.0.15 weight = 4;
}
```

权重越高，被访问的概率越大，如上例，分别是30%，70%
3.上述方式存在一个问题，在负载均衡系统中，假如用户在某台服务器上登录了，那么该用户第二次请求的时候，因为我们是负载均衡系统，每次请求都会重新定位到服务器集群中的某一个，那么已经登陆某一个服务器的用户再重新定位到另一个服务器，其登陆信息将会丢失。
我们可以采用ip_hash指令解决这个问题，如果客户已经访问了某个服务器，当用户再次访问时，会将该请求通过哈希算法，自动定位该服务器。每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

```
upstream backserver{
  ip_hash;
  server 192.168.0.14;
  server 192.168.0.15;
}
```

4.fair（第三方）按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

```
upstream backserver{
  server 192.168.0.14;
  server 192.168.0.15;
  fair;
}
```

5.url_hash（第三方）

按访问url的hash结果来分配请求，是每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

```
upstream backserver{
server squid1:3128;
server squid2:3128;
hash $request_uri;
hash_method crc32;
}
```