## 一致性hash算法

原理：对于下图来看，节点数越少，越容易出现节点在哈希环上的分布不均匀，导致各节点映射的对象数量严重不均衡(数据倾斜)；相反，节点数越多越密集，数据在哈希环上的分布就越均匀。

设计哈希函数 Hash(key)，要求取值范围为 [0, 2^32)

在对IP地址进行hash运算的结果进行取模（2^32），针对每个全服务器的权重不同，都将IP地址进行hash算法运算，然后进行取模（2^32）。

**原理诠释：**

**varnish   1(权重)**  hash(192.168.37.37)%2^32

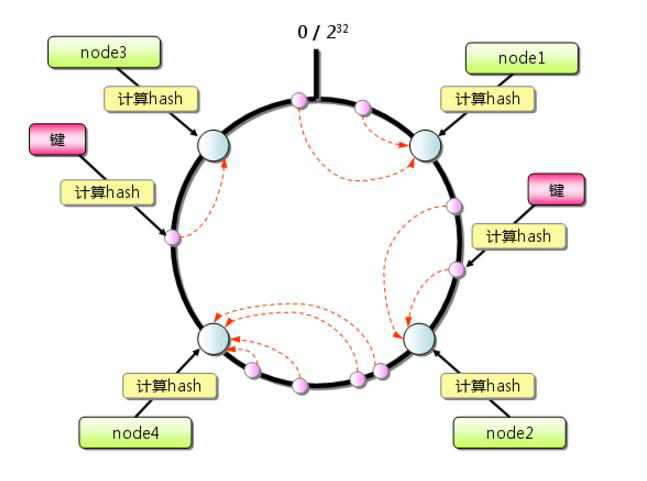
**varnish   2(权重)**  hash（192.168.37.47+random1）%2^32    hash（192.168.37.47+random2）%2^32    权重有两个，对IP地址进行两次hash运算。

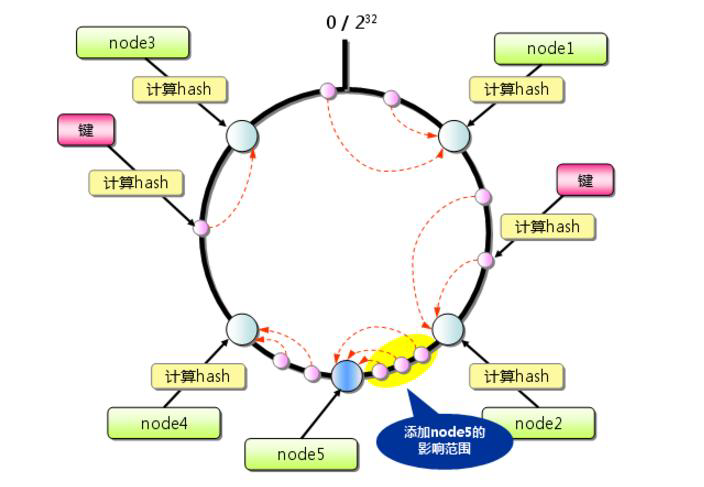
**varnish   3(权重)**  hash（192.168.37.57+random1）%2^32    hash（192.168.37.57+random2）%2^32   hash（192.168.37.57+random3）%2^32   权重有三个，就对IP地址进行三次hash运算。

以上展示，由于在hash环上只有六个随机环，有可能分部在一起，会导致分布不均匀，此时有可能就只有外部两个nginx服务器在工作，中间的四个环不会工作，造成两个环负荷过大。

**解决办法：**在保持权重占比的情况下，可以将三个varnish的权重分别乘以1000，如1\*1000、2\*1000、3\*1000，此时相对的权重占比不变，环的数量增加，会将环的数量增加，不会导致数据倾斜现象，解决了不均匀的问题。

**解决不均匀的代码：hash $request\_uri consistent;**

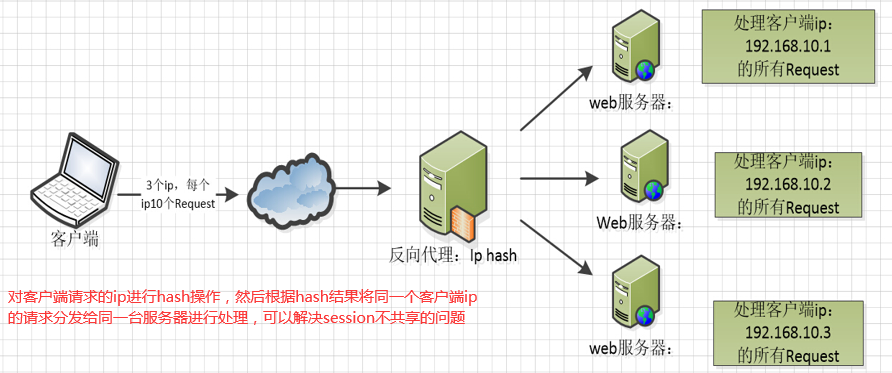




IP-Hash算法

**含义**

　　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应用服务器，可以解决session共享的问题。



**详细配置**

