### [Memcached分布式实现](http://acooly.iteye.com/blog/1120819)

memcached 虽然称为 “ 分布式 ” 缓存服务器，但服务器端并没有 “ 分布式 ” 功能。每个服务器都是完全独立和隔离的服务。 memcached 的分布式，则是完全由客户端程序库实现的。 这种分布式是 memcached 的最大特点。

## 分布式原理

这里多次使用了 “ 分布式 ” 这个词，但并未做详细解释。 现在开始简单地介绍一下其原理，各个客户端的实现基本相同。

下面假设 memcached 服务器有 node1 ～ node3 三台， 应用程序要保存键名为“tokyo”“kanagawa”“chiba”“saitama”“gunma” 的数据。

图 1 分布式简介：准备

首先向 memcached 中添加 “tokyo” 。将 “tokyo” 传给客户端程序库后， 客户端实现的算法就会根据 “ 键 ” 来决定保存数据的 memcached 服务器。 服务器选定后，即命令它保存 “tokyo” 及其值。

图 2 分布式简介：添加时

同样， “kanagawa”“chiba”“saitama”“gunma” 都是先选择服务器再保存。

接下来获取保存的数据。获取时也要将要获取的键 “tokyo” 传递给函数库。 函数库通过与数据保存时相同的算法，根据 “ 键 ” 选择服务器。 使用的算法相同，就能选中与保存时相同的服务器，然后发送 get 命令。 只要数据没有因为某些原因被删除，就能获得保存的值。

图 3 分布式简介：获取时

这样，将不同的键保存到不同的服务器上，就实现了 memcached 的分布式。 memcached 服务器增多后，键就会分散，即使一台 memcached 服务器发生故障 无法连接，也不会影响其他的缓存，系统依然能继续运行。

## 分布式算法

缓存系统中应用比较多的是余数计算分散和一致性 HASH 计算分散。

### 余数计算分散

#### 原理

余数计算分散法简单来说，就是 “ 根据服务器台数的余数进行分散 ” 。

1.       求得传入键的整数哈希值（ int hashCode ）。

2.       使用计算出的 hashCode 除以服务器台数 (N) 取余数（ C=hashCode % N ）

3.       在 N 台服务器中选择序号为 C 的服务器。

#### 特点

余数计算的方法简单，数据的分散性也相当优秀，但也有其缺点。 那就是当添加或移除服务器时，缓存重组的代价相当巨大。 添加服务器后，余数就会产生巨变，这样就无法获取与保存时相同的服务器， 从而影响缓存的命中率。

# Consistent Hashing(一致性 hash 算法)

****算法****

一致性 HASH 算法我的理解，简单来说就是 , 在一个大的数据范围内的构建一个虚拟的环，首（ 0 ）尾（Integer.MAXVALUE ）相接的圆环，然后通过 ****某种**** ****HASH** **算法****增加虚拟节点的方式（ 1 个实体节点可以虚拟 N个虚拟阶段，如 160 ， 200 ， 1000 等）让节点更为均匀的分别在环上。 KEY 请求的时候，也通过相同的****某种HASH** **算法****计算出 HASH 值，然后在在到环上定位同向最接近的虚拟节点，最后通过虚拟节点与实体节点的对应关系找到服务的实体节点。

网上介绍很多，图也多，不想在截取了。那就给个连接：

<http://blog.csdn.net/sparkliang/article/details/5279393>

另外公司现有的项目中也使用 Consistent Hashing 用于分表定位，缓存定位等。工程项目中也有先关算法的实现。

#### 特点

1. 算法实现比较麻烦，需要构建虚拟环。

2. 解决了余数算法增加节点命中大幅额度降低的问题，理论上，插入一个实体节点，平均会影响到：虚拟节点数/2 的节点数据的命中