简单哈希方式

创建哈希表需要先指定大小

当添加的元素达到了装载因子乘以表长时，需要扩容

扩容代价很大，可能影响到所有元素的移动

为了减少扩容时元素的移动，总是将哈希表扩容成原来大小的两倍【数学证明，扩容成两倍大小，能够使得再哈希的元素个数最少】

一致性哈希

在分步式系统中，节点的宕机、某个节点加入或者移出集群是常事，对于分布式储存而言，假设存储集群有十台机器，采用hash方式对数据分片，哈希函数应该是这样：hash(file)%10，这种情况下，扩容是非常不利的，如果要添加一台机器，很多已经存储到机器上的文件都需要重新分配移动，如果文件很大，这种移动就造成了网络的负载。一致性哈希由此提出，修正了使用简单哈希算法的问题，使得分布式哈希可以在P2P环境中得到应用。

一致性哈希提出在动态变化的Cache中，判定哈希算法好坏的四个定义：

平衡性：哈希的结果能够尽可能的分布到所有缓冲中

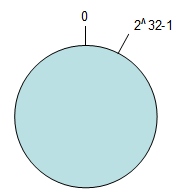
单调性：哈希的结果能够保证原有已分配的内容映射到原有的或者新的缓冲中而非就得缓冲集合

分散性：相同的内容被存储到不同的缓冲中

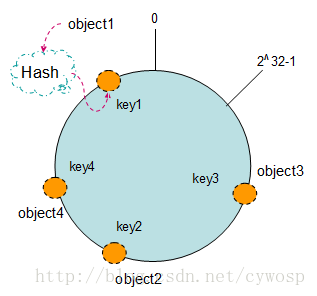
负载：一个特定的缓冲区被用户映射为不同的内容

环形Hash空间

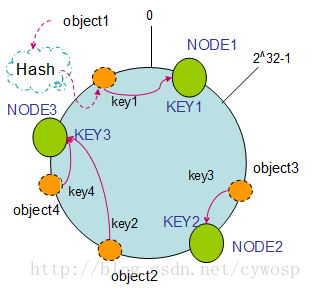
将key哈希到一个具有2^32个桶的空间中，讲这些数字头尾相连，形成一个闭合的环形



将对象通过特定的hash函数计算出key值，散列到hash环上：



将机器通过hash算法映射到环上-当有新的机器加入，通过使用与对象存储一样的hash算法将机器也映射到还中(机器的IP或者机器的别名)，然后以顺时针的方向计算，将所有对象存储到离自己最近的机器中

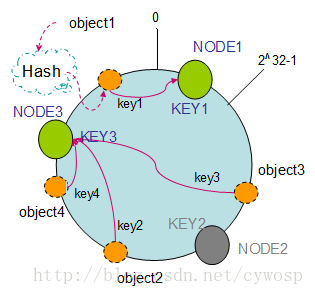


上图看书对象与机器处于同一个哈希空间上，顺拾针转动object1存储到了NODE1中，object3存储到了NODE2中，object2、object4存储到了NODE3中。这样的部署环境中，hash环是不会变更的，通过算出对象的hash值就能快速的定位到对应的机器中，从而找到对象真正的存储位置

机器的删除和添加

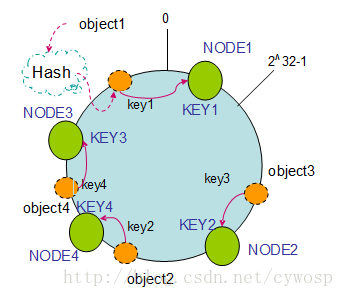
普通的hash求余算法在机器添加或者删除之后会造成大量的对象存储位置失效，一致性哈希这样处理：

1. 节点的删除



如果node2失效，那么object3会迁移到node3上，其他对象没有改动

1. 节点添加增加一个新节点node4



通过顺时针迁移的规则，那么object2被迁移到node4中

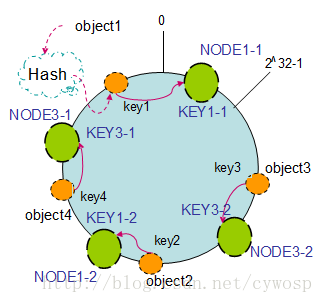
一致性hash的缺点：没有考虑到每台机器的异构性质，不能很好的实现负载均衡—比如说：机器a的配置很高而机器b很低，可能显示中大部分数据都hash到了（a，b）这段环上，直接导致了机器b的存储压力很大；一致性hash还存在热点问题—比如

由于机器b上存储了大量的数据，为了缓解机器b的压力，在环（a，b）段再添加一台机器E，就需要将机器b上的一部分数据拷贝到机器E上，而且只有一台机器，即机器b参与拷贝，这会引起机器b与机器E之间网络负载突然增大，机器b就是我们所说的热点

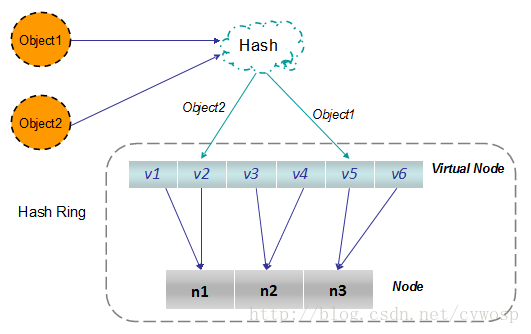
一致性哈希算法的平衡性分析

如上图，如果只部署了node1和node3，那么object1存储到node1，object2、3、4都存储到了node3中，这样就造成了不平衡的状态，在一致性哈希中引入虚拟节点来解决这个问题：

虚拟节点是实际节点在hash空间的复制品，一个实际节点对应了若干虚拟节点(在hash空间时尚以hash值排列)，如果只有node1和node3，以复制个数为2来看：



这样解决了平衡性问题，在实际操作中，对象从hash到虚拟节点到实际节点的转换如下



虚拟节点的hash计算可以采用对应节点的IP地址加数字后缀的方式

{http://blog.csdn.net/cywosp/article/details/23397179/}

分布式哈希查询过程

当某台机器接收到查询请求时，先待查找的数据是否在本地，如果不在本地，则将查询请求转发到顺时针方向的写一个节点上，这种查询方式最坏情况下时间复杂度O(N)，改进的方法：在每台机器上维护一些路由信息，即那些key存储在哪些节点上，机器1本地存储着 items为1的数据；机器2本地存储着 items为2的数据；机器6本地存储着 items为3，4，5，6的数据；机器0本地存储着items为7，0 的数据。

与此同时，机器1上，还存储着items 为2，3，5 的数据 所在的机器地址。比如，机器1知道，items为5的数据存储在机器6上面。

比如，当机器1接收到 查询 items为7的数据在哪台机器上时，它发现 items=7 比它的路由表中的最大的items5还大，于是将查询请求转发到 items=5所对应的机器6上。

机器6上的路由表指出：items=7的数据在机器0上，于是就找到了items=7的数据 在机器0上了。

对于每步的查询而言，目标就是：尽可能地将查询key发送到离存储这个key最近的那台机器上。比如，上面的机器1接收到 key 为 items7 的查询，机器1上存储着 key为 2,3,5的数据，但是由于items 7 比items 2，3，5都要大，因此它判断出存储 items=5 的那台机器 距离 存储items=7的机器 更近，故把查询请求转发给 items=5 所在的那台机器（机器6）