memcache存储机制

2.1、存储机制

Memcache采用的是Slab Allocator方式进行存储数据。这一机制可以很好的整理内存，以便重复利用，从而解决了内存碎片的问题。在该机制出现以前，内存的分配是通过对所有记录简单地进行malloc和free来进行的。但是，这种方式会导致内存碎片，加重操作系统内存管理器的负担，最坏的情况下，会导致操作系统比memcached进程本身还慢。

2.2、Slab Allocator基本原理

1、按照预先规定的大小，将分配的内存以page（默认每个page为1M）为单位分为特定的块（chunk），并且把相同大小的chunk分成组（chunk的集合）；

2、存储数据时，将会寻找与value大小相近的chunk区域进行存储；

3、内存一旦以page的形式分配出去，在重启前不会被回收或者重新分配，以解决内存碎片问题。（分配的内存不会释放，而是重复利用）

2.3、理解四个名词

【可参考下面的形象解析图进行理解】

Slab

用于表示存储的最大size数据，仅仅只是用于定义（通俗的讲就是表示可以存储数据大小的范围）。默认情况下，前后两个slab表示存储的size以1.25倍进行增长。例如slab1为96字节，slab2为120字节

Page

分配给Slab的内存空间，默认为1MB。分给Slab后将会根据slab的大小切割成chunk

Chunk

用于缓存记录的内存空间

Slab calss

特定大小的Chunk集合

2.4、Slab的内存分配具体过程

Memcached在启动时通过**-m参数**指定最大使用内存，但是这个不会一启动就占用完，而是逐步分配给各slab的。如果一个新的数据要被存放，首先选择一个合适的slab，然后查看该slab是否还有空闲的chunk，如果有则直接存放进去；如果没有则要进行申请，slab申请内存时以page为单位，无论大小为多少，都会有1M大小的page被分配给该slab（该page不会被回收或者重新分配，永远都属于该slab）。申请到page后，slab会将这个page的内存按chunk的大小进行切分，这样就变成了一个chunk的数组，再从这个chunk数组中选择一个用于存储数据。若没有空闲的page的时候，则会对改slab进行LRU，而不是对整个memcache进行LRU。

Memcached并不是将所有大小的数据都放在一起的，而是预先将数据空间划分为一系列slabs，每个slab只负责一定范围内的数据存储。memcached根据收到的数据的大小，选择最适合数据大小的slab。假若这个slab仍有空闲chunk的列表，根据该列表选择chunk，然后将数据缓存于其中；若无则申请page（1M）【可以参考上面我画的形象图23333】

具体分析：从上面我们了解到slab的作用。Slab的增长因子默认以1.25倍进行增长。那为什么会导致有些不是1.25倍呢？答案是受小数的影响，你可以使用-f int测试个整数增长因子看看效果。【后面具体讲解】

以下图进行分析，例如slab中112字节，表示可以存储大于88字节且小于或等于112字节的value。

slab的缺点

Slab Allocator解决了当初的内存碎片问题，但新的机制也给memcached带来了新的问题。

这个问题就是，由于分配的是特定长度的内存，因此无法有效利用分配的内存。例如，将100字节 的数据缓存到128字节的chunk中，剩余的28字节就浪费了（如下图所示）。