**memcached 是怎么工作的？**

Memcached 的神奇来自两阶段哈希（two-stage hash）。Memcached 就像一个巨大的、存储了很多<key,value>对的哈希表。通过 key，可以存储或查询任意的数据。

客户端可以把数据存储在多台 memcached 上。当查询数据时，客户端首先参考节点列表计算出 key 的哈希值（阶段一哈希），进而选中一个节点；客户端将请求发送给选中的节点，然后 memcached 节点通过一个内部的哈希算法（阶段二哈希），查找真正的数据（item）。

举个列子，假设有 3 个客户端 1, 2, 3，3 台 memcached A, B, C：
Client 1 想把数据"barbaz"以 key "foo"存储。Client 1 首先参考节点列表（A, B, C），计算 key "foo"的哈希值，假设 memcached B 被选中。接着，Client 1 直接 connect 到 memcached B，通过 key "foo"把数据"barbaz"存储进去。 Client 2 使用与 Client 1 相同的客户端库（意味着阶段一的哈希算法相同），也拥有同样的 memcached 列表（A, B, C）。
于是，经过相同的哈希计算（阶段一），Client 2 计算出 key "foo"在 memcached B 上，然后它直接请求 memcached B，得到数据"barbaz"。

各种客户端在 memcached 中数据的存储形式是不同的（perl Storable, php serialize, java hibernate, JSON 等）。一些客户端实现的哈希算法也不一样。但是，memcached 服务器端的行为总是一致的。

最后，从实现的角度看，memcached 是一个非阻塞的、基于事件的服务器程序。这种架构可以很好地解决 C10K problem ，并具有极佳的可扩展性。

可以参考 A Story of Caching ，这篇文章简单解释了客户端与 memcached 是如何交互的。

**memcached 最大的优势是什么？**

请仔细阅读上面的问题（即 memcached 是如何工作的）。Memcached 最大的好处就是它带来了极佳的水平可扩展性，特别是在一个巨大的系统中。由于客户端自己做了一次哈希，那么我们很容易增加大量 memcached 到集群中。memcached 之间没有相互通信，因此不会增加 memcached 的负载；没有多播协议，不会网络通信量爆炸（implode）。memcached 的集群很好用。内存不够了？增加几台 memcached 吧；CPU 不够用了？再增加几台吧；有多余的内存？在增加几台吧，不要浪费了。

基于 memcached 的基本原则，可以相当轻松地构建出不同类型的缓存架构。除了这篇 FAQ，在其他地方很容易找到详细资料的。

看看下面的几个问题吧，它们在 memcached、服务器的 local cache 和 MySQL 的 query cache 之间做了比较。这几个问题会让您有更全面的认识。

**memcached 和 MySQL 的 query cache 相比，有什么优缺点？**

把 memcached 引入应用中，还是需要不少工作量的。MySQL 有个使用方便的 query cache，可以自动地缓存 SQL 查询的结果，被缓存的 SQL 查询可以被反复地快速执行。Memcached 与之相比，怎么样呢？MySQL 的 query cache 是集中式的，连接到该 query cache 的 MySQL 服务器都会受益。

* 当您修改表时，MySQL 的 query cache 会立刻被刷新（flush）。存储一个 memcached item 只需要很少的时间，但是当写操作很频繁时，MySQL 的 query cache 会经常让所有缓存数据都失效。

* 在多核 CPU 上，MySQL 的 query cache 会遇到扩展问题（scalability issues）。在多核 CPU 上，query cache 会增加一个全局锁（global lock），由于需要刷新更多的缓存数据，速度会变得更慢。

* 在 MySQL 的 query cache 中，我们是不能存储任意的数据的（只能是 SQL 查询结果）。而利用 memcached，我们可以搭建出各种高效的缓存。比如，可以执行多个独立的查询，构建出一个用户对象（user object），然后将用户对象缓存到 memcached 中。而 query cache 是 SQL 语句级别的，不可能做到这一点。在小的网站中，query cache 会有所帮助，但随着网站规模的增加，query cache 的弊将大于利。

* query cache 能够利用的内存容量受到 MySQL 服务器空闲内存空间的限制。给数据库服务器增加更多的内存来缓存数据，固然是很好的。但是，有了 memcached，只要您有空闲的内存，都可以用来增加 memcached 集群的规模，然后您就可以缓存更多的数据。

**memcached 和服务器的 local cache（比如 PHP 的 APC、mmap 文件等）相比，有什么优缺点？**

首先，local cache 有许多与上面(query cache)相同的问题。local cache 能够利用的内存容量受到（单台）服务器空闲内存空间的限制。不过，local cache 有一点比 memcached 和 query cache 都要好，那就是它不但可以存储任意的数据，而且没有网络存取的延迟。

* local cache 的数据查询更快。考虑把 highly common 的数据放在 local cache 中吧。如果每个页面都需要加载一些数量较少的数据，考虑把它们放在 local cached 吧。

* local cache 缺少集体失效（group invalidation）的特性。在 memcached 集群中，删除或更新一个 key 会让所有的观察者觉察到。但是在 local cache 中， 我们只能通知所有的服务器刷新 cache（很慢，不具扩展性），或者仅仅依赖缓存超时失效机制。

* local cache 面临着严重的内存限制，这一点上面已经提到。

**memcached 的 cache 机制是怎样的？**

Memcached 主要的 cache 机制是 LRU（最近最少用）算法+超时失效。当您存数据到 memcached 中，可以指定该数据在缓存中可以呆多久 Which is forever, or some time in the future。如果 memcached 的内存不够用了，过期的 slabs 会优先被替换，接着就轮到最老的

未被使用的 slabs。

**memcached 如何实现冗余机制？**

不实现！我们对这个问题感到很惊讶。Memcached 应该是应用的缓存层。它的设计本身就不带有任何冗余机制。如果一个 memcached 节点失去了所有数据，您应该可以从数据源（比如数据库）再次获取到数据。您应该特别注意，您的应用应该可以容忍节点的失效。不要写一些糟糕的查询代码，寄希望于 memcached 来保证一切！如果您担心节点失效会大大加重数据库的负担，那么您可以采取一些办法。比如您可以增加更多的节点（来减少丢失一个节点的影响），热备节点（在其他节点 down 了的时候接管 IP），等等。

**memcached 如何处理容错的？**

不处理！:) 在 memcached 节点失效的情况下，集群没有必要做任何容错处理。如果发生了节点失效，应对的措施完全取决于用户。节点失效时，下面列出几种方案供您选择：

* 忽略它！ 在失效节点被恢复或替换之前，还有很多其他节点可以应对节点失效带来的影响。

* 把失效的节点从节点列表中移除。做这个操作千万要小心！在默认情况下（余数式哈希算法），客户端添加或移除节点，会导致所有的缓存数据不可用！因为哈希参照的节点列表变化了，大部分 key 会因为哈希值的改变而被映射到（与原来）不同的节点上。

* 启动热备节点，接管失效节点所占用的 IP。这样可以防止哈希紊乱（hashing chaos）。

* 如果希望添加和移除节点，而不影响原先的哈希结果，可以使用一致性哈希算法（consistent hashing）。您可以百度一下一致性哈希算法。支持一致性哈希的客户端已经很成熟，而且被广泛使用。去尝试一下吧！

* 两次哈希（reshing）。当客户端存取数据时，如果发现一个节点 down 了，就再做一次哈希（哈希算法与前一次不同），重新选择另一个节点（需要注意的时，客户端并没有把 down 的节点从节点列表中移除，下次还是有可能先哈希到它）。如果某个节点时好时坏，两次哈希的方法就有风险了，好的节点和坏的节点上都可能存在脏数据（stale data）。

**如何将 memcached 中 item 批量导入导出？**

您不应该这样做！Memcached 是一个非阻塞的服务器。任何可能导致 memcached 暂停或瞬时拒绝服务的操作都应该值得深思熟虑。向 memcached 中批量导入数据往往不是您真正想要的！想象看，如果缓存数据在导出导入之间发生了变化，您就需要处理脏数据了；如果缓存数据在导出导入之间过期了，您又怎么处理这些数据呢？

因此，批量导出导入数据并不像您想象中的那么有用。不过在一个场景倒是很有用。如果您有大量的从不变化的数据，并且希望缓存很快热（warm）起来，批量导入缓存数据是很有帮助的。虽然这个场景并不典型，但却经常发生，因此我们会考虑在将来实现批量导出导入的功能。

Steven Grimm，一如既往地,，在邮件列表中给出了另一个很好的例子：http://lists.danga.com/pipermail/memcached/2007-July/004802.html 。

**我需要把 memcached 中的 item 批量导出导入，怎么办？**

好吧好吧。如果您需要批量导出导入，最可能的原因一般是重新生成缓存数据需要消耗很长的时间，或者数据库坏了让您饱受痛苦。

如果一个 memcached 节点 down 了让您很痛苦，那么您还会陷入其他很多麻烦。您的系统太脆弱了。您需要做一些优化工作。比如处理"惊群"问题（比如 memcached 节点都失效了，反复的查询让您的数据库不堪重负…这个问题在 FAQ 的其他提到过），或者优化不好的查询。记住，Memcached 并不是您逃避优化查询的借口。

如果您的麻烦仅仅是重新生成缓存数据需要消耗很长时间（15 秒到超过 5 分钟），您可以考虑重新使用数据库。这里给出一些提示：

* 使用 MogileFS（或者 CouchDB 等类似的软件）在存储 item。把 item 计算出来并 dump 到磁盘上。MogileFS 可以很方便地覆写 item，并提供快速地访问。您甚至可以把 MogileFS 中的 item 缓存在 memcached 中，这样可以加快读取速度。 MogileFS+Memcached 的组合可以加快缓存不命中时的响应速度，提高网站的可用性。
* 重新使用 MySQL。MySQL 的 InnoDB 主键查询的速度非常快。如果大部分缓存数据都可以放到 VARCHAR 字段中，那么主键查询的性能将更好。从 memcached 中按 key 查询几乎等价于 MySQL 的主键查询：将 key 哈希到 64-bit 的整数，然后将数据存储到 MySQL 中。您可以把原始（不做哈希）的 key 存储都普通的字段中，然后建立二级索引来加快查询…key 被动地失效，批量删除失效的 key，等等。

上面的方法都可以引入 memcached，在重启 memcached 的时候仍然提供很好的性能。由于您不需要当心"hot"的 item 被 memcached LRU 算法突然淘汰，用户再也不用花几分钟来等待重新生成缓存数据（当缓存数据突然从内存中消失时），因此上面的方法可以全面提高性能。

关于这些方法的细节，详见博客：http://dormando.livejournal.com/495593.html 。

**memcached 是如何做身份验证的？**

没有身份认证机制！memcached 是运行在应用下层的软件（身份验证应该是应用上层的职责）。memcached 的客户端和服务器端之所以是轻量级的，部分原因就是完全没有实现身份验证机制。这样，memcached 可以很快地创建新连接，服务器端也无需任何配置。

如果您希望限制访问，您可以使用防火墙，或者让 memcached 监听 unix domain socket。

**memcached 的多线程是什么？如何使用它们？**

线程就是定律（threads rule）！在 Steven Grimm 和 Facebook 的努力下，memcached 1.2 及更高版本拥有了多线程模式。多线程模式允许 memcached 能够充分利用多个 CPU，并在 CPU 之间共享所有的缓存数据。memcached 使用一种简单的锁机制来保证数据更新操作的

互斥。相比在同一个物理机器上运行多个 memcached 实例，这种方式能够更有效地处理 multi gets。

如果您的系统负载并不重，也许您不需要启用多线程工作模式。如果您在运行一个拥有大规模硬件的、庞大的网站，您将会看到多线程的好处。

简单地总结一下：命令解析（memcached 在这里花了大部分时间）可以运行在多线程模式下。memcached 内部对数据的操作是基于很多全局锁的（因此这部分工作不是多线程的）。未来对多线程模式的改进，将移除大量的全局锁，提高 memcached 在负载极高的场景下的性能。

**memcached 能接受的 key 的最大长度是多少？**

key 的最大长度是 250 个字符。需要注意的是，250 是 memcached 服务器端内部的限制，如果您使用的客户端支持"key 的前缀"或类似特性，那么 key（前缀+原始 key）的最大长度是可以超过 250 个字符的。我们推荐使用使用较短的 key，因为可以节省内存和带宽。

**memcached 对 item 的过期时间有什么限制？**

过期时间最大可以达到 30 天。memcached 把传入的过期时间（时间段）解释成时间点后，一旦到了这个时间点，memcached 就把 item 置为失效状态。这是一个简单但 obscure 的机制。

**memcached 最大能存储多大的单个 item？**

1MB。如果你的数据大于 1MB，可以考虑在客户端压缩或拆分到多个 key 中。

**为什么单个 item 的大小被限制在 1M byte 之内？**

简单的回答：因为内存分配器的算法就是这样的。

详细的回答：Memcached 的内存存储引擎（引擎将来可插拔…），使用 slabs 来管理内存。内存被分成大小不等的 slabs　chunks（先分成大小相等的 slabs，然后每个 slab 被分成大小相等 chunks，不同 slab 的 chunk 大小是不相等的）。chunk 的大小依次从一个最小数开始，按某个因子增长，直到达到最大的可能值。

如果最小值为 400B，最大值是 1MB，因子是 1.20，各个 slab 的 chunk 的大小依次是：
slab1 – 400B slab2 – 480B slab3 – 576B …

slab 中 chunk 越大，它和前面的 slab 之间的间隙就越大。因此，最大值越大，内存利用率越低。Memcached 必须为每个 slab 预先分配内存，因此如果设置了较小的因子和较大的最大值，会需要更多的内存。

还有其他原因使得您不要这样向 memcached 中存取很大的数据…不要尝试把巨大的网页放到 mencached 中。把这样大的数据结构 load 和 unpack 到内存中需要花费很长的时间，从而导致您的网站性能反而不好。

如果您确实需要存储大于 1MB 的数据，你可以修改 slabs.c:POWER_BLOCK 的值，然后重新编译 memcached；或者使用低效的 malloc/free。其他的建议包括数据库、MogileFS 等。

我可以在不同的 memcached 节点上使用大小不等的缓存空间吗？这么做之后，memcached 能够更有效地使用内存吗？

Memcache 客户端仅根据哈希算法来决定将某个 key 存储在哪个节点上，而不考虑节点的内存大小。因此，您可以在不同的节点上使用大小不等的缓存。但是一般都是这样做的：拥有较多内存的节点上可以运行多个 memcached 实例，每个实例使用的内存跟其他节点上的实例相同。