Memcached 面试题

1、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。

如果你的系统负载并不重，也许你不需要启用多线程工作模式。如果你在运行一个拥有大规模硬件的、庞大的网站，你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。

2、Memcached 是什么，有什么作用？

Memcached 是一个开源的，高性能的内存绶存软件，从名称上看 Mem 就是内存的意思，而 Cache 就是缓存的意思。Memcached 的作用：通过在事先规划好的内存空间中临时绶存数据库中的各类数据，以达到减少业务对数据库的直接高并发访问，从而达到提升数据库的访问性能，加速网站集群动态应用服

务的能力。

3、Memcached 与 Redis 的区别？

- Redis 不仅仅支持简单的 K/V 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。而 memcache 只支持简单数据类型，需要客户端自己处理复杂对象。
- Redis 支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用（PS：持久化在 rdb、aof）。
- 由于 Memcache 没有持久化机制，因此宕机所有缓存数据失效。Redis 配置为持久化，宕机重启后，将自动加载宕机时刻的数据到缓存系统中。具有更好的灾备机制。
- Memcache 可以使用 Magent 在客户端进行一致性 hash 做分布式。Redis 支持在服务器端做分布式（PS:Twemproxy/Codis/Redis-cluster 多种分布式实现方式）。
- Memcached 的简单限制就是键（key）和 Value 的限制。最大键长为 250 个字符。可以接受的储存数据不能超过 1MB（可修改配置文件变大），因为这是典型 slab 的最大值，不适合虚拟机使用。而 Redis 的 Key 长度支持到 512K。
- Redis 使用的是单线程模型，保证了数据按顺序提交。Memcache 需要使用 cas 保证数据一致性。CAS（Check and Set）是一个确保并发一致性的机制，属于“乐观锁”范畴；原理很简单：拿版本号，操作，对比版本

号，如果一致就操作，不一致就放弃任何操作。

- CPU 利用：由于 Redis 只使用单核，而 Memcached 可以使用多核，所以平均每一个核上 Redis 在存储小数据时比 Memcached 性能更高。而在 100k 以上的数据中，Memcached 性能要高于 Redis 。
- Memcached 内存管理：使用 Slab Allocation。原理相当简单，预先分配一系列大小固定的组，然后根据数据大小选择最合适的块存储。避免了内存碎片。（缺点：不能变长，浪费了一定空间）Memcached 默认情况下下一个 slab 的最大值为前一个的 1.25 倍。
- Redis 内存管理： Redis 通过定义一个数组来记录所有的内存分配情况，Redis 采用的是包装的 malloc/free，相较于 Memcached 的内存 管理方法来说，要简单很多。由于 malloc 首先以链表的方式搜索已管理的内存中可用的空间分配，导致内存碎片比较多。

4、什么是二进制协议，我该关注吗？

关于二进制最好的信息当然是二进制协议规范：

二进制协议尝试为端提供一个更有效的、可靠的协议，减少客户端/服务器端因处理协议而产生的 CPU 时间。

根据 Facebook 的测试，解析 ASCII 协议是 Memcached 中消耗 CPU 时间最多的环节。所以，我们为什么不改进 ASCII 协议呢？

5、如果缓存数据在导出导入之间过期了，你又怎么处理这些数据呢？

因此，批量导出导入数据并不像你想象中的那么有用。不过在一个场景倒是很有用。如果你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。

如果一个 Memcached 节点 down 了让你很痛苦，那么你还会陷入其他很多麻烦。你的系统太脆弱了。你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，反复的查询让你的数据库不堪重负…这个问题在 FAQ 的其他提到过），或者优化不好的查询。记住，Memcached 并不是你逃避优化查询的借口。

6、如何实现集群中的 session 共享存储？

Session 是运行在一台服务器上的，所有的访问都会到达我们的唯一服务器上，这样我们可以根据客户端传来的 sessionID，来获取 session，或在对应 Session 不存在的情况下（session 生命周期到了/用户第一次登录），创建一个新的 Session；但是，如果我们在集群环境下，假设我们有两台服务器 A，B，用户的请求会由 Nginx 服务器进行转发（别的方案也是同理），用户登录时，Nginx 将请求转发至服务器 A 上，A 创建了新的 session，并将 SessionID 返回给客户端，用户在浏览其他页面时，客户端验证登录状态，

Nginx 将请求转发至服务器 B，由于 B 上并没有对应客户端发来 sessionId 的 session，所以会重新创建一个新的 session，并且再将这个新的 sessionID 返回给客户端，这样，我们可以想象一下，用户每一次操作都有 1/2 的概率进行再次的登录，这样不仅对用户体验特别差，还会让服务器上的 session 激增，加大服务器的运行压力。

为了解决集群环境下的 seesion 共享问题，共有 4 种解决方案：

1、 粘性 session

粘性 session 是指 Ngnix 每次都将同一用户的所有请求转发至同一台服务器上，即将用户与服务器绑定。

2、 服务器 session 复制

即每次 session 发生变化时，创建或者修改，就广播给所有集群中的服务器，使所有的服务器上的 session 相同。

3、 session 共享

缓存 session，使用 Redis， Memcached。

4、 session 持久化

将 session 存储至数据库中，像操作数据一样才做 session。

7、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。

1、 当你修改表时，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。

2、 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。

3、 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。

4、 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，只要你有空闲的内存，都可以用来增加 Memcached 集群的规模，然后你就可以缓存更多的数据。

8、Memcached 是原子的吗？

所有的被发送到 Memcached 的单个命令是完全原子的。如果你针对同一份数

据同时发送了一个 set 命令和一个 get 命令，它们不会影响对方。它们将被串行化、先后执行。即使在多线程模式，所有的命令都是原子的，除非程序有 bug。

命令序列不是原子的。如果你通过 get 命令获取了一个 item，修改了它，然后想把它 set 回 Memcached，我们不保证这个 item 没有被其他进程（process，未必是操作系统中的进程）操作过。在并发的情况下，你也可能覆写了一个被其他进程 set 的 item。

Memcached 1.2.5 以及更高版本，提供了 gets 和 cas 命令，它们可以解决上面的问题。如果你使用 gets 命令查询某个 key 的 item，Memcached 会给你返回该 item 当前值的唯一标识。如果你覆写了这个 item 并想把它写回到 Memcached 中，你可以通过 cas 命令把那个唯一标识一起发送给 Memcached。如果该 item 存放在 Memcached 中的唯一标识与你提供的一致，你的写操作将会成功。如果另一个进程在这期间也修改了这个 item，那么该 item 存放在 Memcached 中的唯一标识将会改变，你的写操作就会失败。

9、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，你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。

10、Memcached 的内存分配器是如何工作的？为什么不适用 malloc/free？为何要使用 slabs？

实际上，这是一个编译时选项。默认会使用内部的 slab 分配器。你确实确实应该使用内建的 slab 分配器。最早的时候，Memcached 只使用 malloc/free 来管理内存。然而，这种方式不能与 OS 的内存管理以前很好地工作。反复地 malloc/free 造成了内存碎片，OS 最终花费大量的时间去查找连续的内存块来满足 malloc 的请求，而不是运行 Memcached 进程。如果你不同意，当然可以使用 malloc。

slab 分配器就是为了解决这个问题而生的。内存被分配并划分成 chunks，一直被重复使用。因为内存被划分成大小不等的 slabs，如果 item 的大小与被选择存放它的 slab 不是很合适的话，就会浪费一些内存。Steven Grimm 正在这方面已经做出了有效的改进。