链滴

10 个 Redis 建议 / 技巧

作者：abners

原文链接：https://ld246.com/article/1472994547164

来源网站：链滴

本文由伯乐在线 jasper 校稿，翻自 objectrocket.om
Redis 在当前的技术社区里是非常热门的。从来自 Antirez 一个小小的个人项目到成为内存数据储行业的标准，Redis 已经走过了很长的一段路。随之而来的一系列最佳实践，使得大多数人可以正地使用 Redis。下面我们将探索正确使用 Redis 的 10 个技巧。
<h4 id="1-停止使用-KEYS--">1、停止使用 KEYS *</h4>
Okay，以挑战这个命令开始这篇文章，或许并不是一个好的方式，但其确实可能是最重要的一。很多时候当我们关注一个 redis 实例的统计数据，我们会快速地输入” KEYS *” 命令，这样 key 的息会很明显地展示出来。平心而论，从程序化的角度出发往往倾向于写出下面这样的伪代码：
<pre><code class="highlight-chroma">for key in'keys *':

doAllTheThings()
</code></pre>
但是当你有 1300 万个 key 时，执行速度将会变慢。因为 KEYS 命令的时间复杂度是 O(n)，其中 n 是要返回的 keys 的个数，这样这个命令的复杂度就取决于数据库的大小了。并且在这个操作执行间，其它任何命令在你的实例中都无法执行。
作为一个替代命令，看一下 SCAN 吧，其允许你以一种更友好的方式来执行... SCAN 通过增量代的方式来扫描数据库。这一操作基于游标的迭代器来完成的，因此只要你觉得合适，你可以随时停或继续。
<h4 id="2-找出拖慢-Redis-的罪魁祸首">2、找出拖慢 Redis 的罪魁祸首</h4>
由于 Redis 没有非常详细的日志，要想知道在 Redis 实例内部都做了些什么是非常困难的。幸的是 Redis 提供了一个下面这样的命令统计工具：
<pre><code class="highlight-chroma">127.0.0.1:6379> INFO commandstats
Commandstats
cmdstat_get:calls 78,usec=608,usec_per_call=7.79
cmdstat_setex:call =5,usec=71,usec_per_call=14.20
cmdstat_keys:call =2,usec=42,usec_per_call=21.00
cmdstat_info:calls 10,usec=1931,usec_per_call=193.10
</code></pre>
通过这个工具可以查看所有命令统计的快照，比如命令执行了多少次，执行命令所耗费的毫秒数每个命令的总时间和平均时间)
只需要简单地执行 CONFIG RESETSTAT 命令就可以重置，这样你就可以得到一个全新的统计结。
<h3 id="3-将-Redis-Benchmark-结果作为参考-而不要一概而论">3、将 Redis-Benchmark 结果为参考，而不要一概而论</h3>
Redis 之父 Salvatore 就说过：“通过执行 GET/SET 命令来测试 Redis 就像在雨天检测法拉利雨刷清洁镜子的效果”。很多时候人们跑到我这里，他们想知道为什么自己的 Redis-Benchmark 统的结果低于最优结果 。但我们必须要把各种不同的真实情况考虑进来，例如：

可能受到哪些客户端运行环境的限制？
是同一个版本号吗？
测试环境中的表现与应用将要运行的环境是否一致？

Redis-Benchmark 的测试结果提供了一个保证你的 Redis-Server 不会运行在非正常状态下的准点，但是你永远不要把它作为一个真实的“压力测试”。压力测试需要反应出应用的运行方式，并

需要一个尽可能的和生产相似的环境。
<h4 id="4-Hashes-是你的最佳选择">4、Hashes 是你的最佳选择</h4>
以一种优雅的方式引入 hashes 吧。hashes 将会带给你一种前所未有的体验。之前我曾看到过多类似于下面这样的 key 结构：
<pre><code class="highlight-chroma">foo:first_name
foo:last_name
foo:address
</code></pre>
上面的例子中，foo 可能是一个用户的用户名，其中的每一项都是一个单独的 key。这就增加了错的空间，和一些不必要的 key。使用 hash 代替吧，你会惊奇地发现竟然只需要一个 key：
<pre><code class="highlight-chroma">127.0.0.1:6379> HSET foo first_name "Joe"(integer) 1
127.0.0.1:6379> HSET foo last_name "Engel"(integer) 1
127.0.0.1:6379> HSET foo address "1 Fanatical Pl"(integer) 1
127.0.0.1:6379> HGETALL foo
1)"first_name"
2)"Joe"
3)"last_name"
4)"Engel"
5)"address"
6)"1 Fanatical Pl"
127.0.0.1:6379> HGET foo first_name
"Joe"
</code></pre>
<h4 id="5-设置-key-值的存活时间">5、设置 key 值的存活时间</h4>
无论什么时候，只要有可能就利用 key 超时的优势。一个很好的例子就是储存一些诸如临时认证 ey 之类的东西。当你去查找一个授权 key 时——以 OAUTH 为例——通常会得到一个超时时间。这在设置 key 的时候，设成同样的超时时间，Redis 就会自动为你清除！而不再需要使用 KEYS *来遍所有的 key 了，怎么样很方便吧？
<h4 id="6-选择合适的回收策略">6、选择合适的回收策略</h4>
既然谈到了清除 key 这个话题，那我们就来聊聊回收策略。当 Redis 的实例空间被填满了之后将会尝试回收一部分 key。根据你的使用方式，我强烈建议使用 volatile-lru 策略——前提是你对 key 已经设置了超时。但如果你运行的是一些类似于 cache 的东西，并且没有对 key 设置超时机制，可考虑使用 allkeys-lru 回收机制。我的建议是先在这里查看一下可行的方案。
<h4 id="7-如果你的数据很重要-请使用-Try-Except">7、如果你的数据很重要，请使用 Try/Except /h4>
如果必须确保关键性的数据可以被放入到 Redis 的实例中，我强烈建议将其放入 try/except 块 。几乎所有的 Redis 客户端采用的都是“发送即忘”策略，因此经常需要考虑一个 key 是否真正被到 Redis 数据库中了。至于将 try/expect 放到 Redis 命令中的复杂性并不是本文要讲的，你只需要道这样做可以确保重要的数据放到该放的地方就可以了。
<h4 id="8-不要耗尽一个实例">8、不要耗尽一个实例</h4>
无论什么时候，只要有可能就分散多 redis 实例的工作量。从 3.0.0 版本开始，Redis 就支持集了。Redis 集群允许你基于 key 范围分离出部分包含主/从模式的 key。完整的集群背后的“魔法”以在这里找到。但如果你是在找教程，那这里是一个再适合不过的地方了。如果不能选择集群，考虑下命名空间吧，然后将你的 key 分散到多个实例之中。关于怎样分配数据，在 redis.io 网站上有这篇彩的评论。
<h4 id="9-内核越多越好吗--">9、内核越多越好吗？！</h4>
当然是错的。Redis 是一个单线程进程，即使启用了持久化最多也只会消耗两个内核。除非你计在一台主机上运行多个实例——希望只会是在开发测试的环境下！——否则的话对于一个 Redis 实

是不需要 2 个以上内核的。
<h4 id="10-高可用">10、高可用</h4>
到目前为止 Redis Sentinel 已经经过了很全面的测试，很多用户已经将其应用到了生产环境中包括 ObjectRocket ）。如果你的应用重度依赖于 Redis ，那就需要想出一个高可用方案来保证其不掉线。当然，如果不想自己管理这些东西，ObjectRocket 提供了一个高可用平台，并提供 7×24 小的技术支持，有意向的话可以考虑一下。