拓展 6：平波缓进 —— 懒惰删除

一直以来我们认为 Redis 是单线程的，单线程为 Redis 带来了代码的简洁性和丰富多样的数据结构。不过Redis内部实际上并不是只有一个主线程，它还有几个异步线程专门用来处理一些耗时的操作。

Redis 为什么要懒惰删除(lazy free)?

删除指令 `del` 会直接释放对象的内存，大部分情况下，这个指令非常快，没有明显延迟。不过如果删除的 key 是一个非常大的对象，比如一个包含了千万元素的 hash，那么删除操作就会导致单线程卡顿。

Redis 为了解决这个卡顿问题，在 4.0 版本引入了 `unlink` 指令，它能对删除操作进行懒处理，丢给后台线程来异步回收内存。

```
> unlink key
OK
```

如果有多线程的开发经验，你肯定会担心这里的线程安全问题，会不会出现多个线程同时并发修改数据结构的情况存在。

关于这点，我打个比方。可以将整个 Redis 内存里面所有有效的数据想象成一棵大树。当 `unlink` 指令发出时，它只是把大树中的一个树枝别断了，然后扔到旁边的火堆里焚烧 (异步线程池)。树枝离开大树的一瞬间，它就再也无法被主线程中的其它指令访问到了，因为主线程只会沿着这颗大树来访问。

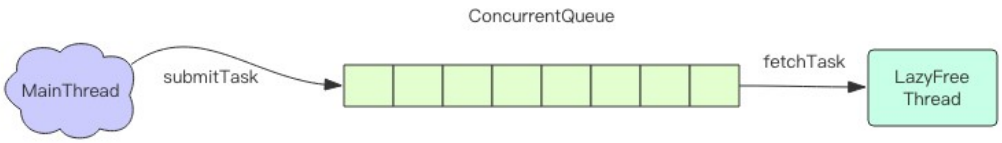
flush

Redis 提供了 `flushdb` 和 `flushall` 指令，用来清空数据库，这也是极其缓慢的操作。Redis 4.0 同样给这两个指令也带来了异步化，在指令后面增加 `async` 参数就可以将整棵大树连根拔起，扔给后台线程慢慢焚烧。

```
> flushall async
OK
```

异步队列

主线程将对象的引用从「大树」中摘除后，会将这个 key 的内存回收操作包装成一个任务，塞进异步任务队列，后台线程会从这个异步队列中取任务。任务队列被主线程和异步线程同时操作，所以必须是一个线程安全的队列。



不是所有的 `unlink` 操作都会延后处理，如果对应 key 所占用的内存很小，延后处理就没有必要了，这时候 Redis 会将对应的 key 内存立即回收，跟 `del` 指令一样。

AOF Sync也很慢

Redis需要每秒一次(可配置)同步AOF日志到磁盘，确保消息尽量不丢失，需要调用sync函数，这个操作会比较耗时，会导致主线程的效率下降，所以Redis也将这个操作移到异步线程来完成。执行AOF Sync操作的线程是一个独立的异步线程，和前面的懒惰删除线程不是一个线程，同样它也有一个属于自己的任务队列，队列里只用来存放AOF Sync任务。

更多异步删除点

Redis 回收内存除了 `del` 指令和 `flush` 之外，还会存在于在 key 的过期、LRU 淘汰、rename 指令以及从库全量同步时接受完 rdb 文件后会立即进行的 flush 操作。

Redis4.0 为这些删除点也带来了异步删除机制，打开这些点需要额外的配置选项。

1. `slave-lazy-flush` 从库接受完 rdb 文件后的 flush 操作
2. `lazyfree-lazy-eviction` 内存达到 maxmemory 时进行淘汰
3. `lazyfree-lazy-expire key` 过期删除
4. `lazyfree-lazy-server-del` rename 指令删除 destKey

扩展阅读

- Redis 懒惰处理的细节