内存管理优化：

redis hash是value内部为一个HashMap，如果该Map的成员数比较少，则会采用类似一维线性的紧凑格式来存储该Map，即省去了大量指针的内存开销，这个参数控制对应的redis.conf配置文件中下面2项：

hash-max-zipmap-entries 64 hash-max-zipmap-value 512

当value这个map内部不超过多少成员时采用线性紧凑格式存储，默认是64，即value内部有64个以下的成员就是使用线性紧凑存储，超过该值自动转成真正的hashmap。

hash-max-zipmap-value 含义是当value这个map内部的每个成员值长度不超过多少字节就会采用线性紧凑存储来节省空间。

以上2个条件任意一个超过设置值都会转换成真正的hashmap，也就不会再节省内存了，那么这个值是不是设置得越大越好呢，答案是否定得，hashmap的优势就是查找和操作的时间复杂度都是O(1)的，而放弃Hash采用一堆存储则是O(n)的时间复杂度，如果成员数量很少，则影响不大，否则会严重影响性能，所以要权衡好这个值的设置，总体上还是最根本的时间成本和空间成本上的平衡。

list-max-ziplist-value 64 list-max-ziplist-entries 512

list数据类型节点值大小小于多少字节会采用紧凑存储格式、list数据类型多少节点以下会采用指针的紧凑存储格式。

内存预分配

redis内部实现没有对内存分配方面做过多的优化，在一定程度上会存在内存碎片，不过大多数情况下这个不会成为redis的性能瓶颈，不过如果在redis内部存储的大部分数据是数值型的话，redis内部采用了一个shared integer的方式来省去分配内存的开销，即在系统启动时先分配一个从1~n那么多个数值对象放在一个池子中，如果存储的数据恰好是这个数值范围内的数据，则直接从池子中取出该对象，并且通过引用计数的方式来共享，这样在系统存储了大量数值下，也能一定程度上节省内存并且提高性能，这个参数值n的设置需要修改源代码的一行宏定义REDIS_SHARED_INTEGERS，该值默认是10000，可以根据自己的需要进行修改，修改后重新编译就可以了。

持久化机制：

定时快照方式：

该持久化方式实际是在redis内部一个定时器事件，每隔固定时间去检查当前数据发生的改变次数与时间是否满足配置的持久化触发的条件，如果满足则通过操作系统fork调用来创建出一个子进程，这个子进程默认会与父进程共享相同的地址空间，这时就可以通过子

进程来遍历整个内存来进行存储操作，而主进程则仍然可以提供服务，当有写入时由操作系统按照内存页为单位来进行copy-on-write保证父子进程之间不会互相影响。

该持久化的主要缺点是定时快照只是代表一段时间内的内存映像，所以系统重启会丢失上次快照与重启之间的所有数据。

基于语句追加方式：

aof方式实际类似mysql的基本语句的binlog方式，即每条会使redis内存数据发生改变的命令都会追加到一个log文件中，也就是说这个log文件就是redis的持久化数据。

aof的方式的主要缺点是追加log文件可能导致体积过大，当系统重启恢复数据时如果是aof方式则加载数据会非常慢，几十个G的数据可能需要几小时才能加载完，当然这个耗时并不是因为磁盘文件读取速度慢，而是由于读取的所有命令都要在内存中执行一遍。另外由于每条命令都要写log，所以使用aof方式，redis
的读写性能也会有所下降。

可以考虑将数据保存到不同的redis实例中，每个实例的内存大小在2G左右，避免将鸡蛋放到一个篮子里，既可以减少缓存失效给系统带来的影响，又可以加快数据恢复的速度，不过同时也给系统设计带来了一定的复杂性。

Redis持久化崩溃问题：

有redis线上运维经验的人会发现redis在物理内存使用比较多，但还没有超过实际物理内存总容量时就会发生不稳定甚至崩溃的问题，有人认为是基于快照方式持久化的fork系统调用造成内存占用加倍而导致的，这种观点是不准确的，因为fork调用的copy-on-write机制是基于操作系统页这个单位的，也就是只有有写入的脏页会被复制，但是一般你的系统不会在短时间内所有的页都发生了写入而导致复制，那么是什么原因导致redis崩溃的
呢？？

答案是redis的持久化使用了buffer IO造成的，所谓的buffer IO是指redis对持久化文件的写入和读取操作都会使用物理内存的page cache，而大多数数据库系统会使用Direct IO来绕过这层page Cache并自行维护一个数据的Cache，而当redis的持久化文件过大（尤其是快照文件），并对其进行读写时，磁盘文件中的数据都会被加载到物理内存中作为操作系统对该文件的一层cache，而这层cache的数据与redis内存中管理的数据实际是重复存储的，虽然内核在物理内存紧张时会做page cache的剔除工作，但内核很可能认为某块page cache更重要，而让你的进程开始swap，这时你的系统就会开始出现不稳定或者崩溃了。我们的经验是当你的物理内存使用超过内存总容量的3/5时就会开始比较危险了。

总结：

1.根据业务需要选择合适的数据类型，并为不同的应用场景设置相应的紧凑存储参数。

2.当业务场景不需要数据持久化时，关闭所有的持久化方式可以获得最佳的性能以及最大的内存使用量。

3.如果需要使用持久化，根据是否可以容忍重启丢失数据在快照方式与语句追加方式之间选择其一，不要使用虚拟内存以及diskstore放肆。

4.不要让你的redis所在机器物理内存使用超过实际内存总量的3/5。