Redis 一共有两种持久化的方式。一种是 RDB，一种是 AOF。

在 RDB 中，可以选择手动执行持久化数据命令让 Redis 进行一次数据快照，也可以根据配置文件的策略，达到策略的某些条件时来自动持久化数据。手动持久化命令有 save 和 bgsave 命令。操作 save 在 Redis 的主线程中工作，会阻塞其他请求操作，应该避免使用。操作 bgsave 调用 Fork，产生子进程，父进程继续处理请求。子进程将数据写入临时文件，并在写完后，替换原有的.rdb 文件。Fork 发生时，父子进程内存共享，为了不影响子进程做数据快照，在此期间修改的数据，会被复制一份，而不进行共享内存，RDB 持久化的数据时 Fork 发生时的数据。此情况下，如果某些情况宕机，则会丢失一段时间的数据。配置文件默认的策略有三种，（1）每隔 900 秒，变化了至少一个键值，做快照。（2）每隔 300 秒，变化了 10 个键值做快照。（3）每隔 60 秒，变化了至少 10000 个键值做快照。

在 AOF 中，其意思是 append only file。配置文件中的 appendonly 可以修改为 no 或者 yes。开启 AOF 持久化后，所执行的每一条指令，都会被记录到 appendonly.aof 文件中。指令不会立即写入到硬盘文件中，而是写入到硬盘缓存中，可以配置多长时间从硬盘缓存写入到硬盘文件中。Redis 默认采用 everysec，即每秒持久化一次。而 always 每次操作都会立即写入 aof 文件。而 no 则是不主动进行同步操作，默认 30s 一次。Redis 允许同时使用两种方式，重启 Redis 后从 aof 中恢复数据，aof 比 rdb 数据损失小。

Redis 的 RDB 方式每次进行快照会重新记录整个数据集的所有信息，RDB 在恢复数据时更快，可以最大化 Redis 性能，子进程对父进程没有性能影响。

Redis 的 AOF 方式有序的记录了 Redis 的命令操作，意外情况的数据丢失更少。不断对 aof 文件添加操作日志记录，auto-aof-rewrite-min-size 默认为 64Mb，限制了允许重写的最小 aof 文件大小。Auto-aof-rewrite-percentage 默认为 100，指的是超过上一次重写 aof 文件大小的百分之多少会再次优化，如果没有重写过，则以启动时为主。bgrewriteaof 命令是手动重写命令，会 fork 子进程，在临时文件中重建数据库状态，对原 aof 没有任何影响，当重建旧的状态后，也会把 fork 发生后的一段时间内的数据追加到临时文件，最后替换掉原来的 aof 文件，新的命令会继续向新的 aof 文件中追加。

分布式锁的解决方式:（1）基于数据库表作为乐观锁，用于分布式锁。（2）使用 Memcached 的 add 方法，用于分布式锁。（3）使用 Memcached 的 cas 方法，用于分布式锁，不常用。（4）使用 Redis 的 setnx 和 expire 方法，用于分布式锁。（5）使用 Redis 的 setnx 和 get 和 getset 方法，用于分布式锁。（6）使用 Redis 的 watch、multi、exec 命令，用于分布式锁，不常用。（7）使用 zookeeper，用于分布式锁，不常用。

乐观锁基本上是基于数据版本的记录机制实现的，为数据增加一个版本表示，在基于数据库表的版本解决方案中，一般是通过为数据库表添加一个“version”字段来实现读取数据时，将此版本号也读取，之后更新时，对其版本号加 1。在更新过程中，会对此版本号加 1 ，如果是一致的，则没有发生改变，会成功执行本次操作，如果版本号不一致，则会更新失败。

数据库中的 update 操作是原子的，存在典型的“ABA”问题，在第一次 select 和第二次 update 过程中，由于两次操作是非原子的，在过程汇总，有一个线程，先占用了资源，对其进行了更改，后将其恢复为原值，实际上执行 update 操作的时候，是不知道这个资源进行了更改的。

乐观锁缺点：（1）使得原来一次 update 操作，变成了一次 select 操作和一次 update 操作，增加了数据库的操作次数。（2）高并发请求下，对数据库连接的开销特别大。（3）乐观锁机制往往基于系统中的数据存储逻辑，可能会造成脏数据更新到数据库中。

在使用 Memcached 的 add 命令时，需要指定当前添加的这个 key 的有效时间，如果不指定有效时间，正常情况下，可以在执行完自己的业务后，通过 delete 方法将 key 删除，即释放占用的资源。但是，在占位成功后，Memcached 或者自己的业务服务器发生宕机，资源无法释放，所以对 key 设置超时时间，可以避免死锁的问题。

Redis 比 Memcached 支持更多的数据类型，Memcached 只支持 String 一种数据类型。Redis 中的 setnx 含义是 SET if Not Exists，主要的两个参数 setnx(key , value)，该方法是原子的，如果 key 不存在，则设置当前 key 成功，返回 1。如果当前 key 已经存在，则设置当前 key 失效，返回 0。但是 setnx 不能设置 key 的超时时间，只能通过 expire 来对 key 设置。具体的步骤如：（1）setnx(lockkey , 1)，如果返回 0，则说明占位失败，返回 1，则说明占位成功。（2）expire 命令对 lockkey 设置超时时间，为了避免发生死锁。（3）执行完业务代码后，通过 delete 删除 key。

由于 setnx 和 expire 方案，如果在 expire 方法之前系统宕机，则也可能会发生死锁。Redis 的 getset 命令有两个主要参数 getset(key , newValue) 。方法为原子性方法，对 key 设置 newValue 这个值，并且返回 key 原来的旧值。假设 key 原来是不存在的，则执行多次这个命令，getset(key , “value1”)返回 nil，然后 key 的值设置为 value1，getset(key , “value2”)返回 value1，此时 key 的值设置为 value2。具体的步骤如下：（1）setnx(lockkey，当前时间+过期超时时间)，如果返回 1，则获取锁成功，如果返回 0，则获取锁失败，转向 2。（2）get(lockkey)获取值 oldExpireTime，并将这个 value 值与当前的系统时间进行比较，如果小于当前系统时间，则认为这个锁已经超时，可以允许别的请求重新获取，转向 3。（3）计算 newExpireTime，为当前时间+过期超时时间，然后 getset(lockkey , newExpireTime)会返回当前 lockkey 的值 currentExpireTime。（4）判断 currentExpireTime 与 oldExpireTime 是否相等，如果相当则说明 getset 成功，否则失败。（5）在获取到锁之后，当前线程可以开始自己的业务处理，当处理完毕后，比较自己的处理时间和对于锁设置的超时时间，如果小于锁设置的超时时间，则直接 delete 释放锁，如果大于锁设置的超时时间，则不做处理。