Redis之所以快，一个最重要的原因在于它是直接将数据存储在内存，并直接从内存中读取数据的，因此一个绝对不容忽视的问题便是，一旦Redis服务器宕机，内存中的数据将会完全丢失。

好在Redis官方为我们提供了两种持久化的机制，RDB和AOF

什么是RDB

RDB是Redis的一种数据持久化到磁盘的策略，是一种以内存快照形式保存Redis数据的方式。所谓快照，就是把**某一时刻的状态**以文件的形式进行**全量备份**到磁盘，这个快照文件就称为RDB文件，其中RDB是Redis DataBase的缩写。

全量备份带来的思考

备份会不会阻塞主线程？

我们知道Redis为所有客户端处理数据时使用的是单线程，这个模型就决定了使用者需要尽量避免进行会阻塞主线程的操作。那么Redis在生成RDB文件的时候，会不会阻塞主线程呢？

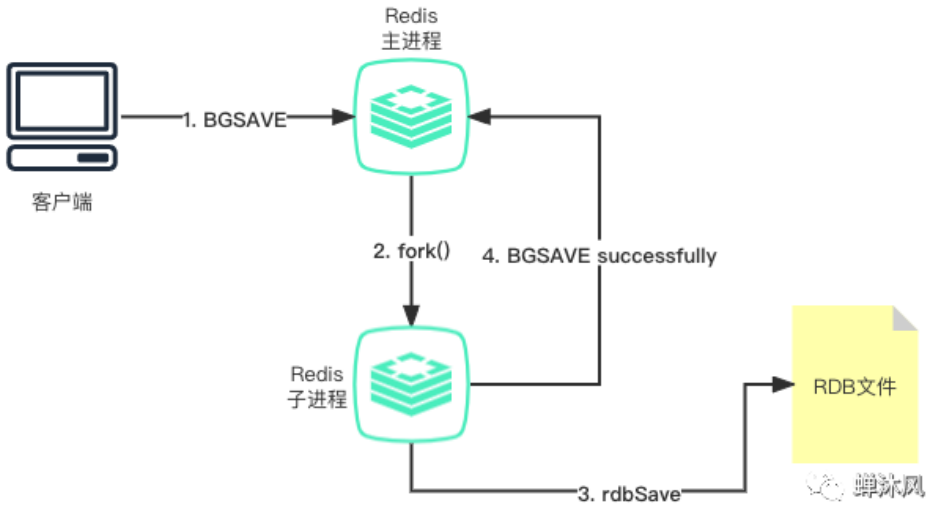
对此，Redis提供了两个命令来生成RDB，一个SAVE，另一个是BGSAVE。**SAVE命令会阻塞Redis的主线程，直到RDB文件创建完成为止，在此期间，Redis不能处理客户端的任何请求。**

```
127.0.0.1:6379> SAVE
OK
```

与SAVE直接阻塞主线程的做法不同，**BGSAVE命令会创建一个子进程，然后由子进程负责专门写入RDB，主进程（父进程）继续处理命令请求，不会被阻塞。**

注：主进程其实会阻塞在fork()过程中，通常情况下该指令执行的速度比较快，对性能影响不大

```
127.0.0.1:6379> BGSAVE
Background saving started
```



RDB文件实际是由rdb.c/rdbSave函数进行创建的，SAVE命令和BGSAVE命令会以不同的方式调用这个函数，下面是两个命令的伪代码

```
void SAVE(){
   # 创建RDB文件
   rdbSave();
}

void BGSAVE(){
   # 创建子进程
   pid = fork();

   if (pid==0){

      # 子进程创建RDB
      rdbSave();

      # 创建完成之后向父进程发送信息
      signal_parent();

   }else if (pid>0){

      # 父进程（主线程）继续处理客户端请求，并通过轮询等待子进程的返回信号
      handle_request_and_wait_signal();

   }else{
```

```
        # 处理异常
        ...
    }
}
```

对时刻备份还是对时段备份

现在我们已经知道如何对Redis某一时刻的状态进行全量备份了，需要重申的是，rdb保存的是某一时刻的全量数据，而不是某一时间段内的全量数据。

为什么要执着于某一时刻的数据，一段时间内的数据不行吗？还真就不行！因为一个时刻的数据反映了系统的该时刻的状态。例如在t1时刻，Redis保存的数据状态为

备份过程中，数据能否修改

为了实现备份某一时刻数据的这个目的，如果是我们来设计Redis，我们会怎么做呢？

一个自然的想法就是拷贝某一个时刻的Redis完整内存数据。这里自然就是子进程对主进程的内存进行全量拷贝了，然而这对于Redis服务几乎是灾难性的，考虑以下两个场景：

- **Redis中存储了大量数据，fork()时拷贝内存数据会消耗大量时间和资源，会导致主进程一段时间的不可用**

- **Redis占用了10G内存，而宿主机内存资源上限仅有16G，此时无法对Redis的数据进行持久化**

因此**备份过程中不能进行内存数据的全量拷贝**。

接下来我们需要关注的问题是，在对内存数据进行快照的过程中，数据还能被修改吗？这个问题至关重要，因为关系到Redis在快照过程中是否能正常处理写请求。

举个例子，我们在时刻t为Redis进行快照，假设内存数据量是2GB，磁盘写入带宽是0.2GB/S，不考虑其他因素的情况下，至少需要10S（2/0.2=10）才能完全备份。如果在时刻t+5S时，客户发送了一个修改目前未被写入内存的数据A的写请求，被改成了A'，如果此时A'被写入磁盘，就会破坏快照的完整性，因为我们期望获得某一时刻的全量备份。

因此，快照过程中我们不希望有数据修改的操作。但这意味着在快照期间Redis无法处理的写操作，无疑会给义务服务带来巨大影响。而且我们知道**Redis在快照期间是依然可以处理**

写请求的，接下来我们来分析一下Redis是如何解决我们刚刚提出的两个问题的。

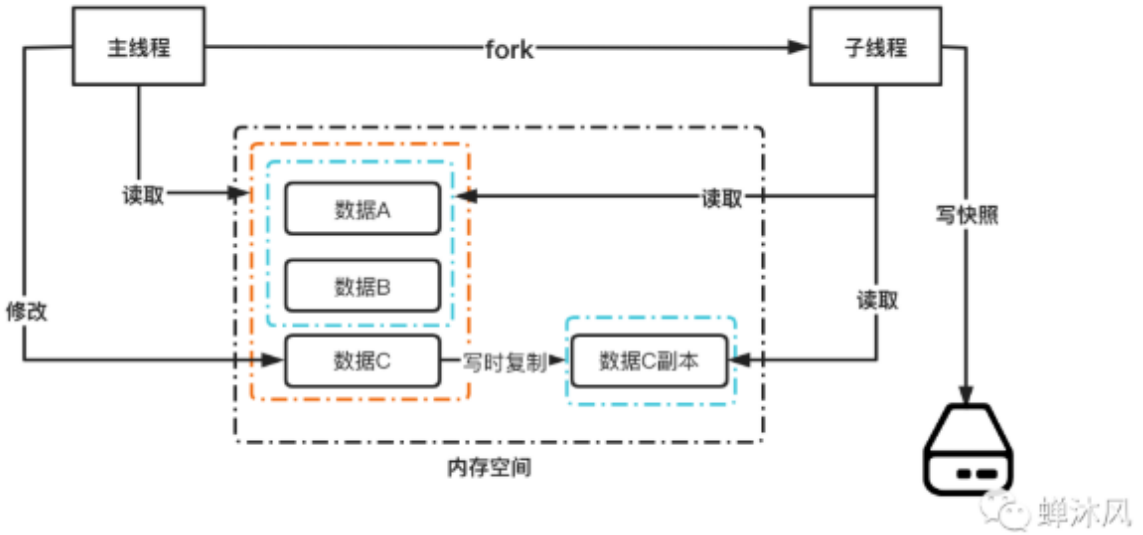
Redis写时复制（COW）

写时复制听起来非常的高端，吓退了不少技术爱好者，其原理其实非常非常简单，本质上就是“有写操作的时候复制一份”，是不是很简单？

注：写时复制不是Redis自身的特性，而是操作系统提供的技术手段。操作系统是一切技术的基础，所有技术的革新都必须建立在操作系统支持的基础上

Redis主进程fork生成的子进程可以共享主进程的所有内存数据，fork并不会带来明显的性能开销，因为不会立刻对内存进行拷贝，它会将拷贝内存的动作推迟到真正需要的时候。

想象一下，如果主进程是读取内存数据，那么和BGSAVE子进程并不冲突。如果主进程要修改Redis内存中某个数据（图中数据C），那么操作系统内核会将被修改的内存数据复制一份（复制的是修改之前的数据），未被修改的内存数据依然被父子两个进程共享，被主进程修改的内存空间归属于主进程，被复制出来的原始数据归属于子进程。如此一来，主进程就可以在快照发生的过程中肆无忌惮地接受数据写入的请求，子进程也仍然能够对某一时刻的内容做快照。



注：写时复制是建立在短时间内写请求不多的假设之下，如果写请求的量非常巨大，那么内存复制的压力自然也不会小。

间隔自动备份

除了上文介绍的手动执行的SAVE和BGSAVE方法之外，Redis还提供了配置文件的方式，可以每隔一定时间自动执行一次BGSAVE方法。

例如，我们可以在Redis配置文件中设置如下参数（如果没有主动设置save选项，则以下配置即为默认配置）

```
save 900 1
save 300 10
save 60 10000
```

那么只要满足以下3个条件之一，BGSAVE命令就会被执行

- 服务器在900秒内，对数据进行了至少1次的修改
- 服务器在300秒内，对数据进行了至少10次修改
- 服务器在60秒内，对数据进行了至少10000次修改

举个例子，以下是Redis服务器在300秒内，对数据进行了至少10次修改之后，服务器自动进行BGSAVE命令时打印的日志

```
1:M 24 Nov 2021 07:02:28.081 * 10 changes in 300 seconds. Saving...

1:M 24 Nov 2021 07:02:28.082 * Background saving started by pid 22

22:C 24 Nov 2021 07:02:28.142 * DB saved on disk

22:C 24 Nov 2021 07:02:28.143 * RDB: 0 MB of memory used by copy-on-write

1:M 24 Nov 2021 07:02:28.183 * Background saving terminated with success
```