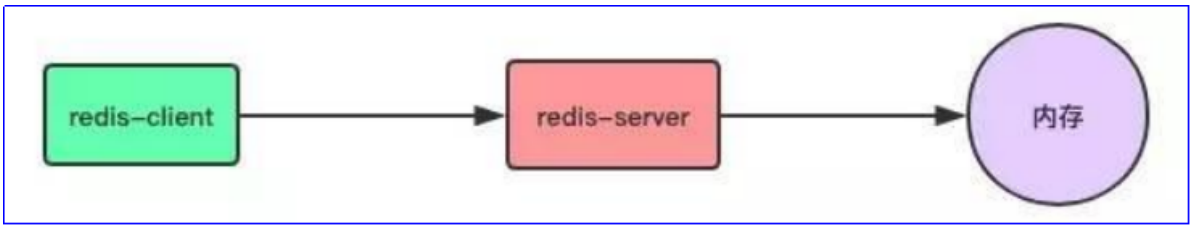
Redis持久化

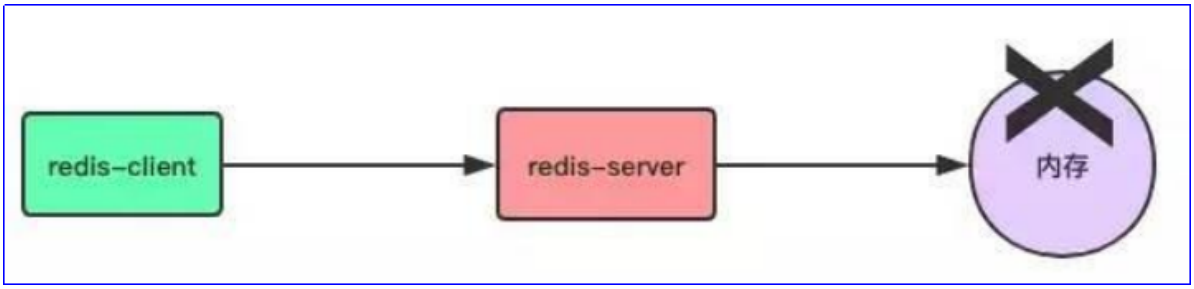
为什么要持久化

redis作为一个键值对内存数据库(NoSQL)，数据都存储在内存当中，在处理客户端请求时，所有操作都在内存当中进行，如下所示：



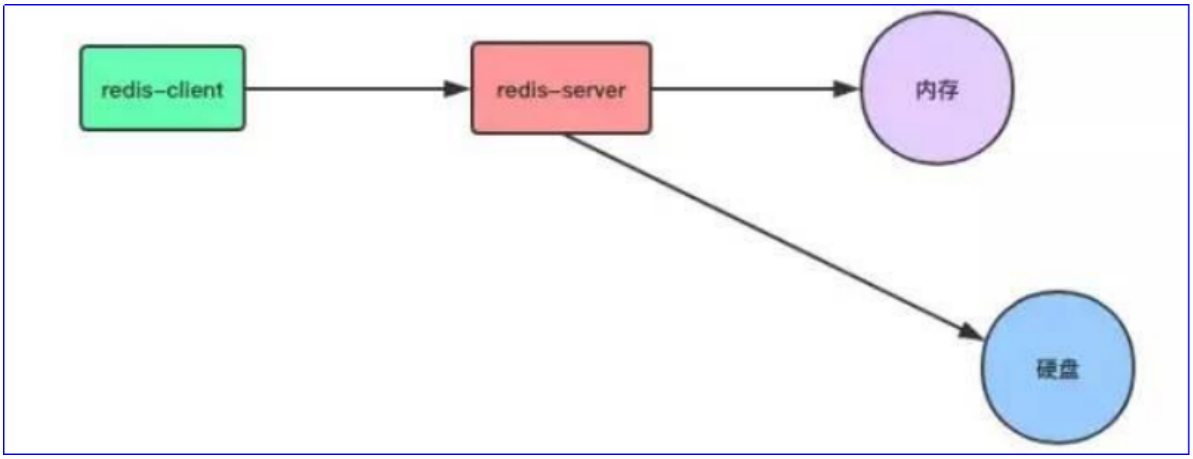
这样做有什么问题？

存储在内存当中的数据，只要服务器关机(各种原因引起的)，内存中的数据就会消失了，不仅服务器关机会造成数据消失，Redis服务器守护进程退出，内存中的数据也一样会消失。



对于只把Redis当缓存来用的项目来说，数据消失或许问题不大，重新从数据源把数据加载进来就可以了，但如果直接把用户提交的业务数据存储在Redis当中，把Redis作为数据库来使用，在其放存储重要业务数据，那么Redis的内存数据丢失所造成的影响也许是毁灭性。

为了避免内存中数据丢失，Redis提供了对持久化的支持，我们可以选择不同的方式将数据从内存中保存到硬盘当中，使数据可以持久化保存。



持续化的两种方式之一 RDB持久化

Redis提供了RDB和AOF两种不同的数据持久化方式

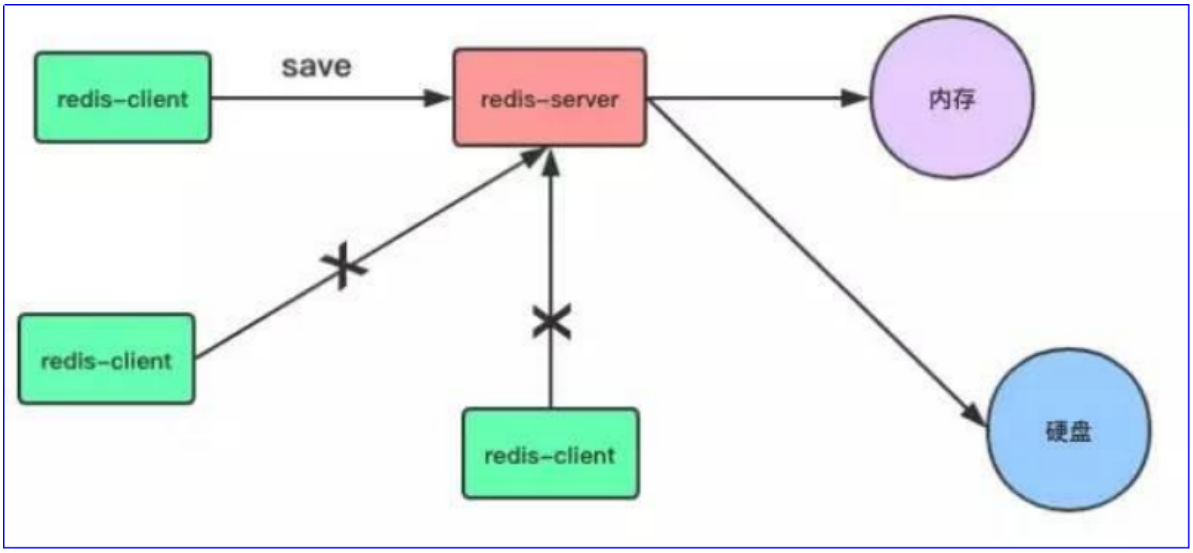
RDB

RDB是一种快照存储持久化方式，具体就是将Redis某一时刻的内存数据保存到硬盘的文件当中，默认保存的文件名为dump.rdb，而在Redis服务器启动时，会重新加载dump.rdb文件的数据到内存当中恢复数据。

开启RDB持久化方式很简单，客户端可以通过向Redis服务器发送save或bgsave命令让服务器生成rdb文件，或者通过服务器配置文件指定触发RDB条件。

save 命令

save 命令是一个同步操作

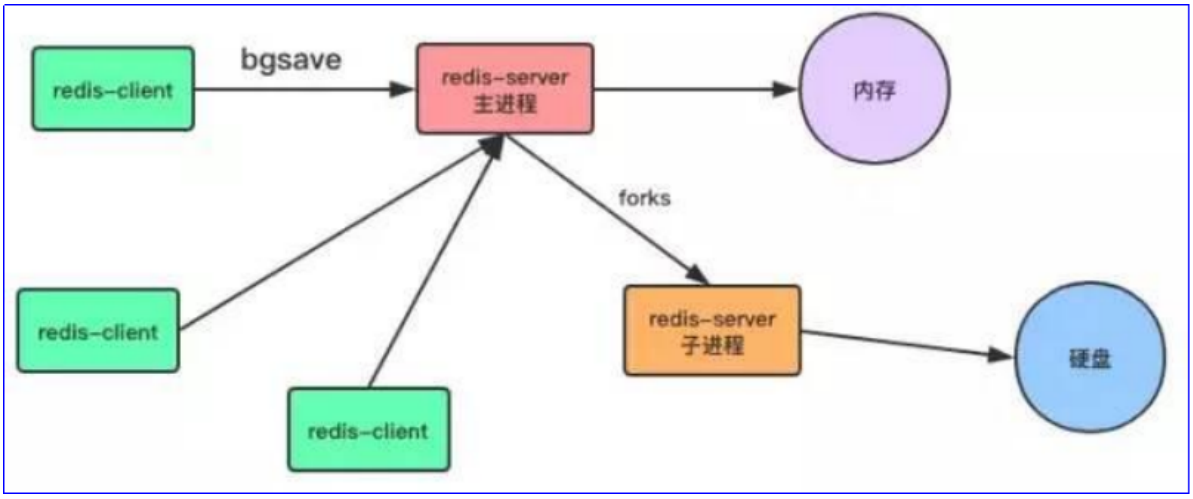


当客户端向服务器发送save命令请求进行持久化时，服务器会阻塞save命令之后的其他客户端的请求，直到数据同步完成。

如果数据量太大，同步数据会执行很久，而这期间Redis服务器也无法接收其他请求，所以，最好不要在生产环境使用save命令。

bgsave

与save命令不同，bgsave命令是一个异步操作。



当客户端发服务发出bgsave命令时，Redis服务器主进程会forks一个子进程来数据同步问题，在将数据保存到rdb文件之后，子进程会退出。

所以，与save命令相比，Redis服务器在处理bgsave采用子线程进行IO写入，而主进程仍然可以接收其他请求，但forks子进程是同步的，所以forks子进程时，一样不能接收其他请求，这意味着，如果forks一个子进程花费的时间太久(一般是很快的)，bgsave命令仍然有阻塞其他客户的请求的情况发生。

服务器配置自动触发

除了通过客户端发送命令外，还有一种方式，就是在Redis配置文件中的save指定到达触发RDB持久化的条件，比如【多少秒内至少达到多少写操作】就开启RDB数据同步。

例如我们可以在配置文件redis.conf指定如下的选项：

```
- 900s内至少达到一条写命令
save 900 1
- 300s内至少达至10条写命令
save 300 10
- 60s内至少达到10000条写命令
save 60 10000
```

之后在启动服务器时加载配置文件。

```
- 启动服务器加载配置文件
./redis-server ../redis.conf
```

这种通过服务器配置文件触发RDB的方式，与bgsave命令类似，达到触发条件时，会forks一个子进程进行数据同步，不过最好不要通过这方式来触发RDB持久化，因为设置触发的时间太短，则容易频繁写入rdb文件，影响服务器性能，时间设置太长则会造成数据丢失。

前面介绍了三种让服务器生成rdb文件的方式，无论是由主进程生成还是子进程来生成，其过程如下：

- 生成临时rdb文件，并写入数据。
- 完成数据写入，用临时文代替代正式rdb文件。
- 删除原来的rdb文件。

RDB默认生成的文件名为dump.rdb，当然，我可以通过配置文件进行更加详细配置，比如在单机下启动多个redis服务器进程时，可以通过端口号配置不同的rdb名称，如下所示：

```
- 是否压缩rdb文件
rdbcompression yes
- rdb文件的名称
dbfilename redis-6379.rdb
- rdb文件保存目录
dir ~/redis/
```

RDB的几个优点

与AOF方式相比，通过rdb文件恢复数据比较快。

rdb文件非常紧凑，适合于数据备份。

通过RDB进行数据备份，由于使用子进程生成，所以对Redis服务器性能影响较小。

RDB的几个缺点

如果服务器宕机的话，采用RDB的方式会造成某个时段内数据的丢失，比如我们设置10分钟同步一次或5分钟达到1000次写入就同步一次，那么如果还没达到触发条件服务器就死机了，那么这个时间段的数据会丢失。

使用save命令会造成服务器阻塞，直接数据同步完成才能接收后续请求。

使用bgsave命令在forks子进程时，如果数据量太大，forks的过程也会发生阻塞，另外，forks子进程会耗费内存。

持久化的两种方式之一AOF持久化

Redis默认不开启AOF持久化方式，我们可以在配置文件中开启并进行更加详细的配置，如下面的redis.conf文件：

- 开启aof机制

```
appendonly yes
```

- aof文件名

```
appendfilename "appendonly.aof"
```

- 写入策略,always表示每个写操作都保存到aof文件中,也可以是everysec或no

```
appendfsync always
```

三种写入策略

在上面的配置文件中，我们可以通过appendfsync选项指定写入策略,有三个选项

```
appendfsync always
# appendfsync everysec
# appendfsync no
```

- always

客户端的每一个写操作都保存到aof文件当，这种策略很安全，但是每个写操作都会触发磁盘的IO操作，所以也很慢。

- everysec

appendfsync的默认写入策略，每秒写入一次aof文件，因此，最多可能会丢失1s的数据。

- no

Redis服务器不负责写入aof，而是交由操作系统来处理什么时候写入aof文件。更快，但也是最不安全的选择，不推荐使用。

默认不重写 AOF 文件

```
no-appendfsync-on-rewrite no
```

AOF 文件重写

AOF 将客户端的每一个写操作都追加到 AOF 文件末尾，比如对一个 key 多次执行 incr 命令，这时候，AOF 保存每一次命令到 AOF 文件中，AOF 文件会变得非常大。

```
incr num 1
incr num 2
incr num 3
incr num 4
incr num 5
incr num 6
...
incr num 100000
```

AOF 文件太大，加载 AOF 文件恢复数据时，就会非常慢，为了解决这个问题，Redis 支持 AOF 重写，通过重写 AOF，可以生成一个恢复当前数据的最少命令集，比如上面的例子中，那么多条命令，可以重写为：

```
set num 100000
```

AOF 文件是一个二进制文件，并不是向上面的例子一样，直接保存每个命令，而使用 Redis 自己的格式，上面知识方便演示。

两种重写方式

通过在 redis.conf 配置文件中的选项 no-appendfsync-on-rewrite 可以设置是否开启重写，这种方式会在每次 fsync 时都重写，影响服务器性以，因此默认值为 no，不推荐使用。

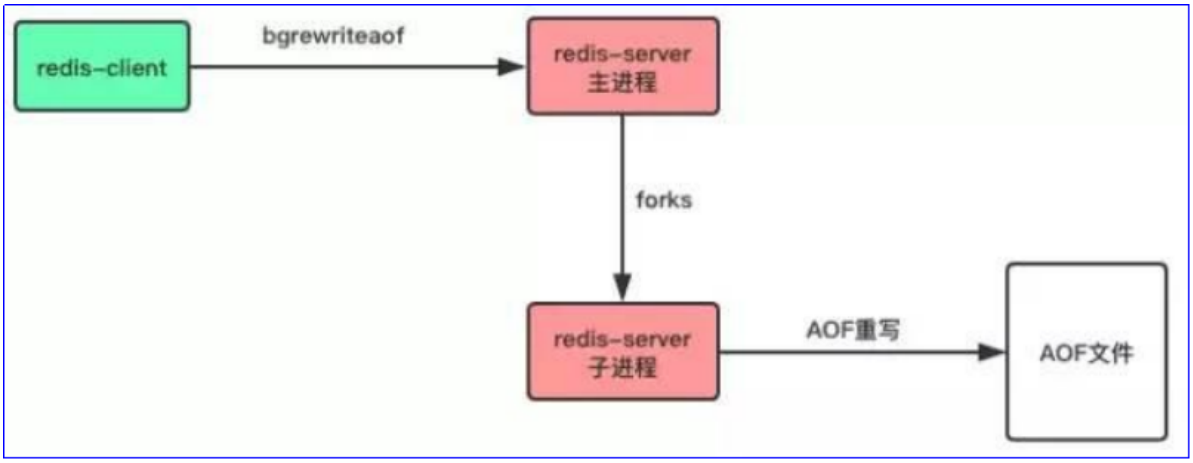
- 默认不重写 AOF 文件 no-appendfsync-on-rewrite no

客户端向服务器发送 bgrewriteaof 命令，也可以让服务器进行 AOF 重写。

- 让服务器异步重写追加aof文件命令

bgrewriteaof

AOF 重写方式也是异步操作，即如果要写入 AOF 文件，则 Redis 主进程会 forks 一个子进程来处理，如下所示：



重写aof文件的好处

- 压缩aof文件，减少磁盘占用量。
- 将aof的命令压缩为最小命令集，加快了数据恢复的速度。

```
# 保存目录dir  ~/redis/
```

AOF的优点

AOF只是追加日志文件，因此对服务器性能影响较小，速度比RDB要快，消耗的内存较少。

AOF的缺点

AOF方式生成的日志文件太大，即使通过AFO重写，文件体积仍然很大。

恢复数据的速度比RDB慢。

选择 RDB 还是选择 AOF 呢?

通过上面的介绍，我们了解了RDB与AOF各自的优点与缺点，到底要如何选择呢？

通过下面的表示，我们可以从几个方面对比一下RDB与AOF,在应用时，要根本自己的实际需求，选择RDB或者AOF，其实，如果想要数据足够安全，可以两种方式都开启，但两种持久化方式同时进行IO操作，会严重影响服务器性能，因此有时候不得不做出选择。

方式	RDB	AOF
启动优化级	低	高
体积	小	大
恢复速度	快	慢
数据安全性	会丢数据	由策略决定
轻重	重	轻

当RDB与AOF两种方式都开启时，Redis会优先使用AOF日志来恢复数据，因为AOF保存的文件比RDB文件更完整

如果只是单纯把Redis作为缓存服务器，那么可以完全不用考虑持久化，但是，在如今的大多数服务器架构中，Redis不单单只是扮演一个缓存服务器的角色，还可以作为数据库，保存我们的业务数据，此时，我们则需要好好了解有关Redis持久化策略的区别与选择。