AOF

RDB的存储弊端

- 存储数据量大，效率低，基于快照思想，每次读写都是全部数据，当数据量大时，效率非常低
- 大数据量下的IO性能较低
- 基于fork创建子进程，内存产生额外消耗
- 宕机带来的数据丢失风险，会丢失上次执行bgsave与宕机时间之间的数据

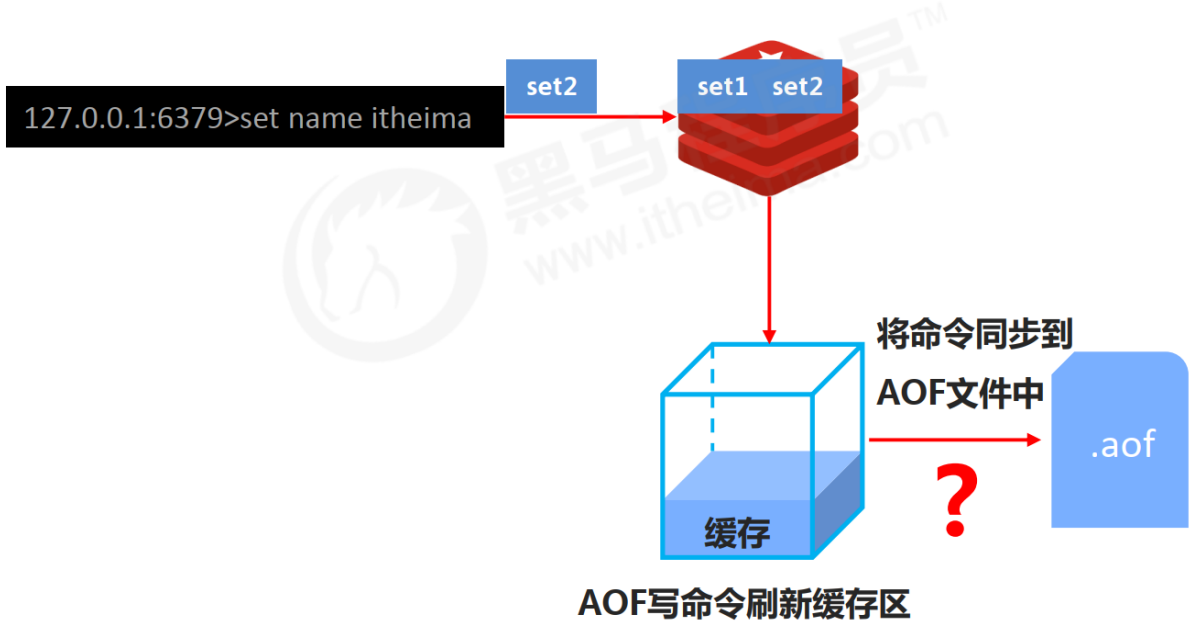
解决思路

- 不全写数据，仅记录部分数据
- 降低区分数据是否改变的难度，改记录数据为记录操作过程
- 对所有操作均进行记录，排除丢失数据的风险

AOF概念

- AOF（append only file）持久化：以独立日志的方式记录每次写命令，重启时在重新执行AOF文件中的命令达到恢复数据的目的，与RDB相比可以简单描述为改记录数据为记录数据产生的过程
- AOF的主要作用是解决了数据持久化的实时性，目前已经是Redis主流的持久化方式

AOF写数据



AOF写数据的三种策略

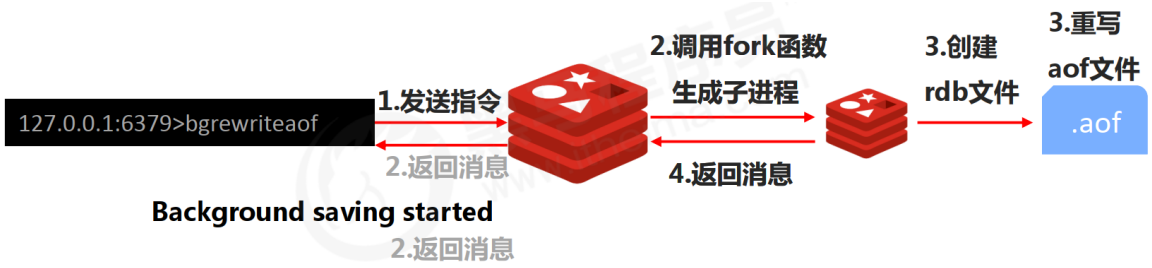
- always（每次）：每次写入操作均同步到AOF文件中，数据零误差，性能较低，不建议使用
- everysec（每秒）：每秒将缓冲区中的指令同步到AOF文件中，数据准确性较高，性能较高，建议使用，也是默认设置，就算系统宕机也最多丢失一秒的数据
- no（系统控制）：由操作系统控制每次同步到AOF文件的周期，整体过程不可控

AOF相关配置

- appendonly yes|no：是否开启AOF持久化功能
- appendfsync always|everysec|no：写数据策略
- appendfilename filename：AOF持久化文件名
- dir：AOF持久化文件保存路径

AOF重写

随着命令不断写入AOF，文件会越来越大，为了解决这个问题Redis引入了AOF重写机制压缩文件体积，AOF文件重写是将REdis进程内的数据转化为写命令同步到新AOF文件的过程，简单来说就是将对同一个数据的若干条命令执行结果转化成最终结果数据对应的指令进行记录



AOF重写作用

- 降低磁盘占用率，提高磁盘利用率
- 提高持久化效率，降低持久化写时间，提高IO性能
- 降低数据恢复用时，提高数据恢复效率

AOF重写规则

- 进程内已超时的数据不再写入文件
- 忽略无效指令，重写时使用进程内数据直接生成，这样新的AOF文件只保留最终数据的写入命令
- 对同一条数据的多条命令合并为同一条命令

AOF重写方式

- 手动重写：bgrewriteaof
- 自动重写：（通过info指令获取当前redis的一些参数信息）

- 自动重写触发条件设置

```
auto-aof-rewrite-min-size size
auto-aof-rewrite-percentage percent
```

- 自动重写触发比对参数（ 运行指令info Persistence获取具体信息 ）

```
aof_current_size
aof_base_size
```

- 自动重写触发条件

$$\frac{aof\_current\_size - aof\_base\_size}{aof\_base\_size} >= auto\text{-}aof\text{-}rewrite\text{-}percentage$$

$$aof\_current\_size > auto\text{-}aof\text{-}rewrite\text{-}min\text{-}size$$

- auto-aof-rewrite-min-size size

  会与参数aof_current_size中的数据对比，如果aof_current_size的大小等于auto-aof-rewrite-min-size的值，则重写

- auto-aof0rewrite-percentage percenteage

  与参数aof_base_size做百分比计算，达到auto-aof0rewrite-percentage设定的百分比就重写

AOF重写流程