Apusic 金蝶天燕

生成时间：2026-05-07

缓存核心用户手册

金蝶Apusic分布式缓存 V2.0.4

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 前言

本文档为金蝶Apusic分布式缓存（AMDC）V2.0.4缓存核心产品的用户手册，详细介绍了 AMDC 缓存核心的功能使用、配置方法及管理操作等内容。

1.1 适用对象

本文档适用于AMDC产品运维工程师、IT系统运维工程师、开发工程师等人员。

1.2 相关文档

了解更多AMDC V2.0.4产品相关的信息，请参阅以下AMDC V2.0.4产品手册文档集：

序号	手册文档	说明
1	金蝶Apusic分布式缓存 V2.0.4 快速使用手册	简单介绍了如何快速上手使用AMDC 。
2	金蝶Apusic分布式缓存 V2.0.4 安装手册	详细介绍如何在各操作系统上安装AMDC，以及AMDC服务启停操作，产品的注册过程。
3	金蝶Apusic分布式缓存 V2.0.4 缓存核心用户手册	详细介绍 AMDC 相关功能的使用、配置、管理及配套工具的使用方法。
4	金蝶Apusic分布式缓存 V2.0.4 管控台用户手册	详细介绍AMDC管控台相关功能的使用和操作说明。
5	金蝶Apusic分布式缓存 V2.0.4 开发手册	详细介绍基于各开发语言进行AMDC客户端应用开发的说明。
6	金蝶Apusic分布式缓存 V2.0.4 迁移手册	详细介绍AMDC历史版本迁移升级到V2.0.4版本的说明，以及Redis迁移到AMDC的说明。
7	金蝶Apusic分布式缓存 V2.0.4 运维手册	详细介绍AMDC的监控、运维、安全加固等运维说明。
8	金蝶Apusic分布式缓存 V2.0.4 性能优化手册	详细介绍AMDC性能调优的说明。

1.3 技术支持

AMDC产品提供全面的技术支持服务，您可以通过以下方式获得技术支持：

- 网址：www.apusic.com
- 电话：400-855-5800
- 邮箱：support@apusic.com
- 金蝶云社区：https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN

您在取得技术支持时，请提供如下信息：

1. 您的姓名
2. 公司信息与联系方式
3. 操作系统及其版本
4. 产品版本号
5. 出现异常及错误的日志、截图等详细信息

2 简介

金蝶Apusic分布式缓存软件（Apusic In-Memory Data Cache，简称`AMDC`），一款完全泛场景适用、高吞吐量、数据安全的分布式缓存软件，为大规模、高并发、高可用的关键应用提供安全可靠的缓存支撑能力。产品兼容Redis协议与持久化数据文件，可简单快捷平稳替换Redis。

3 相关概念

名词	含义	使用说明
单机	存储数据，负责数据读写，负责数据同步	AMDC的默认模式，单机模式，单机模式下，主节点为单机模式
主从	从主节点复制数据，负责数据读写，不参与数据同步	主节点数据的实时副本。它主动连接到指定的主节点，并接收主节点发送的数据流，从而保持与主节点的数据同步。默认情况下，从节点是只读的，所有写操作必须发送到主节点。
哨兵	监控主从节点，负责主节点故障转移，负责主从节点同步	哨兵模式，当主节点挂掉时，自动切换为从节点，当从节点挂掉时，自动切换为主节点。
集群	由多台相互独立的计算机组成的计算机服务系统	在本文中，主从/哨兵/集群三种模式都可以算是集群，cluster集群模式特指多主节点数据分片存储的模式。

4 功能清单

功能	功能说明	补充说明
多数据类型缓存	提供string、list、hash、set、sortedset、geo、hyperloglog、stream的数据缓存类型。	支持Redis协议兼容的数据结构，可直接替代Redis使用，无需修改应用代码
发布订阅	实现了发布订阅功能，增强系统功能性	支持频道订阅、模式订阅，可用于消息队列、事件通知等场景
ACL权限管控	为服务提供了安全的访问机制，支持细粒度的访问权限控制。	基于用户角色的权限管理，支持命令级别、键空间级别的访问控制
内存数据淘汰策略	提供多种数据淘汰策略，满足多种数据淘汰要求，提高内存利用率。	包括LRU、LFU、TTL等多种淘汰算法，可根据业务特点灵活选择
持久化	为缓存核心提高可用性，防止在宕机时丢失数据。	支持RDB快照和AOF日志两种持久化方式，保障数据安全
Lua脚本支持	持使用lua脚本操作缓存核心。	支持复杂逻辑的原子性操作，提升处理效率
多线程模式	支持多线程并发请求处理，提升系统吞吐量。	采用I/O多路复用技术，充分利用多核CPU性能
主从模式	支持主从备份。	实现数据冗余，支持读写分离，提高系统可用性
哨兵模式	为主从模式提供节点监控、自动故障转移、故障通知、配置传播功能。	实现高可用架构，自动检测故障并切换服务
集群模式	支持弹性伸缩（内存的扩/缩容），并且同时具备了故障转移功能。	数据分片存储，支持在线扩容缩容，满足大数据量场景需求
监控运维	性能监控	提供详细的性能指标监控，包括QPS、响应时间、内存使用率等
	慢查询日志	记录执行时间超过阈值的命令，便于性能调优
	统计信息	通过INFO命令获取服务器运行状态和统计数据
高可用性	故障自动恢复	自动检测节点健康状况，实现故障自动切换

	数据一致性	保证主从节点间的数据一致性
	服务发现	支持自动发现集群节点，简化配置管理
安全特性	访问控制	支持密码认证、SSL/TLS加密传输
	审计日志	记录用户操作日志，满足合规性要求
扩展功能	事务支持	支持MULTI/EXEC等事务命令
	管道操作	支持批量命令执行，减少网络往返时间
	键过期管理	支持TTL、EXPIRE等键生命周期管理功能
开发支持	客户端兼容	完全兼容Redis客户端库（Jedis、Lettuce等）
	协议兼容	支持RESP协议，可无缝替换Redis
	工具链支持	提供CLI工具、基准测试工具等

5 产品安装

AMDC缓存核心安装请参考《金蝶Apusic分布式缓存 V2.0.4 安装手册》

5.1 Redis配置兼容

从AMDC v2.0.2开始，兼容Redis的配置文件，AMDC会自动将Redis.conf中的内容转换为AMDC的配置内容。

使用方式：`./amdc-server redis.conf`（指定为redis的配置文件）

6 使用介绍

6.1 缓存核心使用指南

分布式缓存是AMDC最核心的能力，是整个产品的中心，其他功能都是建立在数据缓存业务之上。AMDC把数据直接存到内存中，并利用多线程读写分离，实现高效存储，满足不同类型的数据存储，快捷开发，减少类型转换;支持多种数据淘汰策略，合理利用内存空间。

6.1.1 操作命令

命令	说明
ACL LOAD	从配置的ACL文件中重新加载ACL
ACL SAVE	在已配置的ACL文件中保存当前的ACL规则
ACL LIST	列出ACL配置文件格式的当前ACL规则
ACL USERS	列出所有已配置的ACL规则的用户名
ACL GETUSER username	获取特定ACL用户的规则
ACL SETUSER username [rule [rule ...]]	修改或创建特定ACL用户的规则
ACL DELUSER username [username ...]	删除指定的ACL用户和相关规则
ACL CAT [categoryname]	列出类别内的ACL类别或命令
ACL GENPASS [bits]	为ACL用户生成伪谐像安全密码
ACL WHOAMI	返回与当前连接关联的用户的名称
ACL LOG [count or RESET]	列出最新的ACL安全事件
ACL HELP	显示有关不同的子命令的有用文本
APPEND key value	将值附加到键
ASKING	在-ask重定向后由群集客户端发送
AUTH [username] password	对服务器进行身份验证
BGREWRITEAOF	异步重写仅附加文件
BGSAVE [SCHEDULE]	异步将数据集保存到磁盘
BITCOUNT key [start end [BYTE|BIT]]	计数字符串中的设置位

BITFIELD key [GET encoding offset] [SET encoding offset value] [INCRBY encoding offset increment] [OVERFLOW WRAP|SAT|FAIL]	在字符串上执行任意位字段整数操作
BITFIELD_RO key GET encoding offset	在字符串上执行任意位字段整数操作，禁区的只读变体
BITOP operation destkey key [key ...]	在字符串之间执行按位操作
BITPOS key bit [start [end [BYTE|BIT]]]	在字符串中找到第一个位设置或清除
BLPOP key [key ...] timeout	阻塞式删除并返回第一个元素
BRPOP key [key ...] timeout	阻塞式删除并返回最后一个元素
BRPOPLPUSH srouce target timeout	从列表中取出最后一个元素，并插入到另外一个列表的头部； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。
BLMOVE source destination FROM-TOP|FROM-BOTTOM TO-TOP| TO-BOTTOM timeout	阻塞式返回并删除存储在源列表的第一个或最后一个元素（头尾取决于wherefrom参数），并将该元素压入到存储目标列表的第一个或最后一个元素（头尾取决于whereto参数）
LMPOP numkeys key [key ...] LEFT|RIGHT [COUNT count]	从提供的键名列表中弹出第一个非空列表键中的一个或多个元素
BLMPOP timeout numkeys key [key ...] LEFT|RIGHT [COUNT count]	阻塞式从提供的键名列表中弹出第一个非空列表键中的一个或多个元素；或阻塞连接直到另一个客户端推送到它或直到超时
BZPOPMIN key [key ...] timeout	阻塞式删除并将成员从一个或多个有序集中返回最低分数，或阻塞连接直到另一个客户端推送到它或直到超时
BZPOPMAX key [key ...] timeout	阻塞式删除并将成员从一个或多个有序集中返回最高分数，或阻塞连接直到另一个客户端推送到它或直到超时
CLIENT CACHING YES|NO	指示服务器在下一个请求中跟踪或不键
CLIENT ID	返回当前连接的客户端ID
CLIENT INFO	返回有关当前客户端连接的信息

CLIENT KILL [ip:port] [ID client-id] [TYPE normal|master|slave|pubsub] [USER username] [ADDR ip:port] [LADDR ip:port] [SKIPME yes/no]	杀死客户的连接
CLIENT LIST [TYPE normal|master|replica|pubsub] [ID client-id [client-id ...]]	获取客户端连接列表
CLIENT GETNAME	获取当前的连接名称
CLIENT GETREDIR	获取跟踪通知重定向客户端ID（如果有）
CLIENT UNPAUSE	恢复暂停的客户的处理
CLIENT PAUSE timeout [WRITE|ALL]	停止客户端的处理命令一段时间
CLIENT REPLY ON|OFF|SKIP	指示服务器是否回复命令
CLIENT SETNAME connection-name	设置当前连接名称
CLIENT TRACKING ON|OFF [REDIRECT client-id] [PREFIX prefix [PREFIX prefix ...]] [BCAST] [OPTIN] [OPTOUT] [NOLOOP]	启用或禁用服务器辅助客户端缓存支持
CLIENT TRACKINGINFO	返回关于当前连接的服务器辅助客户端缓存的信息
CLIENT UNBLOCK client-id [TIMEOUT|ERROR]	取消阻止从不同连接中阻止阻止命令的客户端
COMMAND	获取 Redis 命令详细信息数组
COMMAND COUNT	获取Redis命令的总数
COMMAND GETKEYS	给定完整的 Redis 命令提取键
COMMAND INFO command-name [command-name ...]	获取特定Redis命令详细信息的数组
CONFIG GET parameter [parameter ...]	获取配置参数的值
CONFIG REWRITE	用内存配置重写配置文件
CONFIG SET parameter value [parameter value ...]	将配置参数设置为给定的值
CONFIG RESETSTAT	重置 INFO 返回的统计信息
COPY source destination [DB destination-db] [REPLACE]	复制一个键
DBSIZE	返回所选数据库中的键数
DEBUG OBJECT key	获取有关键的调试信息

DEBUG SEGFAULT	使服务器崩溃
DECR key	一个键的整数值减一
DECRBY key decrement	按给定的数字减少一个键的整数值
DEL key [key ...]	删除一个键
DISCARD	放弃在多次执行后发出的所有命令
DUMP key	返回存储在指定键中的值的序列化版本
ECHO message	回显给定的字符串
EVAL script numkeys [key [key ...]] [arg [arg ...]]	执行一个Lua脚本服务器端
EVALSHA sha1 numkeys [key [key ...]] [arg [arg ...]]	执行 Lua 脚本服务器端
EXEC	执行 MULTI
EXISTS key [key ...]	判断key是否存在
EXPIRE key seconds [NX|XX|GT|LT]	以秒为单位设置键的生存时间
EXPIREAT key timestamp [NX|XX|GT|LT]	将键的到期时间设置为 UNIX 时间戳
EXPIRETIME key	获取键的到期 Unix 时间戳
FAILOVER [TO host port [FORCE]] [ABORT] [TIMEOUT milliseconds]	在此服务器与其副本之一之间启动协调故障转移
FLUSHALL [ASYNC|SYNC]	从所有数据库中删除所有键
FLUSHDB [ASYNC|SYNC]	从当前数据库中删除所有键
GEOADD key [NX|XX] [CH] longitude latitude member [longitude latitude member ...]	在使用有序集表示的地理空间索引中添加一个或多个地理空间项目
GEOHASH key member [member ...]	将地理空间索引的成员作为标准 geohash 字符串返回
GEOPOS key member [member ...]	返回地理空间索引成员的经度和纬度
GEODIST key member1 member2 [m|km|ft|mi]	返回地理空间索引的两个成员之间的距离
GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count [ANY]] [ASC|DESC] [STORE key] [STOREDIST key]	查询表示地理空间索引的有序集，以获取与点的给定最大距离匹配的成员

GEORADIUSBYMEMBER key member radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [COUNT count [ANY]] [ASC|DESC] [STORE key] [STOREDIST key]	查询表示地理空间索引的有序集以获取与成员的给定最大距离匹配的成员
GEOSEARCH key [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi] [ASC|DESC] [COUNT count [ANY]] [WITHCOORD] [WITHDIST] [WITHHASH]	查询表示地理空间索引的有序集以获取框或圆区域内的成员
GEOSEARCHSTORE destination source [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi] [ASC|DESC] [COUNT count [ANY]] [STOREDIST]	查询表示地理空间索引的有序集以获取框或圆区域内的成员，并将结果存储在另一个键中
GET key	获取一个键的值
GETBIT key offset	返回存储在 key
GETDEL key	的字符串值中偏移量处的位值，获取某个key的值并删除key
GETRANGE key start end	获取存储在键中的字符串的子字符串
GETSET key value	设置一个键的字符串值并返回它的旧值
HDEL key field [field ...]	删除一个或多个哈希字段
HELLO [protover [AUTH username password] [SETNAME clientname]]	与 Redis 握手
HEXISTS key field	判断一个hash字段是否存在
HGET key field	获取哈希字段的值
HGETALL key	获取哈希中的所有字段和值
HINCRBY key field increment	将散列字段的整数值增加给定的数字
HINCRBYFLOAT key field increment	将哈希字段的浮点值增加给定的数量
HKEYS key	获取散列中的所有字段
HLEN key	获取哈希中的字段数
HMGET key field [field ...]	获取所有给定哈希字段的值
HMSET key field value [field value ...]	将多个哈希字段设置为多个值
HSET key field value [field value ...]	设置一个哈希字段的字符串值

HSETNX key field value	设置一个哈希字段的值，仅当该字段不存在时
HRANDFIELD key [count [WITHVALUES]]	从散列中获取一个或多个随机字段
HSTRLEN key field	获取哈希字段值的长度
HVALS key	获取散列中的所有值
INCR key	键的整数值加一
INCRBY key increment	将键的整数值增加给定的数量
INCRBYFLOAT key increment	将键的浮点值增加给定的数量
INFO [section]	获取有关服务器的信息和统计信息
LOLWUT [VERSION version]	展示一些计算机艺术和 Redis 版本
KEYS pattern	找到所有匹配给定模式的键
LASTSAVE	获取上次成功保存到磁盘的 UNIX 时间戳
LINDEX key index	通过索引从列表中获取元素
LINSERT key BEFORE|AFTER pivot element	在列表中的另一个元素之前或之后插入一个元素
LLEN key	获取列表的长度
LPOP key [count]	删除并获取列表中的第一个元素
LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len]	返回列表中匹配元素的索引
LPUSH key element [element ...]	将一个或多个元素添加到列表中
LPUSHX key element [element ...]	将元素添加到列表中，仅当列表存在时
LRANGE key start stop	从列表中获取一系列元素
LREM key count element	从列表中删除元素
LSET key index element	通过索引设置列表中元素的值
LTRIM key start stop	将列表修剪到指定范围
MEMORY DOCTOR	输出内存问题报告
MEMORY HELP	显示有关不同子命令的有用文本

MEMORY MALLOC-STATS	显示分配器内部统计信息
MEMORY PURGE	要求分配器释放内存
MEMORY STATS	显示内存使用详情
MEMORY USAGE key [SAMPLES count]	估计一个key的内存占用
MGET key [key ...]	获取所有给定键的值
MIGRATE host port key| destination-db timeout [COPY] [REPLACE] [AUTH password] [AUTH2 username password] [KEYS key [key ...]]	以原子方式将键从 Redis 实例传输到另一个实例
MONITOR	实时监听服务器收到的所有请求
MOVE key db	移动一个键到另一个数据库
MSET key value [key value ...]	将多个键设置为多个值
MSETNX key value [key value ...]	仅当不存在任何键时才将多个键设置为多个值
MULTI	标记一个事务块的开始
OBJECT ENCODING key	检查 Redis 对象的内部编码
OBJECT FREQ key	获取Redis对象的对数访问频率计数器
OBJECT IDLETIME key	获取自上次访问 Redis 对象以来的时间
OBJECT REFCOUNT key	获取键值的引用次数
OBJECT HELP	显示有关不同子命令的有用文本
PERSIST key	从键中删除过期时间
PEXPIRE key milliseconds [NX|XX|GT|LT]	以毫秒为单位设置键的生存时间
PEXPIREAT key milliseconds-timestamp [NX|XX|GT|LT]	将键的到期时间设置为以毫秒为单位指定的 UNIX 时间戳
PEXPIRETIME key	以毫秒为单位获取键的到期 Unix 时间戳
PFADD key [element [element ...]]	将指定的元素添加到指定的 HyperLogLog.
PFCOUNT key [key ...]	返回 HyperLogLog 在 key(s) 处观察到的集合的近似基数
PFMERGE destkey sourcekey [sourcekey ...]	将 N 个不同的 HyperLogLog 合并为一个

PING [message]	Ping 服务器
PSETEX key milliseconds value	以毫秒为单位设置键值和过期时间
PSUBSCRIBE pattern [pattern ...]	侦听发布到与给定模式匹配的频道的消息
PUBSUB CHANNELS [pattern]	列出活动频道
PUBSUB NUMPAT	获取独特模式模式订阅的计数
PUBSUB NUMSUB [channel [channel ...]]	获取频道的订阅者数量
PUBSUB HELP	显示有用的文字 ab输出不同的子命令
PTTL key	以毫秒为单位获取键的生存时间
PUBLISH channel message	向频道发布消息
PUNSUBSCRIBE [pattern [pattern ...]]	停止收听发布到与给定模式匹配的频道的消息
QUIT	关闭连接
RANDOMKEY	从键空间返回一个随机键
READONLY	启用对集群副本节点连接的读取查询
READWRITE	禁用对集群副本节点连接的读取查询
RENAME key newkey	重命名一个键
RENAMENX key newkey	重命名键，仅当新键不存在时
RESET	重置连接
RESTORE key ttl serialized-value [REPLACE] [ABSTTL] [IDLETIME seconds] [FREQ frequency]	使用提供的序列化值创建一个键，之前使用 DUMP 获得
ROLE	返回实例在复制上下文中的角色
RPOP key [count]	删除并获取列表中的最后一个元素
RPOPLPUSH source destination	删除列表中的最后一个元素，将其添加到另一个列表中并返回它
LMOVE source destination LEFT|RIGHT LEFT|RIGHT	从列表中弹出一个元素，将其推送到另一个列表并返回它
RPUSH key element [element ...]	将一个或多个元素附加到列表中

RPUSHX key element [element ...]	仅当列表存在时才将元素附加到列表中
SADD key member [member ...]	将一个或多个成员添加到集合中
SAVE	将数据集同步保存到磁盘
SCARD key	获取集合中的成员数量
SCRIPT DEBUG YES|SYNC|NO	为执行的脚本设置调试模式
SCRIPT EXISTS sha1 [sha1 ...]	检查脚本缓存中是否存在脚本
SCRIPT FLUSH [ASYNC|SYNC]	从脚本缓存中删除所有脚本
SCRIPT KILL	终止当前正在执行的脚本
SCRIPT LOAD script	将指定的 Lua 脚本加载到脚本缓存中
SDIFF key [key ...]	减去多组
SDIFFSTORE destination key [key ...]	减去多个集合并将结果集合存储在一个键中
SELECT index	更改当前连接选择的数据库
SET key value [EX seconds|PX milliseconds|EXAT timestamp|PXAT milliseconds-timestamp|KEEPTTL] [NX|XX] [GET]	设置一个键的字符串值
SETBIT key offset value	设置或清除存储在 key
SETEX key seconds value	设置一个键的值和过期时间
SETNX key value	设置键的值，仅当键不存在时
SETRANGE key offset value	从指定的偏移量开始覆盖键处的部分字符串
SHUTDOWN [NOSAVE|SAVE]	将数据集同步保存到磁盘，然后关闭服务器
SINTER key [key ...]	返回由所有给定集合的交集产生的集合的成员
SINTERCARD numkeys key [key ...] [LIMIT limit]	将多个集合相交并返回结果的基数
SINTERSTORE destination key [key ...]	将多个集合相交并将结果集存储在一个键中
SISMEMBER key member [member ...]	返回每个成员是否为存储在key处的集合的成员
REPLICAOF host port	使服务器成为另一个实例的副本，或将其提升为主服务器

SLOWLOG GET [count]	获取慢日志的条目
SLOWLOG LEN	获取慢日志的长度
SLOWLOG RESET	清除慢日志中的所有条目
SLOWLOG HELP	显示有关不同子命令的有用文本
SMEMBERS key	集齐所有成员
SMOVE source destination member	将成员从一组移动到另一组
SORT key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC|DESC] [ALPHA] [STORE destination]	对列表、集合或有序集合中的元素进行排序
SORT_RO key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC|DESC] [ALPHA]	对列表、集合或有序集合中的元素进行排序， SORT 的只读变体
SPOP key [count]	从集合中删除并返回一个或多个随机成员
SRANDMEMBER key [count]	从集合中获取一个或多个随机成员
SREM key member [member ...]	从集合中删除一个或多个成员
STRLEN key	获取存储在键中的值的长度
SUBSCRIBE channel [channel ...]	收听发布到给定频道的消息
SUNION key [key ...]	返回所有给定集合的并集所产生的集合的成员
SUNIONSTORE destination key [key ...]	返回所有给定集合的并集并存储到指定集合中
SWAPDB index1 index2	交换两个Redis数据库
SYNC	内部命令，从主站发起复制流
PSYNC replicationid offset	内部命令，从主站发起复制流
TIME	返回当前服务器时间
TOUCH key [key ...]	更改键的最后访问时间，返回指定的现有键的数量
TTL key	在几秒钟内获得一把钥匙的生存时间
TYPE key	确定存储在 key 的类型
UNSUBSCRIBE [channel [channel ...]]	停止收听发布到给定频道的消息

UNLINK key [key ...]	在另一个线程中异步删除一个键，否则它就像 DEL 一样，但非阻塞
UNWATCH	忘记所有观看过的钥匙吧
WAIT numreplicas timeout	等待同步复制当前连接上下文中发送的所有写命令
WATCH key [key ...]	观察给定的键以确定 MULTI/EXEC 块的执行
ZADD key [NX|XX] [GT|LT] [CH] [INCR] score member [score member ...]	将一个或多个成员添加到有序集中，或者如果它已经存在则更新其分数
ZCARD key	获取有序集合中的成员数量
ZCOUNT key min max	用给定值内的分数计算有序集中的成员
ZDIFF numkeys key [key ...] [WITHSCORES]	减去多个有序集
ZDIFFSTORE destination numkeys key [key ...]	减去多个有序集并将结果有序集存储在一个新键中
ZINCRBY key increment member	在有序集中增加成员的分数
ZINTERCARD numkeys key [key ...] [LIMIT limit]	将多个有序集合相交并返回结果的基数
ZINTERSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]	将多个有序集相交并将结果有序集存储在一个新键中
ZLEXCOUNT key min max	计算给定字典范围内有序集中的成员数量
ZPOPMAX key [count]	删除并返回有序集中得分最高的成员
ZPOPMIN key [count]	删除并返回有序集中得分最低的成员
ZMPOP numkeys key [key ...] MIN|MAX [COUNT count]	删除并返回具有有序集中分数的成员
ZRANDMEMBER key [count [WITHSCORES]]	从有序集合中获取一个或多个随机元素
ZRANGESTORE dst src min max [BYSCORE|BYLEX] [REV] [LIMIT offset count]	将有序集合中的一系列成员存储到另一个键中
ZRANGE key min max [BYSCORE|BYLEX] [REV] [LIMIT offset count] [WITHSCORES]	返回有序集合中的一系列成员
ZRANGEBYLEX key min max [LIMIT offset count]	返回成员范围rs 在一个有序的集合中，按字典序排列

ZREVRANGEBYLEX key max min [LIMIT offset count]	返回有序集中的成员范围，按字典序范围，从高到低的字符串排序
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]	按分数返回有序集中的一系列成员
ZRANK key member	确定有序集合中成员的索引
ZREM key member [member ...]	从有序集中删除一个或多个成员
ZREMRANGEBYLEX key min max	删除给定字典范围之间有序集中的所有成员
ZREMRANGEBYRANK key start stop	删除给定索引内有序集中的所有成员
ZREMRANGEBYSCORE key min max	删除给定分数内有序集中的所有成员
ZREVRANGE key start stop [WITHSCORES]	按索引返回有序集中的一系列成员，分数从高到低排序
ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]	按分数返回有序集中的一系列成员，分数从高到低排序
ZREVRANK key member	确定一个有序集合中某个成员的索引，分数从高到低排序
ZSCORE key member	在有序集中获取与给定成员关联的分数
ZMSCORE key member [member ...]	获取与有序集中的给定成员相关联的分数
ZUNIONSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]	添加多个有序集并将结果有序集存储在一个新键中
SCAN cursor [MATCH pattern] [COUNT count] [TYPE type]	增量迭代键空间
SSCAN key cursor [MATCH pattern] [COUNT count]	增量迭代 Set 元素
HSCAN key cursor [MATCH pattern] [COUNT count]	增量迭代哈希字段和关联值
ZSCAN key cursor [MATCH pattern] [COUNT count]	增量迭代已排序的集合元素和相关分数
XINFO CONSUMERS key groupname	列出消费者组中的消费者
XINFO GROUPS key	列出流的消费者组
XINFO STREAM key [FULL [COUNT count]]	获取有关流的信息
XINFO HELP	显示有关不同子命令的有用文本
XADD key [NOMKSTREAM] [MAXLEN|MINID [=|~] threshold	将新条目附加到流中

[LIMIT count]] *|ID field value [field value ...]	
XTRIM key MAXLEN|MINID [=|~] threshold [LIMIT count]	将流修剪到（大约如果 '~' 被传递）特定大小
XDEL key ID [ID ...]	从流中删除指定的条目，返回实际删除的项目数，如果某些 ID 不存在，则可能与传递的 ID 数不同
XRANGE key start end [COUNT count]	返回流中的一系列元素，其 ID 与指定的 ID 间隔相匹配
XREVRANGE key end start [COUNT count]	与 XRANGE 相比，以相反的顺序（从较大到较小的 ID）返回流中的一系列元素，其中 ID 与指定的 ID 间隔匹配
XLEN key	返回流中的条目数
XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]	返回多个流中从未见过的元素，其 ID 大于调用者为每个流报告的 ID，可以屏蔽
XGROUP CREATE key groupname id|$ [MKSTREAM]	创建一个消费者组
XGROUP CREATECONSUMER key groupname consumername	在消费者组中创建消费者
XGROUP DELCONSUMER key groupname consumername	从消费者组中删除消费者
XGROUP DESTROY key groupname	销毁一个消费组
XGROUP SETID key groupname id|$	将消费者组设置为任意最后交付的 ID 值
XGROUP HELP	显示有关不同子命令的有用文本
XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] [NOACK] STREAMS key [key ...] ID [ID ...]	使用消费者组从流中返回新条目，或访问给定消费者的待处理条目的历史记录，可以屏蔽
XACK key group ID [ID ...]	将待处理消息标记为正确处理，有效地将其从消费者组的待处理条目列表中删除，该命令的返回值是成功确认的消息数，即我们在 PEL 中实际能够解析的 ID
XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]	在流消费者组的上下文中，此命令更改挂起消息的所有权，以便新的所有者是作为命令参数指定的消费者
XAUTOCLAIM key group consumer min-idle-time start [COUNT count] [JUSTID]	在流消费者组的上下文中，此命令自动更改挂起消息的所有权，以便新的所有者是作为

	命令参数指定的消费者
XPENDING key group [[IDLE min-idle-time] start end count [consumer]]	从流消费者组待处理条目列表中返回信息和条目，即获取但从未确认的消息

6.1.2 AMDC集群使用指南

AMDC集群为高可用可扩展集群，集群方式分别为主从模式、哨兵模式、cluster集群模式。

6.1.2.1 主从模式

AMDC主从模式，即（Master-Slave Replication）主从复制，使用一个AMDC实例作为主机，其余的作为备份机，主机和备份机的数据完全一致，主机支持数据的写入和读取等各项操作，而从机则支持与主机数据的同步和读取，当其中一台AMDC出现故障时，访问其他结点的AMDC缓存核心即可。

6.1.2.1.1 主从命令

- 转变为某个节点的从节点：slaveof [ip] [port]（replicaof no one 将使从节点转变为主节点）。

6.1.2.2 哨兵模式

哨兵是一个自动监控处理AMDC缓存核心间故障节点转移工作的AMDC缓存核心端程序，AMDC提供哨兵的命令，哨兵通过发送命令，等待AMDC缓存核心响应，从而监控运行的多个AMDC实例。

6.1.2.2.1 哨兵命令

1. 查看sentinel的状态命令：
   info
2. 获取sentinel中监控的所有master的节点命令：
   sentinel masters
3. 获取master-name主节点的状态信息命令：
   sentinel master [master-name]
4. 获取master-name节点下所有的slaves的状态信息命令：
   sentinel slaves [master-name]
5. 通过sentinel中节点名获取ip地址命令：
   sentinel get-master-addr-by-name [master-name]
6. 添加节点命令：
   sentinel monitor [name] [ip] [port] [quorum]
7. 重置amdc name匹配指定的状态命令：
   sentinel reset [master-name]
8. 删除节点命令：
   sentinel remove [master-name]
9. 强制节点主观下线命令：
   sentinel failover [master-name]

10. 获取哨兵集群中其他哨兵节点信息:
    sentinel sentinels
11. 为指定的主节点设置访问密码:
    sentinel set [master-name] auth-pass [password]

6.1.2.3 集群模式

集群模式是AMDC实现主节点的弹性伸缩，主要作用于增大或者减少AMDC可以使用的内存容量，实现通过扩充节点来满足业务的缓存需求，无需通过增加服务器内存来实现；并且集群模式有与哨兵类似自动故障转移功能，具有高可用性，是更好的选择。 集群模式所有节点都能互相通信，感知对方的状态信息，集群可以自动分配主从节点也可以手动指定这些节点

6.1.2.3.1 集群命令

1. 为接收节点分配新的哈希插槽:
   CLUSTER ADDSLOTS slot [slot ...]
2. 命令从连接的节点触发群集配置年龄的增量。如果节点的配置年龄为零，或者小于群集的最大年龄，则该年龄将递增:
   CLUSTER BUMPEPOCH
3. 返回为给定节点处于活动的故障报告的数量:
   CLUSTER COUNT-FAILURE-REPORTS node-id
4. 返回指定哈希插槽中的本地键数:
   CLUSTER COUNTKEYSINSLOT slot
5. 将哈希插槽设置为接收节点中的未绑定:
   CLUSTER DELSLOTS slot [slot ...]
6. 强制副本执行其主站的手动故障转移:
   CLUSTER FAILOVER [FORCE|TAKEOVER]
7. 删除节点自己的插槽信息:
   CLUSTER FLUSHSLOTS
8. 从节点表中删除节点:
   CLUSTER FORGET node-id
9. 在指定的哈希插槽中返回本地键名称:
   CLUSTER GETKEYSINSLOT slot count
10. 提供有关AMDC Cluster节点状态的信息:
    CLUSTER INFO
11. 返回指定键的哈希槽:
    CLUSTER KEYSLOT key
12. 强制一个节点集群与另一个节点握手:
    CLUSTER MEET ip port
13. 返回节点 id:
    CLUSTER MYID

14. 获取节点的集群配置：
    CLUSTER NODES
15. 将当前节点设置为指定主节点的从节点（slave）：
    CLUSTER REPLICATE [master-node-id]
16. 将 key 原子性迁移到另一个实例:
    MIGRATE [host] [port] [key] [destination-db] [timeout] [COPY] [REPLACE] [KEYS key1 key2...]
17. 设置槽的状态（导入、导出、归属）：
    CLUSTER SETSLOT [STATE] []
18. 强制当前节点将当前集群配置立即保存到 nodes.conf 文件:
    CLUSTER SAVECONFIG
19. 重置节点的集群配置:
    CLUSTER RESET [HARD|SOFT]
20. 设置节点的配置纪元:
    CLUSTER SET-CONFIG-EPOCH [epoch]
21. 返回指定主节点的所有副本节点信息:
    CLUSTER REPLICAS [node-id]
22. 返回集群中所有槽的映射信息，包括每个槽的主节点和副本节点:
    CLUSTER SLOTS

6.1.2.3.2 集群运维命令

以下命令是 AMDC 提供的用于集群日常运维的辅助工具，通过 amdc-cli 执行（非 Cluster 协议命令）

命令	说明
`amdc-cli --cluster create ip:port1 ... --cluster-replicas 1`	创建集群（自动 meet + 分配槽 + 主从）
`amdc-cli --cluster check ip:port`	检查集群健康状态
`amdc-cli --cluster info ip:port`	查看集群统计信息
`amdc-cli --cluster fix ip:port`	修复槽未覆盖等问题（实验性）
`amdc-cli --cluster reshard ip:port`	在线重新分片（迁移槽）
`amdc-cli --cluster add-node new_ip:new_port existing_host:existing_port`	添加新节点
`amdc-cli --cluster del-node host:port node_id`	删除节点

> ***说明**：这些命令主要用于简化集群的创建、检查、维护和扩展操作，适用于管理员进行集群管理。其中 `--cluster fix` 和 `--cluster reshard` 是高级功能，建议在充分理解其行为后再使用。*

6.1.3 SSL使用

SSL是AMDC内置的加密通讯协议，启动SSL即可实现与客户端的SSL双向认证加密通讯，可以使用OPENSSL生成相应的证书和密钥文件。

SSL配置在amdc.yaml和sentinel.yaml中都存在，使用方式一致，具体配置解析可参考缓存配置文件章节或如下：

6.1.3.1 SSL配置项

根据TLS配置的常见参数，以下是补充完整的表格：

参数名称	解析	使用
tls-port	SSL监听端口，如果仅开启SSL监听，需要把Network下的port端口设置为0	port: 6369
tls-cert-file	服务端SSL证书	tls-cert-file: "./certs/ssl_tls_cert/server.crt"
tls-key-file	服务端SSL证书的密钥	tls-key-file: "./certs/ssl_tls_cert/server.key"
tls-key-file-pass	服务端SSL证书密钥的密码	tls-key-file-pass: "password"
tls-ca-cert-file	证书签发机构的证书文件，即可信的根证书	tls-ca-cert-file: "./certs/ssl_tls_cert/ca.crt"
tls-ca-cert-dir	证书签发机构的证书文件路径，如有多个可信根证书，可使用该参数配置	tls-ca-cert-dir: ""
tls-client-cert-file	客户端SSL证书，为集群/主从模式使用	tls-client-cert-file: "./certs/ssl_tls_cert/client.crt"
tls-client-key-file	客户端SSL证书的密钥，为集群/主从模式使用	tls-client-key-file: "./certs/ssl_tls_cert/client.key"
tls-client-key-file-pass	客户端SSL证书密钥的密码	tls-client-key-file-pass: "password"
tls-dh-params-file	Diffie-Hellman参数文件，用于前向保密	tls-dh-params-file: "./certs/ssl_tls_cert/dhparam.pem"

tls-protocols	TLS协议版本，指定允许使用的TLS协议版本	tls-protocols: "TLSv1.2 TLSv1.3"
tls-ciphers	SSL/TLS加密算法套件（用于TLSv1.2及以下版本）	tls-ciphers: "ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256"
tls-ciphersuites	TLSv1.3密码套件（用于TLSv1.3版本）	tls-ciphersuites: "TLS_CHACHA20_POLY1305_SHA256:TLS_AES_256_GCM_SHA384"
tls-prefer-server-ciphers	是否优先使用服务器端定义的加密套件而非客户端定义的	tls-prefer-server-ciphers: "yes"
tls-session-caching	是否启用TLS会话缓存	tls-session-caching: "yes"
tls-session-cache-size	TLS会话缓存的最大大小	tls-session-cache-size: 1000
tls-session-cache-timeout	TLS会话缓存的超时时间（秒）	tls-session-cache-timeout: 300
tls-auth-clients	服务端是否验证客户端证书，默认需要客户端提供证书并且服务端做验证，如果不需要客户端可以配置为"no"，设置为optional，则客户端可以提供证书进行验证，也可以不提供	tls-auth-clients: ""
tls-replication	主从模式是否使用TLS通讯，当master当前参数为true，那么从节点也必须为true	tls-replication: false
tls-cluster	集群模式是否使用TLS通讯	tls-cluster: false

6.1.3.2 通过OPENSSL生成证书

可以使用其他方法获得所需的证书，只要合法可用即可。

对于SSL双向认证：

- 服务器需要CA证书、server证书、server私钥；
- 客户端需要CA证书，client证书、client私钥。

步骤：

1. 下载安装openssl，官网/source/index.html (openssl.org)
2. openssl.cnf文件：

```
[ server_cert ]
keyUsage = digitalSignature, keyEncipherment
nsCertType = server

[ client_cert ]
keyUsage = digitalSignature, keyEncipherment
nsCertType = client
```

3. 创建CA证书：
   - `openssl genrsa -out ca.key 4096`
   - `openssl req -x509 -new -nodes -sha256 -key ca.key -days 3650 -subj "/O=APUSIC /CN=AMDC.com" -out ca.crt`
4. 创建服务器私钥与证书
   - `openssl genrsa -out amdc.key 2048`
   - `openssl req -new -sha256 -subj "/O=APUISC /CN=AMD.com" -key amdc.key | openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -days 365 -extfile openssl.cnf -extensions server_cert -out amdc.crt`
5. 创建客户端私钥与证书
   - `openssl genrsa -out client.key 2048`
   - `openssl req -new -sha256 -subj "/O=APUISC /CN=amdc.com" -key client.key | openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -days 365 -extfile openssl.cnf -extensions server_cert -out client.crt`

6.1.3.3 服务器SSL启动

1. 修改amdc.yaml（关键段落）

```
TLS/SSL:
  # TLS 监听端口，需配合 port 0 使用以仅启用 TLS
  tls-port: 0
  # 服务器 TLS 证书文件路径（PEM 格式）
  tls-cert-file: "./cert/amdc.crt"
  # 服务器 TLS 私钥文件路径（PEM 格式）
  tls-key-file: "./cert/amdc.key"
  # TLS 私钥文件的密码（若加密）
  tls-key-file-pass: ""
  # 客户端 TLS 证书文件路径（用于客户端身份验证）
  tls-client-cert-file: ""
  # 客户端 TLS 私钥文件路径
  tls-client-key-file: ""
  # 客户端 TLS 私钥文件的密码（若加密）
  tls-client-key-file-pass: ""
  # DH 参数文件路径
  tls-dh-params-file: ""
  # 用于验证客户端和节点的 CA 证书文件路径
  tls-ca-cert-file: "./cert/cat.crt"
```

2. 启动amdc `./amdc-server amdc.yaml`

6.1.3.4 客户端SSL连接

- 使用amdc-cli连接：`./amdc-cli -p 6369 --tls --cert ./client.crt --key ./client.key --cacert ./ca.crt`

- Go语言连接：

```go
func TestTls(t *testing.T) {
    caCert, err := ioutil.ReadFile("./certs/ca.crt")
    if err != nil {
        panic(err)
    }
    caCertPool := x509.NewCertPool()
    caCertPool.AppendCertsFromPEM(caCert)
    cliCert, err := tls.LoadX509KeyPair("./certs/client.crt", "./certs/client.key")
    if err != nil {
        panic(err)
    }
    cli := redis.NewClient(&redis.Options{
        Addr: "127.0.0.1:6369",
        TLSConfig: &tls.Config{
            Certificates:       []tls.Certificate{cliCert}, //客户端证书，双向认证必须携带
            RootCAs:            caCertPool,                 //校验服务器证书【CA证书】
            InsecureSkipVerify: true,                       //不用校验服务器证书
        },
        DialTimeout: time.Minute,
    })

    fmt.Println(cli.Info("server").String())
}
```

6.1.4 数据持久化

amdc用户提供了RDB、AOF两种持久化方式，在操作命令中也有相关的操作方式，在此将展开两种持久方式的使用以及相关的工具。

6.1.4.1 RDB

RDB持久化方式，会将amdc中的数据以RDB格式存储到硬盘中。

6.1.4.1.1 生成RDB文件的方式

生成RDB文件的方式有三种：

1. 配置文件中进行save配置RDB的生成条件，触发条件时系统自动使用bgsave生成RDB文件；
2. 使用save命令，使用该命令后将会阻塞所有请求，在RDB文件生成后，恢复请求处理；
3. 使用bgsave命令，该命令不会阻塞请求，但是生成RDB文件的速度会比save慢。

6.1.4.1.2 RDB恢复数据的方式

1. 将一个或多个RDB文件放到缓存的配置文件dir中指定的目录下，启动amdc将会自动加载RDB数据。

6.1.4.1.3 恢复损坏的RDB文件

amdc提供RDB文件修复工具

使用方式：`./amdc-check-rdb [rdb-filename.rdb]`

6.1.4.2 AOF

AOF持久化方式，会将amdc中的数据以aof格式写入到文本文件中，存储到appendonlydir文件夹下，并且amdc将会以追加的方式将新的请求追加到文本中。

6.1.4.2.1 生成AOF文件的方式

1. 在配置文件中将AppendOnly改为"yes"即可。

6.1.4.2.2 AOF数据恢复方式

1. 将一个或多个AOF文件或整个appendonlydir文件夹放到缓存的配置文件dir中指定的目录下，启动amdc将会自动加载AOF数据。

6.1.4.2.3 恢复损坏的AOF文件

amdc提供AOF文件修复工具，但是此工具使用的前提是，必须是最后一个追加文件损坏才能修复。这么设计的原因是，若是base文件就已经损坏，就意味着大部分的数据都已经丢失，没有修复的必要。

使用方式：`./aof-check-aof --fix [filename.aof|filename.mainfest]`

6.1.5 命令审计

AMDC提供monitor命令，可以监控所有的请求信息。配合shell客户端实现监控内容输出到特定的文件中，以此来实现。

请求信息的记录格式：`时间戳 [数据库号 客户端地址：客户端端口号] 命令`

使用方式：`amdc-cli -h [ip] -p [port] [-a passowrd] monitor >amdc_commands.log`

6.2 amdc-cli客户端使用介绍

为了方便AMDC使用，shell客户端给用户带来方便快捷的命令行使用方式，并提供帮助建立集群、管理集群slot、LRU测试等实用功能。

6.2.1 客户端参数

使用方式：`amdc-cli [参数] [命令 [命令参数 ...]]`

参数	说明
-h [hostname]	缓存核心的ip（默认值：127.0.0.1）
-p [port]	缓存核心的端口（默认值：6359）
-a [password]	缓存核心的requirepass密码，可以使用AMDCCLI_AUTH环境变量来设置和输入密码，这样会更安全
--user [username]	使用ACL用户登录时的用户名

--pass [password]	使用ACL用户登录时对应的密码
--askpass	无视AMDCCLI_AUTH环境变量，强制使用直接输入的密码
-u [uri]	缓存核心的uri地址（兼容redis协议）
-r [repeat]	重复执行特定的命令[repeat]次
-i [interval]	使用-r的时候，设置每隔多少秒执行重复一次
-n [db]	数据库编号（默认有16个数据库，编号0-15）
-3	切换为RESP3协议
-x	从stdin读取的输入做为amdc-cli的最后一个参数[br/]例： amdc-cli -x set key [/opt/file
-d [delimiter]	原始格式的响应块之间的分隔符(如：\n)
-D [delimiter]	多个原始格式的响应之间分隔符(如：\n)
-c	连接集群模式
-e	当命令执行错误的时候返回错误代码
--raw	使用原始格式输出（当tty不是默认输出设备时）
--no-raw	强制格式化输出，即使标准输出不是tty
--quoted-input	强制将输入作为带引号的字符串处理
--csv	输出为CSV格式
--show-pushes [yn]	是否打印 resp3 推送消息，默认开启
--stat	动态打印缓存核心的状态信息：内存/客户端连接数等
--lru-test	模拟具有 80-20 分布（key的使用度）的缓存工作负载
--replica	模拟一个备份节点展示从主节点接收到的命令
--slave [host:ip]	指定一个节点为当前连接节点的主节点
--rdb [filename]	从缓存核心获取一个RDB文件作为本地文件
--pipe	从stdin传输原始的amdc协议（兼容redis的）到缓存核心
--pipe-timeout [n]	pipe模式下的超时时间
--bigkeys	查找value内存占用大的key

--memkeys	查找key-value内存占用大的key
--memkeys-samples	查找key-value内存占用大的key，输出的信息更简洁
--hotkeys	查找使用次数多的key，但是需要缓存策略为lfu
--scan	用scan命令列出所有的key
--pattern [pat]	使用--scan/--bigkeys/--hotkeys的时候进行key的正则表达式匹配，默认为*
--qouted-pattern [pat]	和--pattern差不多，但是指定字符串类型需要用引号括起来，否则会被认为是非二进制安全字符串
--intrinsic-latency [sec]	系统固有延迟测试，测试会持续多少秒
--eval [file]	发送和执行LUA脚本文件
--ldb	和--eval一起使用，启用amdc lua debugger（调试器）
--ldb-sync-mode	和--ldb一样，但是会与debugger同步，缓存核心将会被阻塞
--cluster [command] [args...] [opts...]	集群管理命令和参数，详细命令和参数可以使用
--cluster help	查看帮助信息
--verbose	详细模式
--no-auth-warning	使用-a直接输入密码时不暂时warning信息
--help	帮助信息
--version	amdc-cli 版本信息
--tls	Establish a secure TLS connection.
--tls-host	Server name indication for TLS.
--cacert	CA Certificate file to verify with.
--cacert-dir	Directory where trusted CA certificates are stored.

```
                    If neither cacert nor cacert-dir are specified,
the default
                    system-wide trusted root certs configuration will
apply.                                                          |
```

| --insecure | Allow insecure TLS connection by skipping cert validation. | | --cert | Client certificate to authenticate with. | | --key | Private key file to authenticate with. | | --tls-ciphers | Sets the list of preferred ciphers (TLSv1.2 and below) in order of preference from highest to lowest separated by colon (":"). See the ciphers(1ssl) manpage for more information about the syntax of this string. | | --tls-ciphersuites | Sets the list of preferred ciphersuites (TLSv1.3) in order of preference from highest to lowest separated by colon (":"). See the ciphers(1ssl) manpage for more information about the syntax of this string, and specifically for TLSv1.3 ciphersuites. | | --latency | Enter a special mode continuously sampling latency. If you use this mode in an interactive session, it runs forever displaying real-time stats. Otherwise if --raw or --csv is specified, or if you redirect the output to a non TTY, it samples the latency for 1 second (you can use -i to change the interval), then produces a single output and exits. | | --latency-history | Like --latency but tracking latency changes over time. Default time interval is 15 sec. Change it using -i. | | --latency-dist | Shows latency as a spectrum, requires xterm 256-colors. Default time interval is 1 sec. Change it using -i. |

使用方式：amdc-cli --cluster [命令 [命令参数 ...]]

一级参数	二级参数	说明
create host1:port1 ... hostN:portN		创建集群
	--cluster-replicas [arg]	从节点个数
check host:port		检查集群
	--cluster-search-multiple-owners	检查是否有槽同时被分配给了多个节点
info host:port		查看集群状态
fix host:port		修复集群
	--cluster-search-multiple-owners	修复槽的重复分配问题
reshard host:port		指定集群的任意一节点进行迁移slot，重新分slots
	--cluster-from [arg]	需要从哪些源节点上迁移slot，可从多个源节点完成迁移，以逗号隔开，传递的是节点的node id，还可以直接传递--from all，这样源节点就是集群的所有节点，不传递该参数的话，则会在迁移过程中提示用户输入
	--cluster-to [arg]	slot需要迁移的目的节点的node id，目的节点只能填写一个，不传递该参数的话，则会

		在迁移过程中提示用户输入
	--cluster-slots [arg]	需要迁移的slot数量，不传递该参数的话，则会在迁移过程中提示用户输入。
	--cluster-yes	指定迁移时的确认输入
	--cluster-timeout [arg]	设置migrate命令的超时时间
	--cluster-pipeline [arg]	定义cluster getkeysinslot命令一次取出的key数量，不传的话使用默认值为10
	--cluster-replace	是否直接replace到目标节点
rebalance host:port		指定集群的任意一节点进行平衡集群节点slot数量
	--cluster-weight [node1=w1...nodeN=wN]	指定集群节点的权重
	--cluster-use-empty-masters	设置可以让没有分配slot的主节点参与，默认不允许
	--cluster-timeout [arg]	设置migrate命令的超时时间
	--cluster-simulate	模拟rebalance操作，不会真正执行迁移操作
	--cluster-pipeline [arg]	定义cluster getkeysinslot命令一次取出的key数量，默认值为10
	--cluster-threshold [arg]	迁移的slot阈值超过threshold，执行rebalance操作
	--cluster-replace	是否直接replace到目标节点
add-node new_host:new_port existing_host:existing_port		添加节点，把新节点加入到指定的集群，默认添加主节点
	--cluster-slave	新节点作为从节点，默认随机一个主节点
	--cluster-master-id [arg]	给新节点指定主节点
del-node host:port node_id		删除给定的一个节点，成功后关闭该节点服务
call host:port command arg arg .. arg		在集群的所有节点执行相关命令

set-timeout host:port milliseconds		设置cluster-node-timeout
import host:port		将外部amdc数据导入集群(非cluster模式的节点)
	--cluster-from [arg]	将指定实例的数据导入到集群
	--cluster-copy	migrate时指定copy
	--cluster-replace	migrate时指定replace

6.2.2 客户端使用

AMDC客户端语言涵盖了主流的编程语言，例如Java、PHP、Python、C、C++、Node.js等，可以通过amdc-cli进行连接，同时也支持redis客户端连接amdc进行操作，两者效果一直。

6.2.2.1 一般操作

- 通过url方式连接amdc-server

  ```
  ./amdc-cli -u amdc://user:password@127.0.0.1:6359/db
  ```

- 显示amdc-cli 命令帮助信息 --help

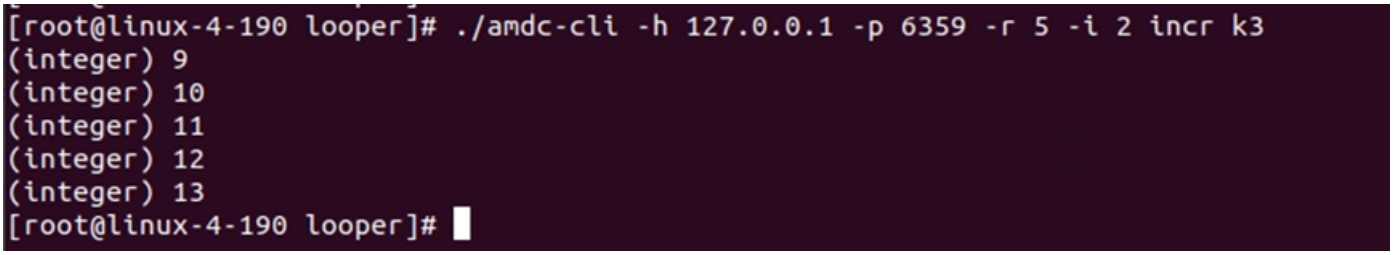
  略。

- 使用amdc-cli客户端连接amdc

  > ```
  > ./amdc-cli -h host -p port -a password
  > ```
  >
  > > *-h 用于指定 ip*
  > > *-p 用于指定端口*
  > > *-a 用于指定认证密码*

- 将返回数据输出到当前命令行 --raw（隐藏数据类型）和--no-raw

- 连续运行n次相同命令 -r 设置运行间隔时间-i（秒）



```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 -r 5 -i 2 incr k3
(integer) 9
(integer) 10
(integer) 11
(integer) 12
(integer) 13
[root@linux-4-190 looper]#
```

- 连接指定db： -n

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 -n 2
127.0.0.1:6359[2]>
127.0.0.1:6359[2]>
127.0.0.1:6359[2]> set k5 v
OK
127.0.0.1:6359[2]> get k5
"v"
```

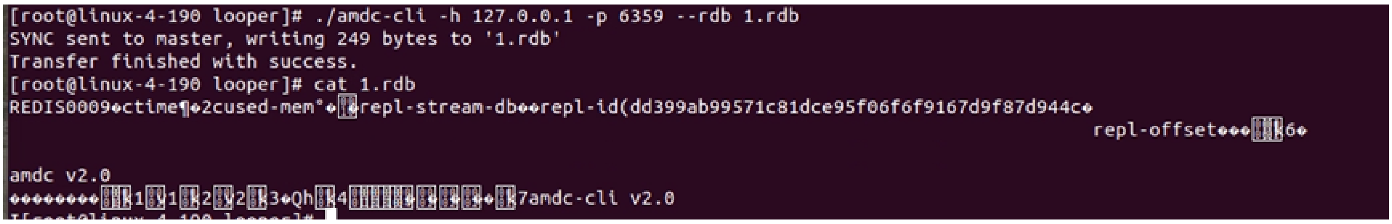
- 执行命令输出以逗号分隔的格式(csv格式) --csv

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --csv lrange k4 0 -1
"6","5","4","3","2","1","0"
[root@linux-4-190 looper]#
```

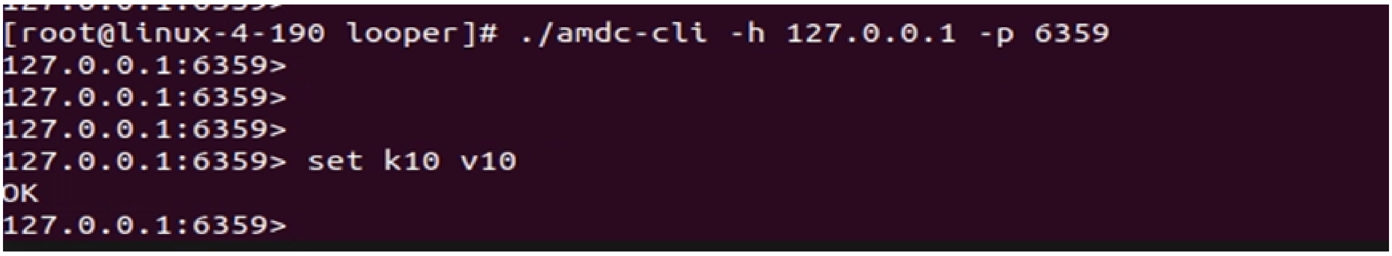
- 标准输入（stdin）读取数据作为amdc-cli的最后一个参数 -x

```
[root@linux-4-190 looper]# echo "amdc-cli v2.0" | ./amdc-cli -h 127.0.0.1 -p 6359 -x set k7
OK
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359
127.0.0.1:6359> get k7
"amdc-cli v2.0\n"
127.0.0.1:6359>
```

- rdb备份 --rdb



- 从机模式，将客户端当作服务端的从节点 --slave



- 管道模式，一次执行多个命令 --pipe

```
[root@linux-4-190 looper]# cat a.txt
set k50 v50
rpush k60 1 2 3 4 5 6 7 8 9
hset k70 key70 val70
[root@linux-4-190 looper]# cat a.txt | ./amdc-cli -h 127.0.0.1 -p 6359 --pipe
All data transferred. Waiting for the last reply...
Last reply received from server.
errors: 0, replies: 3
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359
127.0.0.1:6359> get k50
"v50"
127.0.0.1:6359> lrange k60 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "8"
9) "9"
127.0.0.1:6359> HGET k70 key70
"val70"
127.0.0.1:6359>
```

6.2.2.2 统计操作

- 连续统计模式，实时查看amdc key个数，占用内存等数据 --stat

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --stat
------- data ------ --------------------- load -------------------- - child -
keys       mem      clients blocked requests            connections
6          1.04M    2       0       26778 (+0)          37
6          1.13M    2       0       26779 (+1)          37
6          1.18M    2       0       26780 (+1)          37
6          1.22M    2       0       26781 (+1)          37
6          1.26M    2       0       26782 (+1)          37
6          1.30M    2       0       26783 (+1)          37
6          1.34M    2       0       26784 (+1)          37
```

- 查找大key --bigkeys

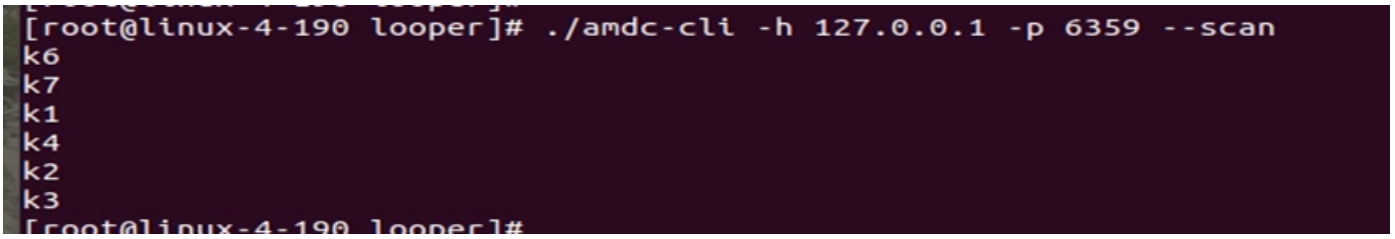
```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --bigkeys

# Scanning the entire keyspace to find biggest keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).

[00.00%] Biggest string found so far 'k6' with 1024 bytes
[00.00%] Biggest list   found so far 'k4' with 4 items

-------- summary -------

Sampled 6 keys in the keyspace!
Total key length in bytes is 12 (avg len 2.00)

Biggest string found 'k6' has 1024 bytes
Biggest   list found 'k4' has 4 items

0 hashs with 0 fields (00.00% of keys, avg size 0.00)
0 sets with 0 members (00.00% of keys, avg size 0.00)
5 strings with 1047 bytes (83.33% of keys, avg size 209.40)
0 zsets with 0 members (00.00% of keys, avg size 0.00)
1 lists with 4 items (16.67% of keys, avg size 4.00)
0 streams with 0 entries (00.00% of keys, avg size 0.00)
[root@linux-4-190 looper]#
```

- 查找热key --hotkeys

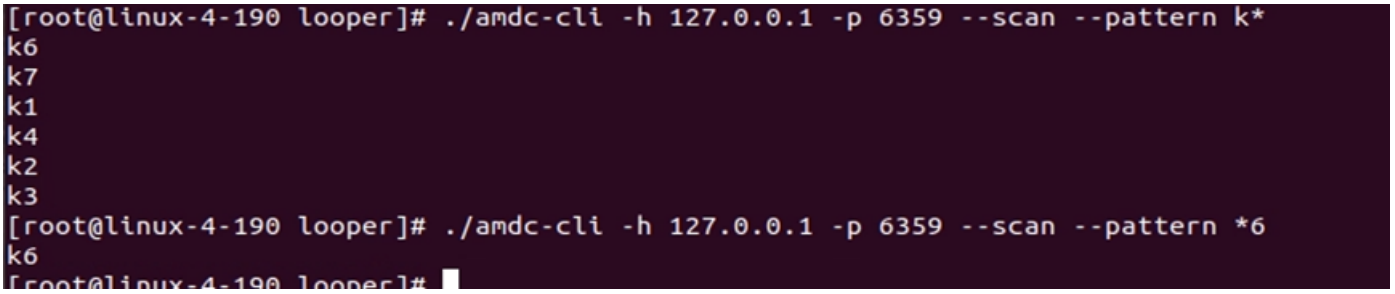
```
# Scanning the entire keyspace to find biggest keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).

[00.00%] Biggest string found so far '"k7"' with 2 bytes
[00.00%] Biggest zset   found so far '"k30"' with 3 members
[00.00%] Biggest string found so far '"k18"' with 3 bytes
[00.00%] Biggest list   found so far '"k4"' with 8 items
[50.00%] Biggest set    found so far '"k40"' with 3 members
[50.00%] Biggest hash   found so far '"k5"' with 2 fields

-------- summary -------

Sampled 20 keys in the keyspace!
Total key length in bytes is 51 (avg len 2.55)

Biggest   list found '"k4"' has 8 items
Biggest   hash found '"k5"' has 2 fields
Biggest string found '"k18"' has 3 bytes
Biggest    set found '"k40"' has 3 members
Biggest   zset found '"k30"' has 3 members

1 lists with 8 items (05.00% of keys, avg size 8.00)
1 hashs with 2 fields (05.00% of keys, avg size 2.00)
16 strings with 41 bytes (80.00% of keys, avg size 2.56)
0 streams with 0 entries (00.00% of keys, avg size 0.00)
1 sets with 3 members (05.00% of keys, avg size 3.00)
1 zsets with 3 members (05.00% of keys, avg size 3.00)
[root@linux-4-190 looper]#
```

6.2.2.3 查询操作

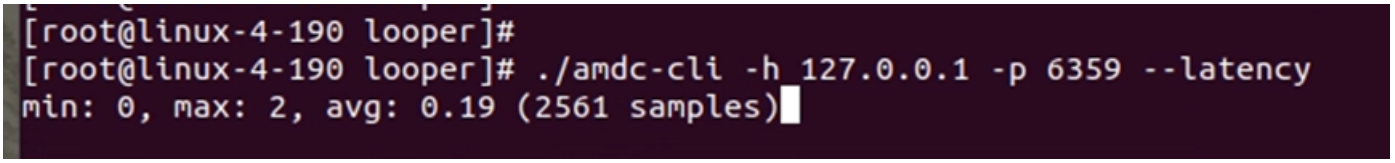
- 获取所有键列表 --scan

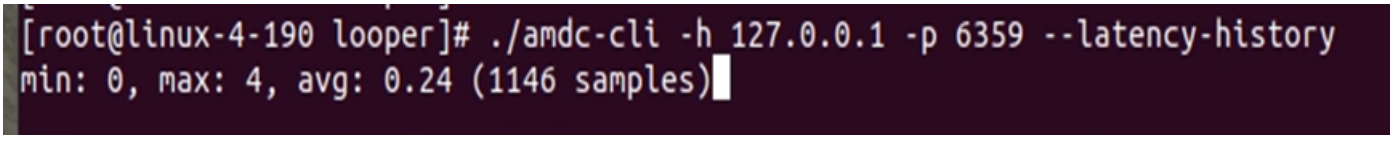


- 扫描指定的key --pattern （正则表达式）



- 查看一段时间内 amdc的最小、最大、平均访问延迟 --latency或--latency-history





6.2.2.4 测试操作

- 测量n秒内amdc的延迟时间 --intrinsic-latency

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --intrinsic-latency 5
Max latency so far: 6 microseconds.
Max latency so far: 10 microseconds.
Max latency so far: 12 microseconds.
Max latency so far: 13 microseconds.
Max latency so far: 18 microseconds.
Max latency so far: 20 microseconds.
Max latency so far: 28 microseconds.
Max latency so far: 49 microseconds.
Max latency so far: 382 microseconds.
Max latency so far: 400 microseconds.
Max latency so far: 1170 microseconds.

1236720 total runs (avg latency: 4.0430 microseconds / 4042.95 nanoseconds per run).
Worst run took 289x longer than the average latency.
[root@linux-4-190 looper]#
```

- LRU 模拟，模拟缓存淘汰策略 --lru-test

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --lru-test 100000
28000 Gets/sec | Hits: 27884 (99.59%) | Misses: 116 (0.41%)
28250 Gets/sec | Hits: 28250 (100.00%) | Misses: 0 (0.00%)
28000 Gets/sec | Hits: 28000 (100.00%) | Misses: 0 (0.00%)
27000 Gets/sec | Hits: 26969 (99.89%) | Misses: 31 (0.11%)
29000 Gets/sec | Hits: 28929 (99.76%) | Misses: 71 (0.24%)
28750 Gets/sec | Hits: 28738 (99.96%) | Misses: 12 (0.04%)
29500 Gets/sec | Hits: 29500 (100.00%) | Misses: 0 (0.00%)
28500 Gets/sec | Hits: 28500 (100.00%) | Misses: 0 (0.00%)
28500 Gets/sec | Hits: 28382 (99.59%) | Misses: 118 (0.41%)
28250 Gets/sec | Hits: 28250 (100.00%) | Misses: 0 (0.00%)
29500 Gets/sec | Hits: 29500 (100.00%) | Misses: 0 (0.00%)
29250 Gets/sec | Hits: 29250 (100.00%) | Misses: 0 (0.00%)
28750 Gets/sec | Hits: 28750 (100.00%) | Misses: 0 (0.00%)
28500 Gets/sec | Hits: 28251 (99.13%) | Misses: 249 (0.87%)
28750 Gets/sec | Hits: 28750 (100.00%) | Misses: 0 (0.00%)
28750 Gets/sec | Hits: 28500 (99.13%) | Misses: 250 (0.87%)
29500 Gets/sec | Hits: 29500 (100.00%) | Misses: 0 (0.00%)
26250 Gets/sec | Hits: 26022 (99.13%) | Misses: 228 (0.87%)
```

- 模拟显示从主服务器接收到的命令的副本 --replica

```
[root@linux-4-190 looper]# redis-cli -h 127.0.0.1 -p 6359 --replica
sending REPLCONF capa eof
SYNC with master, discarding 478 bytes of bulk transfer...
SYNC done. Logging commands from master.
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
"PING"
```

6.2.2.5 LUA操作

- 执行lua脚本

```
[root@linux-4-190 looper]# cat script.lua
return redis.call('GET',"k1")

[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --eval ./script.lua
"v1"
[root@linux-4-190 looper]#
```

- 启动lua调试模式：--ldb 或 --ldb-sync-mode

```
[root@linux-4-190 looper]# redis-cli -h 127.0.0.1 -p 6359 --eval ./script.lua --ldb
Lua debugging session started, please use:
quit    -- End the session.
restart -- Restart the script in debug mode again.
help    -- Show Lua script debugging commands.

v1
lua debugger>
lua debugger>
[root@linux-4-190 looper]# redis-cli -h 127.0.0.1 -p 6359 --eval ./script.lua --ldb-sync-mode
Lua debugging session started, please use:
quit    -- End the session.
restart -- Restart the script in debug mode again.
help    -- Show Lua script debugging commands.

v1
lua debugger>
lua debugger>
```

6.2.2.6 集群操作

目前多租户版本不支持集群操作

6.3 RDB集群数据迁移工具使用介绍

目前多租户版本不支持集群操作

6.4 性能测试工具使用介绍

amdc-benchmark，是提供给用户测试AMDC的专用性能测试工具，以便快速了解AMDC的性能情况，并为调优提供可靠依据。

6.4.1 AMDC benchmark参数

amdc-benchmark

举例: `amdc-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests>]`

命令参数

```
-h <hostname>      实例host名称 (默认127.0.0.1)
-p <port>          实例端口 (默认6359)
-a <password>      实例密码
-c <clients>       并发连接数 (默认50)
-n <requests>      总请求数 (默认100000)
-d <size>          SET/GET 数据的大小，以bytes为单位 (默认2 )
-dbnum <db>        指定db库 (默认AMDC有0-15号库，当前参数默认0)
-k <boolean>       1=保持连接 0=重新连接(默认1)
-r <keyspacelen>  SET/GET/INCR 使用随机 key， SADD 使用随机值，
keyspacelen指随机数据的最大长度（最大长度范围是0-12)
-P <numreq>        使用管道合并指定数量的请求， 默认1 (不使用管道)
-e                 若服务器返回信息错误，在标准输出中显示（每秒显示错误不超过1
个)
-q                 仅显示 query/sec 值
--csv              输出为CSV格式
-l                 循环，一直进行测试
-t <tests>         指定测试命令
-I                 空闲模式，开个N个空闲连接并等待
--threads <num>   开启指定数量的线程，默认1
```

memtier-benchmark参数

举例: `memtier-benchmark [-h <host>] [-p <port>] [-c <clients>] [-n <requests>] [-t <threads>]`

命令参数

```
-h <hostname>      实例host名称 (默认127.0.0.1)
-p <port>          实例端口 (默认6359)
-s <socket>        使用Unix域套接字
-a <password>      认证密码
--tls              开启TLS加密
-c <clients>       客户端数量
-t <threads>       线程数
--ratio            读写比例
```

```
--pipeline <num>   管道命令数
--randomdata       使用随机数据
--hide-histogram   隐藏详细的直方图信息
--csv <file>       输出csv格式文件
--test-time <sec>  持续测试时间
```

7 产品所有配置说明

7.1 缓存核心所有配置

AMDC缓存核心配置三个可主动配置的配置文件，分别为缓存配置（amdc.yaml）、哨兵配置（sentinel.yaml）和授权中心配置（acls.properties）。

7.1.1 缓存配置文件

AMDC缓存核心的配置文件存放在：/安装根目录/amdc/amdc.yaml，部分配置可以通过控制台进行修改，以下为AMDC缓存核心详细配置：

参数名称	默认值	备注
include	空	用于包含其他配置文件，可实现配置的复用与分层管理，`CONFIG REWRITE` 不会改写此选项
loadmodule	空	启动时加载指定的模块，若加载失败服务器会终止启动，可多次使用以加载多个模块
bind	127.0.0.1 -::1	监听的网络地址，- 前缀表示地址不可用时不影响启动；默认仅监听本地回环地址，注释此行可监听所有接口（暴露在公网有安全风险）
protected-mode	yes	保护模式，当未指定 `bind` 且无密码时，仅允许本地回环地址和 Unix 域套接字连接
port	6359	监听的 TCP 端口，设为 0 则不监听 TCP 套接字
tcp-backlog	511	TCP 监听的 backlog 大小，实际值可能受系统 `somaxconn` 和 `tcp_max_syn_backlog` 限制
unixsocket	空	Unix 域套接字路径，未指定则不监听 Unix 套接字
unixsocketperm	700	Unix 域套接字的权限
timeout	0	客户端空闲超时时间（秒），0 表示禁用
tcp-keepalive	300	TCP 保活时间（秒），用于检测死连接和维持网络连接
license	license.lic	授权文件路径

tls-port	空	TLS 监听端口，需配合 `port 0` 使用以仅启用 TLS
tls-cert-file	空	服务器 TLS 证书文件路径（PEM 格式）
tls-key-file	空	服务器 TLS 私钥文件路径（PEM 格式）
tls-key-file-pass	空	TLS 私钥文件的密码（若加密）
tls-client-cert-file	空	客户端 TLS 证书文件路径（用于客户端身份验证）
tls-client-key-file	空	客户端 TLS 私钥文件路径
tls-client-key-file-pass	空	客户端 TLS 私钥文件的密码（若加密）
tls-dh-params-file	空	DH 参数文件路径，旧版 OpenSSL（<3.0）需配置，新版不推荐
tls-ca-cert-file	空	用于验证客户端和节点的 CA 证书文件路径
tls-ca-cert-dir	空	用于验证客户端和节点的 CA 证书目录路径
tls-auth-clients	空	客户端证书验证策略，`no` 不接受证书，`optional` 可选但需有效
tls-replication	no	是否在复制链接中启用 TLS
tls-cluster	no	是否在集群总线协议中启用 TLS
tls-protocols	"TLSv1.2 TLSv1.3"	允许的 TLS 版本，大小写不敏感
tls-ciphers	DEFAULT:!MEDIUM	TLSv1.2 及以下允许的密码套件
tls-ciphersuites	TLS_CHACHA20_POLY1305_SHA256	TLSv1.3 允许的密码套件
tls-prefer-server-ciphers	no	是否优先使用服务器的密码套件偏好
tls-session-caching	yes	是否启用 TLS 会话缓存以加速重连
tls-session-cache-size	20480	TLS 会话缓存的最大条目数，0 表示无限制
tls-session-cache-timeout	300	TLS 会话缓存的超时时间（秒）

daemonize	no	是否以守护进程模式运行，是则会生成 pid 文件
supervised	no	与监控系统（upstart/systemd）的交互模式，默认不交互
pidfile	/var/run/amdc_6379.pid	pid 文件路径，守护进程模式下默认生成
loglevel	notice	日志级别：`debug`（详细）、`verbose`（较多）、`notice`（适中，推荐生产环境）、`warning`（仅关键信息）
logfile	""	日志文件路径，空字符串表示输出到标准输出
syslog-enabled	no	是否启用系统日志
syslog-ident	amdc	系统日志标识
syslog-facility	local0	系统日志设备，需为 USER 或 LOCAL0-LOCAL7
crash-log-enabled	yes	是否启用崩溃日志
crash-memcheck-enabled	yes	是否在崩溃日志中启用快速内存检查
worker-threads	1	设置工作线程数，默认为 1
databases	16	数据库数量，默认使用 DB 0，可通过 `SELECT` 切换
always-show-logo	no	是否在启动日志中显示 ASCII 艺术 logo，默认仅交互会话显示
set-proc-title	yes	是否修改进程标题以显示运行时信息
proc-title-template	"{title} {listen-addr} {server-mode}"	进程标题模板，支持多个变量
save	3600 1、300 100、60 10000	RDB 持久化触发条件，格式为 <秒数> <修改次数>；`save ""` 禁用 RDB
stop-writes-on-bgsave-error	yes	RDB 持久化失败时是否停止接受写入
rdbcompression	yes	是否使用 LZF 压缩 RDB 文件中的字符串对象
rdbchecksum	yes	是否在 RDB 文件末尾添加 CRC64 校验和，禁用可提升性能

sanitize-dump-payload	no	加载 RDB 或 RESTORE 数据时是否进行全面校验，默认因集群迁移问题暂设为 no
dbfilename	dump.rdb	RDB 文件名称
rdb-del-sync-files	no	无持久化配置时是否删除复制用的 RDB 文件，仅在 AOF 和 RDB 均禁用时有效
dir	./	数据文件（RDB、AOF）的存放目录
replicaof	空	主从复制配置，格式为 `<masterip> <masterport>`
masterauth	空	主服务器的认证密码
masteruser	空	用于复制的主服务器用户名
replica-serve-stale-data	yes	复制同步期间是否响应客户端请求（可能返回过期数据）
replica-read-only	yes	从服务器是否为只读模式
repl-diskless-sync	no	是否启用无盘复制（直接通过套接字传输 RDB）
repl-diskless-sync-delay	5	无盘复制时等待新从节点的延迟时间（秒）
repl-diskless-load	disabled	从节点加载 RDB 的方式，`disabled`（先存盘）、`on-empty-db`（仅空库时无盘）、`swapdb`（内存保留旧数据）
repl-ping-replica-period	10	从节点向主节点发送 PING 的间隔（秒）
repl-timeout	60	复制超时时间（秒），涵盖 SYNC 传输、主从心跳等
repl-disable-tcp-nodelay	no	复制链接是否禁用 TCP_NODELAY，yes 可减少带宽但增加延迟
repl-backlog-size	1mb	复制积压缓冲区大小，影响部分重同步能力
repl-backlog-ttl	3600	最后一个从节点断开后，积压缓冲区释放的延迟时间（秒），0 表示不释放
replica-priority	100	从节点优先级，用于 Sentinel 选主，0 表示不可选为主

replica-announced	yes	从节点是否在 Sentinel 报告中显示
min-replicas-to-write	0	主节点接受写入所需的最少从节点数，0 禁用此功能
min-replicas-max-lag	10	从节点的最大滞后时间（秒），配合 `min-replicas-to-write` 使用
replica-announce-ip	空	从节点向主节点宣告的 IP 地址（用于 NAT 等场景）
replica-announce-port	空	从节点向主节点宣告的端口（用于 NAT 等场景）
tracking-table-max-keys	1000000	客户端缓存跟踪表的最大键数，0 表示无限制
acllog-max-len	128	ACL 日志的最大条目数
aclfile	空	外部 ACL 用户文件路径，与配置文件内用户配置不可混用
requirepass	空	默认用户的密码（等价于 ACL 的 default 用户密码）
acl-pubsub-default	allchannels	新用户默认的 Pub/Sub 频道权限，`allchannels`（允许所有）、`resetchannels`（禁止所有）
rename-command	空	命令重命名（已过时），可禁用危险命令（如 `rename-command CONFIG ""`）
maxclients	10000	最大并发客户端连接数，受系统文件描述符限制
maxmemory	空	内存使用上限，达到后按 `maxmemory-policy` 处理
maxmemory-policy	noeviction	内存达到上限时的淘汰策略，如 `volatile-lru`、`allkeys-lfu` 等
maxmemory-samples	5	LRU/LFU/TTL 算法的采样数，越大越精确但 CPU 消耗越高
maxmemory-eviction-tenacity	10	淘汰处理的激进程度，0（最低延迟）到 100（优先处理淘汰）

replica-ignore-maxmemory	yes	从节点是否忽略自身的 `maxmemory` 配置（默认由主节点处理淘汰）
active-expire-effort	1	主动过期键的扫描力度，1-10，越大越频繁（可能增加 CPU 和延迟）
lazyfree-lazy-eviction	no	内存淘汰时是否使用惰性删除（后台线程释放内存）
lazyfree-lazy-expire	no	键过期时是否使用惰性删除
lazyfree-lazy-server-del	no	服务器内部删除键（如 RENAME 覆盖）时是否使用惰性删除
replica-lazy-flush	no	从节点全量同步时是否惰性清空数据库
lazyfree-lazy-user-del	no	用户执行 DEL 命令时是否使用惰性删除（等价于 UNLINK）
lazyfree-lazy-user-flush	no	用户执行 FLUSH 命令未指定同步 / 异步时的默认行为
oom-score-adj	no	是否控制内核 OOM killer 的进程评分，`yes/relative` 为相对调整，`absolute` 为绝对设置
oom-score-adj-values	0 200 800	OOM 评分调整值，分别对应主节点、从节点、后台子进程
disable-thp	yes	是否禁用内核透明大页（THP），避免 fork 和 CoW 的性能问题
appendonly	no	是否启用 AOF 持久化
appendfilename	"appendonly.aof"	AOF 文件名称
appendfsync	everysec	AOF 刷盘策略：`no`（依赖系统）、`always`（每次写入）、`everysec`（每秒一次，默认）
no-appendfsync-on-rewrite	no	AOF 重写期间是否禁用主进程的 fsync，可能增加数据丢失风险但提升性能
auto-aof-rewrite-percentage	100	AOF 自动重写的触发百分比（当前大小较上次重写增长比例）
auto-aof-rewrite-min-size	64mb	AOF 自动重写的最小文件大小

aof-load-truncated	yes	启动时是否加载被截断的 AOF 文件，no 则会报错退出
aof-use-rdb-preamble	yes	AOF 重写时是否使用 RDB 前缀（提升重写和恢复速度）
lua-time-limit	5000	Lua 脚本的最大执行时间（毫秒），超时后仅允许 SCRIPT KILL 和 SHUTDOWN NOSAVE
cluster-enabled	no	是否启用集群模式
cluster-config-file	nodes-6379.conf	集群配置文件路径，由节点自动维护
cluster-node-timeout	15000	集群节点超时时间（毫秒），用于判断节点故障
cluster-replica-validity-factor	10	从节点故障转移的有效性因子，影响故障转移条件
cluster-migration-barrier	1	从节点迁移的屏障（原主节点需保留的从节点数）
cluster-allow-replica-migration	yes	是否允许从节点自动迁移到无从节点的主节点
cluster-require-full-coverage	yes	集群是否需要所有哈希槽都被覆盖才接受请求
cluster-replica-no-failover	no	从节点是否禁止自动故障转移（仍可手动触发）
cluster-allow-reads-when-down	no	集群不可用时，节点是否仍响应其负责槽的读请求
cluster-announce-ip	空	集群节点对外公布的 IP 地址（用于 NAT/Docker 场景）
cluster-announce-port	空	集群节点对外公布的客户端端口
cluster-announce-tls-port	空	集群节点对外公布的 TLS 客户端端口
cluster-announce-bus-port	空	集群节点对外公布的总线端口（默认客户端端口 + 10000）
slowlog-log-slower-than	10000	慢日志记录的阈值（微秒），负数禁用，0 记录所有命令

slowlog-max-len	128	慢日志的最大条目数，超出后淘汰旧条目
latency-monitor-threshold	0	延迟监控的阈值（毫秒），0 禁用监控
notify-keyspace-events	""	键空间事件通知配置，由多个字符组成（如 "Ex" 表示过期事件）
gopher-enabled	no	是否启用 Gopher 协议支持
hash-max-ziplist-entries	512	哈希表使用压缩列表编码的最大条目数
hash-max-ziplist-value	64	哈希表使用压缩列表编码的最大值大小（字节）
list-max-ziplist-size	-2	列表使用压缩列表编码的节点大小限制，-1 至 - 5 对应 4Kb 至 64Kb，正数为条目数
list-compress-depth	0	列表的压缩深度，0 禁用压缩，正数表示首尾不压缩的节点数
set-max-intset-entries	512	集合使用整数集编码的最大条目数（元素为 64 位有符号整数）
zset-max-ziplist-entries	128	有序集合使用压缩列表编码的最大条目数
zset-max-ziplist-value	64	有序集合使用压缩列表编码的最大值大小（字节）
hll-sparse-max-bytes	3000	HyperLogLog 使用稀疏表示的最大字节数（含 16 字节头）
stream-node-max-bytes	4096	流数据结构中单个节点的最大字节数
stream-node-max-entries	100	流数据结构中单个节点的最大条目数
activerehashing	yes	是否启用主动重哈希，定期释放哈希表内存
client-output-buffer-limit normal	0 0 0	普通客户端的输出缓冲区限制（硬限制、软限制、软限制持续时间）
client-output-buffer-limit replica	256mb 64mb 60	从节点客户端的输出缓冲区限制

client-output-buffer-limit pubsub	32mb 8mb 60	Pub/Sub 客户端的输出缓冲区限制
client-query-buffer-limit	1gb	客户端查询缓冲区的最大大小
proto-max-bulk-len	512mb	协议中批量请求的最大大小（字节），需≥1mb
hz	10	后台任务执行频率（赫兹），1-500，越高响应性越好但 CPU 消耗增加
dynamic-hz	yes	是否启用动态 Hz，客户端多时自动提高频率
aof-rewrite-incremental-fsync	yes	AOF 重写时是否每生成 32MB 数据执行一次 fsync，减少延迟峰值
rdb-save-incremental-fsync	yes	RDB 保存时是否每生成 32MB 数据执行一次 fsync，减少延迟峰值
lfu-log-factor	10	LFU 淘汰算法的计数器对数因子，影响计数器增长速度
lfu-decay-time	1	LFU 计数器的衰减时间（分钟），超过此时长计数器减半
activedefrag	no	是否启用主动内存碎片整理（需 Jemalloc 支持）
active-defrag-ignore-bytes	100mb	触发主动碎片整理的最小碎片浪费字节数
active-defrag-threshold-lower	10	触发主动碎片整理的最小碎片率（百分比）
active-defrag-threshold-upper	100	触发最大力度碎片整理的碎片率（百分比）
active-defrag-cycle-min	1	碎片整理的最小 CPU 占比（百分比），达到下限时使用
active-defrag-cycle-max	25	碎片整理的最大 CPU 占比（百分比），达到上限时使用
active-defrag-max-scan-fields	1000	每次主字典扫描处理的最大字段数（适用于集合、哈希等结构）
jemalloc-bg-thread	yes	是否启用 Jemalloc 的后台线程进行内存释放

server_cpulist	空	服务器 / IO 线程的 CPU 亲和性列表（如 "0-7:2" 表示 0、2、4、6 核）
bio_cpulist	空	后台 I/O 线程的 CPU 亲和性列表
aof_rewrite_cpulist	空	AOF 重写子进程的 CPU 亲和性列表
bgsave_cpulist	空	BGSAVE 子进程的 CPU 亲和性列表
ignore-warnings	空	需要忽略的警告列表（空格分隔），如 "ARM64-COW-BUG"

7.1.2 哨兵配置配置文件

哨兵配置文件为：`sentinel.yaml`

参数名称	默认值	备注
bind	127.0.0.1	监听的网络地址，默认仅本地可访问；需显式配置（如bind 127.0.0.1 192.168.1.1）或禁用保护模式
port	26379	哨兵进程监听的 TCP 端口
protected-mode	yes	保护模式，未配置bind且无密码时，仅允许本地连接；公网环境需设置为no并配合防火墙
unixsocket	空	Unix 域套接字路径，未指定则不监听 Unix 套接字
unixsocketperm	700	Unix 域套接字的权限
daemonize	no	是否以守护进程模式运行，yes则后台运行并生成 pid 文件
pidfile	/var/run/amdc-sentinel.pid	pid 文件路径，守护进程模式下默认生成
logfile	""	日志文件路径，空字符串表示输出到标准输出；守护进程模式下若为空则日志会被丢弃
sentinel announce-ip	空	用于 NAT 环境，指定哨兵对外公布的 IP 地址（替代自动检测的本地 IP）
sentinel announce-port	空	用于 NAT 环境，指定哨兵对外公布的端口（替代port配置）
dir	/.	工作目录，哨兵会在此目录执行持久化等操作
maxclients	10000	最大并发客户端连接数，受系统文件描述符限制

user	""	<用户名> <ACL规则...>
sentinel monitor mymaster	127.0.0.1 6379 2	监控的主节点配置，格式为<主节点名称> <端口>；quorum为判断主节点客观下线的最小哨兵数量
sentinel auth-pass	空	主节点和从节点的认证密码，若实例配置了requirepass则必须设置
sentinel auth-user	空	用于哨兵与实例的认证（配合auth-pass使用）
sentinel down-after-milliseconds mymaster	30000	主节点（或从节点、哨兵）被判定为主观下线（S_DOWN） 的超时时间（毫秒），默认 30 秒
acllog-max-len	128	ACL 日志的最大条目数，用于记录被 ACL 阻止的命令和认证事件
aclfile	空	外部 ACL 用户文件路径，与配置文件内用户配置不可混用
requirepass	空	哨兵自身的认证密码，所有哨兵需配置相同密码以相互通信
sentinel sentinel-user	空	哨兵之间通信使用的用户名
sentinel sentinel-pass	空	哨兵之间通信使用的密码，配合sentinel-user使用
sentinel parallel-syncs mymaster	1	故障转移时，同时重新配置为新主节点从节点的数量；值越小对客户端影响越小
sentinel failover-timeout mymaster	180000	故障转移超时时间（毫秒），默认 3 分钟；影响重试间隔、从节点重配置等多个场景
sentinel notification-script	空	哨兵事件通知脚本路径（如警告级事件），脚本接收事件类型和描述作为参数
sentinel client-reconfig-script	空	客户端重配置脚本路径，故障转移后触发，用于通知客户端主节点地址变更
sentinel deny-scripts-reconfig	yes	是否禁止运行时通过SENTINEL SET修改脚本配置，默认禁止以避免安全风险
sentinel rename-command	空	重命名命令（如主节点将CONFIG改为GUESSME时，哨兵需同步修改）
sentinel resolve-hostnames	no	是否启用主机名支持，默认仅支持 IP；启用需确保 DNS 配置正常

sentinel announce-hostnames	no	启用主机名宣告，与resolve-hostnames配合使用，默认使用 IP 宣告

7.1.3 授权认证中心配置

使用认证中心时，填写acls.properties；需要提供apusic license认证中心地址，认证中心地址必须是ip:port格式，多个地址用，隔开。

注：该配置在旧版的amdc.yaml中存在，新版本中已移除。但仍就兼容旧版本。

配置项	默认值	说明
apusic_acls_enable	false	是否开启apusic license认证中心， 如果开启则使用true，否则是false
apusic_acls_authUrls	""	认证中心地址，必须是ip:port格式， 多个地址用，隔开
apusic_acls_ns	"public"	命名空间
apusic_acls_tenant	""	租户名称

8 产品工具使用说明

8.1 amdc-check-rdb工具使用手册

9 常见问题处理

amdc-check-rdb是AMDC的RDB文件校验与修复工具，用于检查RDB快照文件的完整性、合法性，排查文件损坏问题，并可对部分损坏的RDB文件进行修复，避免因RDB文件损坏导致AMDC启动失败或数据丢失。

核心作用：

- 校验RDB文件是否完整、无损坏，检测文件格式是否符合AMDC规范；
- 修复因意外中断（如服务器宕机、断电）导致损坏的RDB文件；
- 查看RDB文件的基础信息（如版本、数据量、过期时间等）。

9.0.1 工具位置与运行前提

9.0.1.1 工具位置

AMDC安装完成后，amdc-check-rdb工具位于AMDC安装目录文件夹下：

```
Linux: /amdc/amdc-check-rdb（默认路径，若自定义安装需对应调整）
```

9.0.1.2 运行前提

- 无需启动AMDC服务，可直接在命令行运行；
- 需拥有RDB文件的读取权限（若文件权限不足，需使用chmod修改权限，Linux）；
- RDB文件路径需正确（绝对路径或相对路径均可，相对路径以当前终端所在目录为准）。

9.0.2 基本语法

amdc-check-rdb的语法简洁，核心仅需指定待处理的RDB文件路径，可选参数用于修复或查看详细信息：

```
#基础语法（校验RDB文件）
amdc-check-rdb [options] /path/to/dump.rdb

#常用简写（无选项时，默认仅校验文件完整性）
amdc-check-rdb dump.rdb
```

9.0.3 核心选项与参数说明

amdc-check-rdb 选项较少，重点关注修复和调试相关选项，具体如下：

选项	说明	使用场景
--fix	修复损坏的RDB 文件，将可恢复的数据保留，丢弃无法修复的部分	RDB 文件损坏，AMDC启动失败时
--help（简写-h）	显示工具的帮助信息，包括所有选项说明和基本语法	忘记命令语法时快速查询

9.0.4 常见操作示例

9.0.4.1 校验RDB文件完整性

无需选项，直接指定RDB文件路径，工具会自动校验文件是否完整、格式是否正确：

```
#校验RDB文件
amdc-check-rdb/path/to/dump.rdb
```

正常输出（文件完整）：

```
[offset 0] Checking RDB file dump.rdb
[offset 29] AUX FIELD amdc-ver = '2.0.4'
[offse 46] AUX FIELD amdc-bits = '64'
[offset 58] AUX FIELD ctime = '1769518212'
[offset 73] AUX FIELD used-mem = '2442688'
[offset 91] AUX FIELD repl-stream-db = '0'
[offset 141] AUX FIELD repl-id =
'069cb616c524d6ec885da6218b669f1d9c970125'
[offset 156] AUX FIELD repl-offset ='0'
[offset 172] AUX FIELD aof-preamblle ='0'
[offset 181] Checksum OK
[offset 181] \o/ RDB looks OK! \o/
[info]0 keys read
[info]0 expires
[info] 0 already expired
```

异常输出（文件损坏）：

```
[offset 0] Checking RDB file dump.rdb
[offset 21] AUX FIELD ctime = '1768265496'
[offet 42] AUX FIELD used-mem = '4275241848'
[offset 60] AUX FIELD repl-stream-db = '0'
[offset 11 ] AUX FIELD repl-id =
'f0a6b7b58edlece135f1ba84bcc8add612e8b347'
[offset 135] AUX FIELD repl-offset = '67232446953'
[offset 137] Selecting DB ID 0
--- RDB ERROR DETECTED ---
[offset 73027] Internal error in RDB reading offset 0, function at
rdb.c:1936 -> Intset integrity check failed.
[additional info] While doing: read-object-value
[additional info] Reading key
'jszt.894679007767823360.wf.isallownextparticipant.crrccardispatch.audi
[additional info] Reading type 11 (set-intset)
[info] 159 keys read
[info] 158 expires
[info] 157 already expired
692:C 02 Feb 2026 10:49:23.182 # Terminating server after rdb file
reading failure.
```

9.0.4.2 修复损坏的RDB文件

当RDB文件损坏时，使用--fix选项进行修复，修复后dump文件名为 dump-<进程号>.rdb：

```
#修复损坏的RDB文件
amdc-check-rdb --fix /path/to/dump.rdb
```

修复成功输出：

```
[offset 0] Checking RDB file --fix
[offset 21] AUX FIELD ctime = '1768265496'
[offset 42] AUX FIELD used-mem = '4275241848'
[offset 60] AUX FIELD repl-stream-db = '0'
[offset 10] AUX FIELD repl-id =
'f0a6b7b58edlece135f1ba84bcc8add612e8b347'
```

```
[offset 135] AUX FIELD repl-offsse = '67232446953'
[offset 137] Selecting DB ID 0
--- RDB ERROR DETECTED---
[offset 73027] Internal error in RDB reading offset 0, function at
rdb.c:1936 -> Intset integrity check failed.
[additional info] While doing: read-object-value
[additional info] Reading key
'jszt.894679007767823360.wf.isallownextparticipant.crrccardispatch.audi
[additional info] Reading type 11 (set-intset)
[info] 159 keys read
[info] 158 expires
[info] 157 already expired
--- RDB ERROR DETECTED---
[offset 178543] Internal error in RDB reading offset 0, function at
rdb.c:1936 -> Intset integrity check failed.
[additional info] While doing: read-object-value
[additional info] Reading key
'jszt.894679007767823360.wf.isallownextparticipant.crrclinefeedback.sub
[additional info] Reading type 11 (set-intset)
[info] 392 keys read
[info] 390 expires
[info] 388 already expired
--- RDB ERROR DETECTED---
[offset 260382] Internal error in RDB reading offset 0, function at
rdb.c:1936 -> Intset integrity check failed.
[additional info] While doing: read-object-value
[additional info] Reading key
'jszt.894679007767823360.wf.isallownextparticipant.ebruleexeccase.save'
[additional info] Reading type 11(set-intset)
[info] 571 keys read
[info] 568 expires
[info] 564 already expired
--- RDB ERROR DETECTED---
[offset 136547728] Invalid object type: 111
[additional info] While doing: read-type
[info] 114982 keys read
```

```
[info] 114589 expires
[info] 113699 already expired
[offset 136547728] \o/ RDB partially fixed: dump-790.rdb! \o/
[info] 114982 / 406455 keys fixed
```

9.0.5 注意事项

- 修复文件前是否需要备份原RDB文件：--fix选项不会直接覆盖原文件，修复后dump文件名为 dump-<进程号>.rdb，安全起见操作前可以备份RDB文件；
- 仅能修复部分损坏的文件：若RDB文件头部信息损坏、校验和严重错误，工具可能无法修复，此时需依赖备份文件；
- 修复后可能丢失部分数据：对于RDB文件中格式异常的键值对，工具会尝试跳过并丢弃该数据；若检测到无法修复的数据片段，则会停止扫描文件剩余内容，因此工具可能会仅完成了文件局部数据的修复；
- 工具版本需与AMDC版本匹配：不同版本的RDB文件格式可能有差异，使用与AMDC服务同版本的amdc-check-rdb工具，避免校验或修复失败。

9.1 amdc-check-aof工具使用手册

9.1.1 工具简介

amdc-check-aof是AOF(Append Only File）文件校验与修复工具，用于检查AOF文件的完整性、语法正确性，修复因意外中断导致的AOF文件损坏，确保AMDC启动时能正常加载AOF文件恢复数据。

核心作用：

- 校验AOF文件的语法正确性、完整性，检测文件是否存在损坏；
- 修复因服务器宕机、断电等意外导致的AOF文件截断、语法错误；
- 移除AOF文件中无效的命令，确保AMDC能正常加载。

9.1.2 工具位置与运行前提

9.1.2.1 工具位置

与amdc-check-rdb位于同一目录，即AMDC的安装目录文件夹下：

```
Linux: /amdc/amdc-check-aof
```

9.1.2.2 运行前提

无需启动AMDC服务，可直接在命令行运行；

- 拥有AOF文件的读取、写入权限（修复时需写入文件）；
- AOF文件路径正确，若使用相对路径，需确保终端当前目录与文件路径匹配。

9.1.3 基本语法

amdc-check-aof的语法与amdc-check-rdb类似，核心指定AOF文件路径，可选参数用于修复、调试：

```
#基础语法（校验AOF文件）
amdc-check-aof [options]
/path/to/appendonlydir/appendonly.aof.manifest

#常用简写（无选项时，默认校验文件完整性）
amdc-check-aof appendonlydir/appendonlydir/appendonly.aof.manifest
```

9.1.4 核心选项与参数说明

amdc-check-aof的选项重点关注修复、强制修复和详细输出，具体如下：

选项	说明	使用场景
`--fix`	修复损坏的AOF 文件，移除无效命令、截断错误部分，修复后覆盖原文件（建议先备份）	AOF 文件损坏，AMDC启动失败
`--truncate-to-timestamp`	精准裁剪到指定时间戳，保留该时间点前所有数据	已知服务异常时间点、需精准保留指定时间前的数据
`--help`（简写 `-h`）	显示工具帮助信息，包括所有选项和语法	忘记命令时快速查询

9.1.5 常见操作示例

9.1.5.1 校验AOF文件完整性

直接指定AOF文件路径，工具自动校验文件语法和完整性：

```
#校验AOF文件
amdc-check-aof /path/to/appendonlydir/appendonly.aof.manifest
```

正常输出aof-use-rdb-preamble no（文件完整）：

```
Start checking Multi Part AOF
Start to check BASE AOF (RESP format).
AOF analyzed:
filename=appendonly.aof.1.base.aof,size=127,ok_up_to=127,ok_up_to_line=
BASE AOF appendonly.aof.1.base.aof is valid
Start to check INCR files.
INCR AOF appendonly.aof.1.incr.aof is empty
All AOF files and manifest are valid
```

正常输出aof-use-rdb-preamble yes（文件完整）：

```
Start checking Multi Part AOF
Start to check BASE AOF (RDB format).
[offset 0] Checking RDB file appendonlydir/appendonly.aof.1.base.rdb
[offset 25] AUX FIELD amdc-ver = '2.0.4'
[offset 38] AUX FIELD amdc-bits = '64'
[offset 50] AUX FIELD ctime = '1770690624'
[offset 65] AUX FIELD used-mem = '836208'
[offset 81] AUX FIELD aof-preamble = '1'
[offset 83] Selecting DB ID 0
[offset 151] Checksum OK
[offset 151] \o/ RDB looks OK! \o/
[info] 2 keys read
[info] 0 expires
[info] 0 already expired
RDB preamble is OK, proceeding with AOF tail...
AOF analyzed: filename=appendonly.aof.1.base.rdb, size=151,
ok_up_to=151, ok_up_to_line=1, diff=0
BASE AOF appendonly.aof.1.base.rdb is valid
Start to check INCR files.
INCR AOF appendonly.aof.1.incr.aof is empty
All AOF files and manifest are valid
```

异常输出 aof-use-rdb-preamble no（文件损坏）：

```
Start checking Multi Part AOF
Start to check BASE AOF (RESP format).
AOF appendonly.aof.1.base.aof format error
AOF analyzed:
filename=appendonly.aof.1.base.aof,size=194,ok_up_to=127,ok_up_to_line=
AOF appendonly.aof.1.base.aof is not valid. Use the --fix option to
try fixing it.
```

异常输出 aof-use-rdb-preamble yes（文件损坏）：

```
Start checking Multi Part AOF
Start to check BASE AOF (RDB format).
[offset 0] Checking RDB file appendonlydir/appendonly.aof.1.base.rdb
[offset 25] AUX FIELD amdc-ver = '2.0.4'
[offset 38] AUX FIELD amdc-bits = '64'
[offset 50] AUX FIELD ctime = '1770690624'
[offset 65] AUX FIELD used-mem = '836208'
[offset 81] AUX FIELD aof-preamble = '1'
[offset 83] Selecting DB ID 0
[offset 151] Checksum OK
[offset 151] \o/ RDB looks OK! \o/
[info] 2 keys read
[info] 0 expires
[info] 0 already expired
RDB preamble is OK, proceeding with AOF tail...
AOF appendonly.aof.1.base.rdb format error
AOF analyzed:
filename=appendonly.aof.1.base.rdb,size=167,ok_up_to=151,ok_up_to_line=
AOF appendonly.aof.1.base.rdb is not valid. Use the --fix option to
try fixing it.
```

9.1.5.2 修复损坏的AOF文件

AOF文件损坏时，优先使用--fix选项修复，修复前务必备份原文件:

```
#1. 备份原AOF文件（Linux）
cp /path/to/appendonlydir /path/to/appendonlydir.bak

#2. 常规修复
amdc-check-aof --fix
/path/to/appendonlydir/appendonly.aof.1.base.aof
```

修复成功输出 aof-use-rdb-preamble no（文件损坏）：

```
Start checking Old-Style AOF
AOF appendonlydir/appendonly.aof.1.base.aof format error
AOF analyzed:
filename=appendonlydir/appendonly.aof.1.base.aof,size=146,ok_up_to=127,
This will shrink the AOF appendonlydir/appendonly.aof.1.base.aof
from 146 bytes, with 19 bytes, to 127 bytes
Continue? [y/N]: y
Successfully truncated AOF appendonlydir/appendonly.aof.1.base.aof
```

修复成功输出 aof-use-rdb-preamble yes（文件损坏）：

```
Start checking Old-Style AOF
[offset 0] Checking RDB file appendonlydir/appendonly.aof.1.base.rdb
[offset 25] AUX FIELD amdc-ver = '2.0.4'
[offset 38] AUX FIELD amdc-bits = '64'
[offset 50] AUX FIELD ctime = '1770690624'
[offset 65] AUX FIELD used-mem = '836208'
[offset 81] AUX FIELD aof-preamble = '1'
[offset 83] Selecting DB ID 0
[offset 151] Checksum OK
[offset 151] \o/ RDB looks OK! \o/
[info] 2 keys read
[info] 0 expires
[info] 0 already expired
RDB preamble is OK, proceeding with AOF tail...
AOF appendonlydir/appendonly.aof.1.base.rdb format error
```

```
AOF analyzed:
filename=appendonlydir/appendonly.aof.1.base.rdb,size=167,ok_up_to=151,
This will shrink the AOF appendonlydir/appendonly.aof.1.base.rdb
from 167 bytes, with 16 bytes, to 151 bytes
Continue? [y/N]: y
Successfully truncated AOF appendonlydir/appendonly.aof.1.base.rdb
```

9.1.6 注意事项

- 备份优先：修复AOF文件前，必须备份原文件，避免修复失败导致数据彻底丢失；
- 修复原理：工具通过截断无效命令、修正语法错误来修复文件，会丢弃无法识别的命令，修复后需检查数据完整性；
- 与AOF重写的区别：amdc-check-aof用于修复损坏文件，而AMDC的AOF重写（BGREWRITEAOF）用于压缩AOF文件，二者功能不同；
- appendonlydir 目录下可通过 --fix 参数修复指定文件，支持的文件包括：appendonly.aof.x.base.rdb、appendonly.aof.x.base.aof、appendonly.aof.x.incr.aof 为可选指定。

9.2 amdc-benchmark 工具使用手册

9.2.1 工具简介

amdc-benchmark是AMDC自带的性能测试工具，用于模拟多客户端并发访问AMDC服务，测试AMDC在不同场景下的性能指标，包括每秒请求数（QPS）、响应时间（延迟）、并发连接数等，帮助运维人员和开发人员评估AMDC服务的承载能力、优化配置参数，确保AMDC服务在生产环境中稳定运行。

核心作用：

- 测试AMDC服务在不同并发连接数、不同命令类型下的吞吐量（QPS）；
- 评估AMDC服务的响应延迟（最小延迟、最大延迟、平均延迟），排查性能瓶颈；
- 模拟不同数据量、不同命令组合的访问场景，验证AMDC配置（如内存、线程数）的合理性；
- 对比AMDC在不同环境（如单机、集群）、不同版本下的性能差异，为部署方案提供依据。

提示：amdc-benchmark属于压力测试工具，测试时会占用AMDC服务的CPU、内存及网络资源，生产环境需谨慎使用，建议在测试环境或业务低峰期执行，避免影响线上业务。

9.2.2 工具位置与运行前提

9.2.2.1 工具位置

与前三个工具位于同一目录，即AMDC安装文件夹下，无需额外安装，随AMDC一起部署：

```
Linux: /amdc/amdc-benchmark
```

提示：若将AMDC安装目录添加到系统环境变量，可直接在终端输入amdc-benchmark 运行，无需输入完整路径。

9.2.2.2 运行前提

- 必须启动AMDC服务：amdc-benchmark 需连接AMDC服务才能执行测试，确保amdc-server 已正常运行；
- 网络通畅：本地测试需确保 127.0.0.1:6359（默认端口）可访问；远程测试需确保AMDC服务端口（默认6359）开放，且配置允许远程连接；
- 权限充足：运行amdc-benchmark的用户需拥有终端执行权限，若AMDC服务设置密码，测试时需指定密码；
- 资源预留：测试前预留足够的CPU、内存资源，避免测试环境自身资源不足，影响测试结果准确性。

9.2.3 基本语法

amdc-benchmark的语法核心是指定连接参数和测试参数，通过选项控制测试场景，基础语法如下：

```
#基础语法（连接AMDC服务 + 执行测试）
amdc-benchmark [连接选项] [测试选项]

#常用简写（本地默认连接，无额外选项，执行默认测试）
amdc-benchmark
```

说明：

- 连接选项：用于指定AMDC服务的主机、端口、密码等，与amdc-cli的连接选项基本一致；
- 测试选项：用于控制并发数、测试命令、测试时长、数据量等，是amdc-benchmark 的核心选项；
- 默认测试：若不指定任何选项，工具会以10个并发连接、测试16种常用AMDC命令，每种命令执行10000次，最终输出性能报告。

9.2.4 核心选项与参数说明

amdc-benchmark的选项分为「连接选项」和「测试选项」两类，其中测试选项是重点，具体如下：

9.2.4.1 连接选项

选项	取值类型	说明	默认值
-h <host>	字符串（IP / 域名）	指定AMDC 服务的主机地址(IP)，本地测试可省略，远程测试必须指定	127.0.0.1
-p <port>	整数（端口号）	指定AMDC 服务的端口号	6359
-s <socket>	字符串（套接字路径）	指定AMDC 服务的密码（若服务设置了 requirepass 配置）	无（默认无密码）
-a <password>	字符串	指定连接超时时间（秒），避免测试时连接阻塞	无（默认不超时）

9.2.4.2 核心测试选项

选项	取值类型	说明	默认值
-c <clients>	整数	指定并发连接的客户端数量，模拟多用户同时访问	50
-n <requests>	整数	指定每种测试命令的总请求数，所有客户端共同完成	100000
-d <size>	整数（字节）	指定SET/GET命令的value 数据大小，模拟真实业务的数值长度	3
--dbnum <db>	整数（数据库编号）	指定测试使用的AMDC 数据库（SELECT 命令），对应AMDC的db0~db15	0
-k <boolean>	布尔值（1/0）	连接保活策略：1 = 保持连接（长连接）、0 = 每次请求重连（短连接），模拟真实业务连接模式	1
-r <keyspacelen>	整数	为SET/GET/INCR等命令使用随机键名、SADD用随机值、ZADD 用随机成员和分数；自动将参数中的randint 替换为 0 ~ keyspacelen-1 的12位随机数，避免热点键，贴合真实场景	无
-P <numreq>	整数	开启管道请求，每次管道打包 < numreq > 个请求发送，模拟业务批量操作（提升吞吐）	无
--help	无	显示工具帮助信息，包括所有选项和基本语法	忘记命令语法时快速查询
--version	无	显示amdc-benchmark 工具版本，与AMDC 服务版本一致	确认版本兼容性

9.2.4.3 高级扩展参数（多线程 / 集群 / 特殊测试模式）

选项	取值类型	说明	默认值
`--threads <num>`	整数	开启多线程模式运行压测工具，利用多核CPU 提升压测端性能（避免压测端成为瓶颈）	`4`
`--cluster`	无（开关参数）	开启集群模式，适配AMDC Cluster 集群压测，自动识别槽位、分发请求	关闭
`-l`	无（开关参数）	循环测试模式：无限运行测试，直到手动按Ctrl+C停止，用于长时间稳定性压测	关闭
`-I`	无（开关参数）	空闲模式：仅打开-c 指定的N 个空闲连接，不发送任何请求，用于测试AMDC 的空闲连接承载能力	关闭

9.2.5 常见操作示例（核心重点）

以下示例覆盖日常最常用的测试场景，结合实际业务需求，适配AMDC V2.0.3和V2.0.4版本，可直接参考使用：

9.2.5.1 本地默认测试

不指定任何选项，执行默认测试：50个并发客户端、测试16种默认命令，每种命令执行100000次，输出完整性能报告：

bash

```
amdc-benchmark
```

默认测试命令包括：SET、GET、INCR、LPUSH、LPOP、SADD、SPOP等常用命令，适合快速了解本地AMDC的基础性能。

9.2.5.2 指定并发数和请求数测试

模拟真实并发场景，指定并发客户端数量和每种命令的请求数，是最常用的测试方式：

```
#示例：50个并发客户端，每种命令执行100000次，测试本地AMDC性能
amdc-benchmark -c 50 -n 100000

#示例：100个并发客户端，每种命令执行50000次，测试远程AMDC（带密码）
amdc-benchmark -h 192.168.1.100 -p 6359 -a 123456 -c 100 -n 50000
```

9.2.5.3 针对性测试核心命令

实际业务中通常只使用少数几种AMDC命令，通过-t选项指定命令，避免无用测试，提高效率：

```
#示例：测试SET和GET命令，50个并发，每种命令执行100000次
amdc-benchmark -t SET,GET -c 50 -n 100000

#示例：测试哈希命令（HSET、HGET),30个并发，值大小为1KB
amdc-benchmark -t HSET,HGET -c 30 -n 50000 -d 1024
```

9.2.5.4 按时间进行压力测试对性测试核心命令

模拟长时间稳定访问，指定测试持续时间，适合测试Redis长时间运行的稳定性：

```
#示例：循环测试（手动停止），测试SET命令，60个并发，值大小为512字节
amdc-benchmark -t SET -c 60 -1 -d 512
```

9.2.5.5 简化输出，快速查看QPS

无需详细延迟信息，仅查看每种命令的QPS，适合快速对比性能差异：

```
#示例：默认测试，简化输出QPS
amdc-benchmark -q

#示例：测试远程AMDC的SET、GET命令，简化输出
amdc-benchmark -h 192.168.1.100 -a 123456 -t SET,GET -c 50 -n 100000
-q
```

9.2.5.6 测试结果解读（关键指标）

执行测试后，工具会输出详细报告，核心关注以下4个指标，以SET命令为例：

```
SET: 100000 requests completed in 0.89 seconds
50 parallel clients
3 bytes payload
keep alive:1
host configuration "save": 3600 1 300 100 60 10000
host configuration"appendonly": no
multi-thread:no
Latencyby percentile distribution:
```

```
0.000% <= 0.103 ms (cumulative count 123)
50.000% <= 0.215 ms (cumulative count 50123)
90.000% <= 0.327 ms (cumulative count 90123)
99.000% <= 0.439 ms (cumulative count 99012)
99.900% <= 0.551 ms (cumulative count 99901)
99.990% <= 0.663 ms (cumulative count 99990)
99.999% <= 0.775 ms (cumulative count 99999)
100.000% <= 0.887 ms (cumulative count 100000)
Cumulative distribution of latencies:
0.103 ms: 123 requests (0.12%)
0.215 ms: 50123 requests (50.12%)
0.327 ms: 90123 requests (90.12%)
0.439 ms: 99012 requests (99.01%)
0.551 ms: 99901 requests (99.90%)
0.663 ms: 99990 requests (99.99%)
0.775 ms: 99999 requests (99.999%)
0.887 ms: 100000 requests (100.00%)

Summary:
throughput summary: 112359.55 requests per second
latency summary (msec):
| avg   | min   | p50   | p95   | p99   | max   |
|-------|-------|-------|-------|-------|-------|
| 0.220 | 0.080 | 0.215 | 0.383 | 0.439 | 0.887 |
```

核心指标解读：

- throughput summary:QPS（每秒请求数），核心性能指标，数值越高越好（示例中为112359.55 QPS）；
- avg / min / max：平均延迟、最小延迟、最大延迟（毫秒），数值越低越好；
- p50 / p95 / p99：延迟分位数，代表50%/95%/99%的请求延迟不超过该数值（示例中99%的请求延迟 ≤ 0.439ms），分位数更能反映真实性能；
- requests completed：完成的总请求数，与-n指定的请求数一致，说明测试正常。

9.2.6 注意事项

- 生产环境谨慎使用：amdc-benchmark会占用AMDC大量CPU、内存和网络资源，可能导致线上业务卡顿、超时，建议仅在测试环境或业务低峰期执行，且测试后及时停止；
- 测试环境与生产环境一致：测试时尽量保证AMDC配置（内存、线程数、持久化方式）、服务器硬件（CPU、内存）与生产环境一致，否则测试结果无参考价值；
- 避免单一测试场景：真实业务场景复杂，建议结合并发数、数据量、命令类型多场景测试，全面评估性能；
- 排除干扰因素：测试时关闭AMDC持久化(appendonly no、save "")，避免持久化操作影响测试结果；同时关闭服务器上其他占用资源的进程；
- 密码安全：使用-a选项输入密码时，密码会明文显示在终端历史记录中，建议测试完成后清理终端历史，或通过其他方式隐藏密码；
- 合理设置测试参数：并发数不宜过高（超出服务器承载能力），请求数/测试时长不宜过短（避免测试结果波动过大），建议多次测试取平均值。

9.3 amdc-conf-conv工具使用手册

9.3.1 工具简介

amdc-conf-conv是AMDC专属的配置文件格式转换工具，核心用于将AMDC V2.0.2及以下版本AMDC的配置文件转换为AMDC V2.0.4版本AMDC的配置文件，解决不同版本AMDC配置文件格式不兼容的问题，实现AMDC V2.0.2及以下版本AMDC向AMDC V2.0.4版本AMDC的平滑迁移，无需手动修改配置项，提升版本迁移的效率和准确性。

核心作用：

- 一键转换AMDC V2.0.2及以下版本AMDC的YAML配置文件为AMDC V2.0.4版本AMDC兼容的YAML配置文件，自动适配配置项格式、命名规范；
- 支持服务端（server）、哨兵（sentinel）两种配置类型的精准转换，分别匹配对应版本的功能配置；
- 保留原配置文件中的所有有效配置项，确保转换后AMDC V2.0.4版本AMDC可直接加载使用；
- 规避手动转换配置的语法错误、项缺失问题，降低版本迁移的配置风险。

配置转换工具执行逻辑为「先读取--c-yaml模版文件的原有结构与基础配置->提取--go-yaml指向的AMDC V2.0.2及以下版本配置文件的有效配置项->将模版文件中对应配置项替换为AMDC V2.0.2及以下版本的配置值->最终覆盖写入--c-yaml指向的文件」，并非直接将AMDC V2.0.2及以下版本配置生成新的AMDC V2.0.4版本配置文件，--c-yaml指向的配置文件一定不能为空文件。

9.3.2 工具位置与运行前提

9.3.2.1 工具位置

与AMDC其他配套工具（如amdc-check-aof、amdc-check-rdb）位于同一目录，随AMDC 安装包一同部署，无需额外安装：

```
Linux: /amdc/amdc-conf-conv
```

9.3.2.2 运行前提

- 无需启动AMDC服务，可直接在命令行运行；
- 拥有AMDC V2.0.2及以下版本配置文件的读取权限，以及目标AMDC V2.0.4版本配置文件的读写权限；
- 确保AMDC V2.0.2及以下/AMDC V2.0.4版本AMDC的配置文件（yaml格式）路径正确、文件完整无损坏，可正常读取。

9.3.3 基本语法

amdc-conf-conv 语法规则固定，需指定配置类型和源或目标配置文件路径，无默认执行逻辑，必须传入完整有效参数才能运行，基础语法如下：

```
#基础语法（指定配置类型+源AMDC V2.0.2及以下版配置+目标C版配置）
amdc-conf-conv [--server|--sentinel] --go-yaml <amdc_go.yaml> --c-yaml <amdc.yaml>
```

9.3.4 核心选项与参数说明

amdc-check-aof的选项重点关注修复、强制修复和详细输出，具体如下：

选项	说明	使用场景
`--server`	指定转换的是AMDC 服务端（server）类型的配置文件；`[--server	--sentinel]` 二选一必传参数，不可省略、不可同时使用
`--sentinel`	指定转换的是AMDC 哨兵（sentinel）类型的配置文件；`[--server	--sentinel]` 二选一必传参数，不可省略、不可同时使用
`--go-yaml`	AMDC V2.0.2及以下版本AMDC 配置文件的完整路径（绝对或相对），指定转换源文件	所有转换场景，必传源文件路径
`--c-yaml`	转换后C 版本AMDC 配置文件的完整路径（绝对或相对），作为替换模版覆盖写入	所有转换场景，必传目标文件路径
`--help`	显示工具帮助信息，包括所有选项和语法	忘记命令时快速查询

9.3.5 常见操作示例

以下示例覆盖server和sentinel两种核心配置类型的转换场景，适配Linux环境，可直接参考使用，建议执行前先备份相关配置文件。

9.3.5.1 转换AMDC V2.0.2及以下版本AMDC服务端配置为AMDC V2.0.4版本

指定--server参数，转换AMDC V2.0.2及以下版本服务端配置文件conf.yaml为AMDC V2.0.4版本服务端配置文件amdc.yaml:

```
#使用绝对路径（推荐，无路径兼容问题）
amdc-conf-conv --server --go-yaml /amdc/conf.yaml --c-yaml
/amdc/amdc.yaml

#使用相对路径（需确保终端当前目录为/amdc）
cd /amdc
amdc-conf-conv --server --go-yaml conf.yaml --c-yaml amdc.yaml
```

转换成功：

```
Skip unsupported configuration item: ReplSlaveIgnoreMaxMemory
Skip unsupported configuration item: PoolSliceSize
Skip unsupported configuration item: PoolSliceNum
Skip unsupported configuration item: PoolNewWorkerNum
Skip unsupported configuration item: AofNoFsyncOnRewrite
Skip unsupported configuration item: GoroutineMaxCpuNum
Skip unsupported configuration item: SlowLogSlowerThan
Skip unsupported configuration item: LuaMaxLocalVarNum
Skip unsupported configuration item: LuaMaxStackSize
Skip unsupported configuration item: ReplServeStaleData
Skip unsupported configuration item: ReplPingSlavePeriod
Skip unsupported configuration item: Ip
Skip unsupported configuration item: Segment
\o/ Config file converted successfully! (conf.yaml -> amdc.yaml)\o/
```

conf.yaml转换前：

```
onf.yaml
Network:
  MaxClients: 10000
  # 连接在空闲超过设置的时间后，服务器会主动关闭连接，当设置为0时，不启用超时机制，timeout单位
  Timeout: 0
  # TCP keepalive，单位为秒，当为0时不设置tcp keepalive
  TcpKeepAlive: 300

General:
  # db数量
  Databases: 16
  # 日志等级，用于过滤输出日志，包括debug, info, warn, error, fatal五个等级
  LogLevel: "info"
  # 日志文件输出目录，当设置为空字符串时，日志文件不会写入磁盘， eg: LogFile: "/tmp/server
  LogFile: "./server-6359.log"
  # license文件位置
  LicensePath: "./license.lic"

MemoryManagement:
  # 最大内存限制，如果maxmemory值为0，表示不做限制，如果数字后没有单位，则默认单位为字节，单位
  # eg: "1gb", "1GB", "1000mb", "1000m", "1000000KB", "1000000kb", "1000000000B"
  MaxMemory: "6GB"
  # 缓存淘汰策略，noeviction, volatile-lru, allkeys-lru, volatile-random, allkeys-r
  MaxMemoryPolicy: "volatile-lru"
  # 每次缓存淘汰时的采样数量
  MaxMemorySamples: 5
  # lfu-log-factor可以调整计数器counter的增长概率，lfu-log-factor越大，counter增长概率越
  # 计算公式为： 1 / (old_value * lfu_log_factor + 1)
  LFULogFactor: 10.000000
  # lfu-decay-time是一个以分钟为单位的数值，可以调整counter的减少速度
  LFUDecayTime: 1.000000
  # 从节点是否忽略maxmemory检查
  ReplSlaveIgnoreMaxMemory: "yes"
  # PoolSliceSize
  PoolSliceSize: 40
  # PoolSliceNum
  PoolSliceNum: 40
  # PoolSliceNum
  PoolNewWorkerNum: 4

SnapShotting:
  # save <seconds> <changes>，指定在多长时间内，有多少次更新操作后，server会将rdb文件写入
  # Save:
  #   - "3600 1"
  #   - "300 10"
  #   - "60 10000"
  # 当 Save: "" 为空时，不启用RDB自动保存
```

amdc.yaml转换前：

mdc1.yaml

```yaml
REPLICATION:
    # 从节点向主节点公布的端口（用于 NAT/Docker 等场景）
    replica-announce-port: 0

KEYS TRACKING:
    # 客户端缓存跟踪表的最大键数，0 表示无限制
    tracking-table-max-keys: 1000000

SECURITY:
    # user: "<user> <acl rules...>"
    user: ""
    # ACL 日志的最大条目数
    acllog-max-len: 128
    # 外部 ACL 用户文件路径，与配置文件内用户配置不可混用
    aclfile: ""
    # 默认用户的密码（等价于 ACL 的 default 用户密码）
    requirepass: ""
    # 新用户默认的 Pub/Sub 频道权限，allchannels（允许所有）、resetchannels（禁止所有）
    acl-pubsub-default: allchannels
    # 命令重命名，eg: rename-command: "flushall ''", rename-command: "flushall no-
    rename-command: ""

CLIENTS:
    # 最大并发客户端连接数，受系统文件描述符限制
    maxclients: 10000

MEMORY MANAGEMENT:
    # 内存使用上限，达到后按 maxmemory-policy 处理
    maxmemory: 0
    # 内存达到上限时的淘汰策略，如 volatile-lru、allkeys-lfu 等
    maxmemory-policy: allkeys-lru
    # LRU/LFU/TTL 算法的采样数，越大越精确但 CPU 消耗越高
    maxmemory-samples: 5
    # 淘汰处理的激进程度，0（最低延迟）到 100（优先处理淘汰）
    maxmemory-eviction-tenacity: 10
    # 从节点是否忽略自身的 maxmemory 配置（默认由主节点处理淘汰）
    replica-ignore-maxmemory: yes
    #主动过期键的扫描力度，1-10，越大越频繁（可能增加 CPU 和延迟）
    active-expire-effort: 1

LAZY FREEING:
    # 内存淘汰时是否使用惰性删除（后台线程释放内存）
    lazyfree-lazy-eviction: no
    # 键过期时是否使用惰性删除
    lazyfree-lazy-expire: no
    # 服务器内部删除键（如 RENAME 覆盖）时是否使用惰性删除
```

amdc.yaml转换后：

mdc.yaml

```yaml
REPLICATION:
    # 从节点向主节点公布的端口（用于 NAT/Docker 等场景）
    replica-announce-port: 0

KEYS TRACKING:
    # 客户端缓存跟踪表的最大键数，0 表示无限制
    tracking-table-max-keys: 1000000

SECURITY:
    # user: "<user> <acl rules...>"
    user: default on nopass ~* &* +@all
    # ACL 日志的最大条目数
    acllog-max-len: 128
    # 外部 ACL 用户文件路径，与配置文件内用户配置不可混用
    aclfile: ""
    # 默认用户的密码（等价于 ACL 的 default 用户密码）
    requirepass: "test123"
    # 新用户默认的 Pub/Sub 频道权限，allchannels（允许所有）、resetchannels（禁止所有）
    acl-pubsub-default: allchannels
    # 命令重命名，eg: rename-command: "flushall ''", rename-command: "flushall no-
    rename-command: ""

CLIENTS:
    # 最大并发客户端连接数，受系统文件描述符限制
    maxclients: 10000

MEMORY MANAGEMENT:
    # 内存使用上限，达到后按 maxmemory-policy 处理
    maxmemory: 6gb
    # 内存达到上限时的淘汰策略，如 volatile-lru、allkeys-lfu 等
    maxmemory-policy: volatile-lru
    # LRU/LFU/TTL 算法的采样数，越大越精确但 CPU 消耗越高
    maxmemory-samples: 5
    # 淘汰处理的激进程度，0（最低延迟）到 100（优先处理淘汰）
    maxmemory-eviction-tenacity: 10
    # 从节点是否忽略自身的 maxmemory 配置（默认由主节点处理淘汰）
    replica-ignore-maxmemory: yes
    #主动过期键的扫描力度，1-10，越大越频繁（可能增加 CPU 和延迟）
    active-expire-effort: 1

LAZY FREEING:
    # 内存淘汰时是否使用惰性删除（后台线程释放内存）
    lazyfree-lazy-eviction: no
    # 键过期时是否使用惰性删除
    lazyfree-lazy-expire: no
    # 服务器内部删除键（如 RENAME 覆盖）时是否使用惰性删除
```

9.3.5.2 转换AMDC V2.0.2及以下版本AMDC哨兵配置为AMDC V2.0.4版本

指定--sentinel参数，转换AMDC V2.0.2及以下版本哨兵配置文件sentinel.yaml为AMDC V2.0.4版本哨兵配置文件sentinel.yaml:

```
#绝对路径示例
amdc-conf-conv --sentinel --go-yaml /amdc/sentinelgo.yaml --c-yaml
/amdc/sentinel.yaml

#相对路径示例（终端当前目录为 / amdc）
amdc-conf-conv --sentinel --go-yaml sentinelgo.yaml --c-yaml
sentinel.yaml
```

修复成功输出：

```
Config file converted successfully! (sentinelgo.yamnl-
>sentinel.yaml)
```

sentinel_go.yaml转换前：

```
Sentinel:
    # 绑定的ip地址，可以绑定多个ip地址，支持ipv4/ipv6地址，eg:
    # Bind:
    #   - "127.0.0.1"
    #   - "::1"
    Bind:
    - "0.0.0.0"
    # 端口号
    port: 26369
    # 日志等级，用于过滤输出日志，包括debug，info，warn，error，fatal五个等级
    loglevel: notice
    # 日志文件输出目录，当设置为空字符串时，日志文件不会写入磁盘
    logfile: ""
    # sentinel相关配置项
    SentinelItems:
    - "sentinel monitor mymaster 127.0.0.1 6369 1" # 设置监听的主节点信息，IP地址/端口
    - "sentinel down-after-milliseconds mymaster 30000" # 设置主节点主观下线的超时时间
    - "sentinel failover-timeout mymaster 180000"
    - "sentinel parallel-syncs mymaster 1"
    # - "sentinel auth-pass mymaster 123456"              # 设置哨兵访问主节点的密码,
    # - "sentinel config-epoch mymaster 0"                # 设置节点的选举纪元
    # - "sentinel leader-epoch mymaster 0"
    # - "sentinel known-replica mymaster 127.0.0.1 6380"  # 设置其他已知的从节点信息
    # - "sentinel current-epoch 0"                        # 设置当前哨兵选举纪元

SSL:
    # 是否开启ssl，如果开启则使用true，否则是false
    Enable: false
    # SSL监听端口，如果仅开启SSL监听，需要把NetWord: Port端口设置为0

    # 服务端SSL证书文件
    TlsCertFile: "./certs/ssl_tls_cert/server.crt"
    # 服务端SSL证书的密钥
    TlsKeyFile: "./certs/ssl_tls_cert/server.key"
    # 证书签发机构的证书文件，即可信从根证书
    TlsCaCertFile: "./certs/ssl_tls_cert/ca.crt"
    # 证书签发机构的证书文件夹，即多个可信的根证书，可放在一个文件夹中
    TlsCaCertDir: ""
    # 客户端SSL证书文件,为集群/主从模式使用
    TlsClientCertFile: "./certs/ssl_tls_cert/client.crt"
    # 客户端SSL证书的密钥，为集群/主从模式使用
    TlsClientKeyFile: "./certs/ssl_tls_cert/client.key"
    # 服务器是否验证客户端证书，设置为yes需要客户端提供证书，并且服务端作验证。 如果不需要验证客
    # 设置为optional,即客户端可以提供证书也可以不提供证书，但是如果客户端提供证书，那么amdc服
    TlsAuthClients: "no"
    # 主从模式是否使用TLS通信，当master开启了tls模式，那么从节点必须开启，否则无法连接master
```

sentinel.yaml转换前：

```
TLS/SSL:
    tls-ciphers: ""
    # TLSv1.3 允许的密码套件
    tls-ciphersuites: ""
    # 是否优先使用服务器的密码套件偏好
    tls-prefer-server-ciphers: no
    # 是否启用 TLS 会话缓存以加速重连
    tls-session-caching: yes
    # TLS 会话缓存的最大条目数，0 表示无限制
    tls-session-cache-size: 204800
    # TLS 会话缓存的超时时间（秒）
    tls-session-cache-timeout: 300

SENTINEL:
    # 定义要监控的主节点，格式为 <master-name> <ip> <amdc-port> <quorum>（如 mymaster
    sentinel monitor: mymaster 127.0.0.1 6359 2
    # 主从节点的认证用户名，格式为 <master-name> <username>
    sentinel auth-user: mymaster ""
    # 主从节点的认证密码，格式为 <master-name> <password>
    sentinel auth-pass: mymaster ""
    # 判定节点主观下线的超时时间（毫秒），默认 30000（30 秒）
    sentinel down-after-milliseconds: mymaster 30
    # Sentinel 之间通信的认证用户名
    sentinel sentinel-user: ""
    # Sentinel 之间通信的认证密码其他设置
    sentinel sentinel-pass: ""
    # 故障转移时从节点同步新主节点的并发数量，默认 1
    sentinel parallel-syncs: mymaster 1
    # 故障转移超时时间（毫秒），默认 180000（3 分钟）
    sentinel failover-timeout: mymaster 180000
    # 发生警告级事件时执行的通知脚本（如邮件、短信通知），格式为 <master-name> <script-path
    sentinel notification-script: mymaster ""
    # 故障转移后通知客户端重新配置的脚本，格式为 <master-name> <script-path>
    sentinel client-reconfig-script: mymaster ""
    # 是否禁止运行时修改脚本配置，默认 yes
    sentinel deny-scripts-reconfig: yes
    # 重命名命令，格式为 <master-name> <old-command-name> <new-command-name>
    sentinel rename-command: mymaster "" ""
    # 是否支持主机名（默认 no，需依赖 DNS）
    sentinel resolve-hostnames: no
    # 是否对外公布主机名（默认 no）
    sentinel announce-hostnames: no
    # NAT/Docker 环境下对外公布的 IP
    sentinel announce-ip: ""
    # NAT/Docker 环境下对外公布的端口
    sentinel announce-port: 0
```

sentinel.yaml转换后：

```
TLS/SSL:
    # TLSv1.3 允许的密码套件
    tls-ciphersuites: ""
    # 是否优先使用服务器的密码套件偏好
    tls-prefer-server-ciphers: no
    # 是否启用 TLS 会话缓存以加速重连
    tls-session-caching: yes
    # TLS 会话缓存的最大条目数，0 表示无限制
    tls-session-cache-size: 20480
    # TLS 会话缓存的超时时间（秒）
    tls-session-cache-timeout: 300

SENTINEL:
    # 定义要监控的主节点，格式为 <master-name> <ip> <amdc-port> <quorum>（如 mymaster
    sentinel monitor: mymaster 127.0.0.1 6369 1
    # 主从节点的认证用户名，格式为 <master-name> <username>
    sentinel auth-user: mymaster ""
    # 主从节点的认证密码，格式为 <master-name> <password>
    sentinel auth-pass: mymaster ""
    # 判定节点主观下线的超时时间（毫秒），默认 30000（30 秒）
    sentinel down-after-milliseconds: mymaster 30000
    # Sentinel 之间通信的认证用户名
    sentinel sentinel-user: ""
    # Sentinel 之间通信的认证密码其他设置
    sentinel sentinel-pass: ""
    # 故障转移时从节点同步新主节点的并发数量，默认 1
    sentinel parallel-syncs: mymaster 1
    # 故障转移超时时间（毫秒），默认 180000（3 分钟）
    sentinel failover-timeout: mymaster 180000
    # 发生警告级事件时执行的通知脚本（如邮件、短信通知），格式为 <master-name> <script-path
    sentinel notification-script: mymaster ""
    # 故障转移后通知客户端重新配置的脚本，格式为 <master-name> <script-path>
    sentinel client-reconfig-script: mymaster ""
    # 是否禁止运行时修改脚本配置，默认 yes
    sentinel deny-scripts-reconfig: yes
    # 重命名命令，格式为 <master-name> <old-command-name> <new-command-name>
    sentinel rename-command: mymaster "" ""
    # 是否支持主机名（默认 no，需依赖 DNS）
    sentinel resolve-hostnames: no
    # 是否对外公布主机名（默认 no）
    sentinel announce-hostnames: no
    # NAT/Docker 环境下对外公布的 IP
    sentinel announce-ip: ""
    # NAT/Docker 环境下对外公布的端口
    sentinel announce-port: 0
```

9.3.6 注意事项

- 替换规则：--c-yaml指向的文件为AMDC V2.0.4版本配置替换模版，工具执行逻辑为「先读取--c-yaml模版文件的原有结构与基础配置->提取--go-yaml指向的AMDC V2.0.2及以下版本配置文件的有效配置项->将模版文件中对应配置项替换为AMDC V2.0.2及以下版本的配置值->最终覆盖写入--c-yaml指向的文件」，并非直接将AMDC V2.0.2及以下版本配置生成新的AMDC V2.0.4版本配置文件；
- 模版文件直接覆盖：工具完成配置项替换后，会直接覆盖--c-yaml指向的模版文件，无任何弹窗或命令行提示，转换前务必对该模版文件进行完整备份，避免原有AMDC V2.0.4版本配置结构或基础项丢失；
- 参数规则严格遵循：需二选一指定--server或--sentinel配置类型，不可省略、不可同时使用；--go-yaml(AMDC V2.0.2及以下版本源配置）和--c-yaml(AMDC V2.0.4版本替换模版）为必传参数，后必须紧跟实际文件路径，参数缺失、格式错误或路径无效会直接触发工具报错并退出；
- 转换方向与类型唯一：本工具仅支持AMDC V2.0.2及以下版本AMDC配置->AMDC V2.0.4版本AMDC配置的单向替换，不支持AMDC V2.0.4版本向AMDC V2.0.2及以下版本的反向转换，也不兼容其他版本AMDC的配置替换；且-server或--sentinel 需与源文件、模版文件的配置类型一致（服务端配置仅能替换服务端模版，哨兵配置仅能替换哨兵模版）；
- 替换后必须验证有效性：替换完成（无报错输出）后，需先检查--c-yaml文件是否正常覆盖、配置项是否已完成替换，再启动对应类型的AMDC V2.0.4版本AMDC（服务端或哨兵）加载该文件，确认服务能正常启动无配置相关报错，避免因配置项匹配异常导致服务启动失败。

9.4 常见问题排查

9.4.1 工具无法运行

问题1：终端输入amdc-check-rdb提示“command not found”（Linux)；

- 原因：工具路径未添加到系统环境变量；
- 解决：输入完整路径运行（如/usr/local/amdc/amdc-check-rdb)，或添加环境变量（echo "export PATH=/usr/local/amdc/:$PATH" >> /etc/profile，source /etc/profile）。

9.4.2 校验 / 修复文件失败

问题1:amdc-check-rdb或amdc-check-aof 提示“Permission denied"；

- 原因：无文件读取/写入权限；
- 解决：使用chmod修改文件权限(chmod 644 dump.rdb)，或使用sudo运行工具(sudo amdc-check-rdb --fix dump.rdb)。

问题2：修复后AMDC仍无法启动；

- 原因：文件损坏过于严重，或工具版本与AMDC版本不匹配；
- 解决：使用备份文件恢复，或升级或降级工具版本，与AMDC服务版本一致。

9.4.3 amdc-cli 无法连接AMDC服务

问题1：连接提示“Could not connect to AMDC at 127.0.0.1:6359: Connection refused"；

- 原因：AMDC服务未启动；
- 解决：启动AMDC服务（amdc-server /path/to/amdc.yaml, Linux）。

问题2：远程连接提示“Could not connect to AMDC at 192.168.1.100:6359:Connection timed out";

- 原因：防火墙未开放6359端口，或AMDC配置不允许远程连接；
- 解决：开放防火墙端口(firewall-cmd --add-port=6359/tcp --permanent， Linux)，修改AMDC配置(bind 0.0.0.0， protected-mode no)，重启AMDC服务。

问题3：连接后执行命令提示“NOAUTH Authentication required”；

- 原因：AMDC服务设置了密码，连接时未输入；
- 解决：连接时用-a选项输入密码，或交互模式输入AUTH密码。

9.5 总结

本文档详细介绍了AMDC四个核心工具的使用方法，重点总结如下：

- amdc-check-rdb：专注于RDB文件的校验与修复，核心选项--fix，修复前务必备份；
- amdc-check-aof：专注于AOF文件的校验与修复，支持常规修复(--fix)，风险需谨慎；
- amdc-benchmark:AMDC自带的性能测试工具，用于测试AMDC服务的吞吐量、响应时间等指标，核心是通过选项控制测试场景，适配不同性能评估需求。
- amdc-conf-conv:amdc-conf-conv是AMDC专属的配置文件格式转换工具，用于将AMDC V2.0.2及以下版本AMDC的配置文件转换为AMDC V2.0.4版本AMDC的配置文件。

以上四个工具均为AMDC自带，无需额外安装，日常运维中需熟练掌握，尤其是amdc-cli和amdc-benchmark，可高效完成AMDC的各类操作、故障排查及性能评估。使用过程中，务必注意数据安全和测试规范，避免误操作导致数据丢失或服务异常。

10 常见问题解决

10.1 问题一：端口占用与解决

安装AMDC会占用一些端口，同时客户端链接需要依赖内部的DNS服务，如果不能保证端口未被使用，或者是安装机器环境缺乏有效DNS，则会导致启动失败。 产品默认占用的端口：

- AMDC缓存核心默认端口：6359
- AMDC哨兵服务默认端口：26359

如果启动之前上述端口被占用，将导致启动失败。可以切换端口或停止正在使用该端口的程序。

10.2 AMDC缓存核心端口修改

修改配置文件:vim amdc.yaml，port字段

> *更多配置请参考：缓存配置（amdc.yaml）*

10.3 ACL文件加载

将ACL文件存放于AMDC缓存配置项"ACLFile"指定的目录之下，并在AMDC缓存命令行中执行 ACL load 即可重新加载ACL文件中的用户配置。

注：

1. 重新加载ACL文件将会代替缓存中原有的ACL规则。
2. 若ACL文件中有一项或多项是不合法的数据，整个文件都将无法被加载，缓存系统中现有的规则将继续使用。

10.4 解决AMDC主从故障切换

1. 缓存核心集群中若slave故障，系统将自动重连 若Slave出现故障导致宕机，恢复正常后会自动重新连接，Master收到Slave的连接后，将其完整的数据文件发送给Slave，如果Mater同时收到多个Slave发来的同步请求，Master只会在后台启动一个进程保存数据文件，然后将其发送给所有的Slave，确保Slave正常。
2. 缓存核心集群中若master出现故障，手动重启主服务即可实现客户端对系统的读写操作 当缓存核心master出现故障时，仍支持客户端对从服务的读操作，需要手动重启主服务，待主服务重启系统恢复正常后，再支持客户端对主服务的读写操作。

10.5 请求延迟问题

若AMDC请求出现明显延迟，可以通过以下办法查明具体问题：

1. slowlog命令：该命令可以查询出执行时间较长的命令（时间长度可以通过amdc.yaml文件中的slowlog参数进行配置），从而更好地跟踪代码执行过程。
2. –bigkeys：使用amdc-cli客户端，连接并加上 –bigkeys参数，来查询value最大的值，从而判断是否因为值过大导致请求延时。
3. –memkeys：同上，该参数查询key-value整体的内存占用大小，从而判断是否因为key-value过大导致请求延时。

10.6 如何检测执行了那些命令

使用amdc-cli进行连接，然后使用monitor命令即可监控AMDC持续接受到了哪些命令，也可使用以下命令后台记录到monitor.log中：

```
 amdc-cli -h [ip] -p [port] [-a password] monitor >monitor.log 2>&1
&
```

想要停止记录monitor.log时使用以下命令：

```
 ps -ef | grep amdc-cli | grep -v grep | awk '{print $2}' | xargs
kill -9
```

10.7 进程还在但无法连接

在客户端超过linux限制或者使用内存超过license限制内存时，有可能会出现进程在，但是连接超时/无法连接的情况。

1. 客户端超过linux限制： 某些linux系统会默认客户端连接数上限为1024，当业务应用过多，产生超过1024的客户端连接时，超过部分会被linux限制无法继续连接。

   解决办法：永久设置用户级别的文件描述符限制，在/etc/security/limits.conf中添加：

   ```
   * soft nofile 65535
   * hard nofile 65535
   ```

   注：不同的linux系统可能修改的方式有差异，如果以上无法生效，请自行搜索对应系统的TCP连接数.

2. 使用内存超过license限制内存：license的默认内存限制一般为6G（大部分都是），当业务系统写入超过6G的数据并且持续写入的情况下，会造成AMDC频繁触发内存淘汰算法，造成AMDC的性能下降，在持续大量的并发情况下会出现某些请求超时，从而导致业务段报错。

解决办法：扩容即可解决对应问题，扩容有两种方法，一为申请更大内存的license文件，直接扩大节点可使用的内存（不建议超过16G，过大会导致查询效率变低）；二为做成分布式集群模式（对应redis的cluster模式）。

10.8 dir 路径问题

配置文件中的配置项 "dir" 配置的路径，会在程序读取完配置文件后，将程序的当前执行文件路径转为dir配置的地址，并非amdc-server该可执行文件的文件路径。 在此请注意： 处理方式一:证书路径、日志文件路径等，统一用绝对路径填写，稳固可靠。 处理方式二：如若依旧需要使用相对路径，请对配置中的路径检查， 请基于配置项 "dir" 配置的路径来修改对应证书、日志等等的路径。

10.9 密码设置问题

在使用amdc主从模式的时候，我们知道需要配置requirepass为节点增设密码。

这里注意，还要对从节点的 masterauth 配置项填入主节点的 requirepass，以便从节点和主节点正常交互。否则如果主节点设置了 requirepass 而从节点未设置 masterauth，或密码错误，复制连接会立即被主节点拒绝，在日志中会出现 (error) NOAUTH Authentication required. 或 (error) ERR invalid password 错误。

--建议无论集群服务还是主从服务，无论主节点、从节点，都将requirepass、masterauth 配置完整，防止主从切换后，旧主的认证失效问题。

同理，哨兵监控的同时需要配置节点的密码。

哨兵需要持续监控所有主从节点，并在主节点故障时执行自动故障转移。因此，哨兵也必须能够"进入"这些受密码保护的节点。这是通过哨兵自身的配置文件中的 sentinel auth-pass 指令实现的： sentinel auth-pass

10.10 日志配置问题

系统可以在机器上进行日志配置管理amdc的server.log，结合自身业务对默认写入server.log的日志做简单的处理。能让日志更加个性化易用。

10.11 主从缓冲区配置问题

repl-backlog-size 是主从缓冲区的大小，默认配置为1mb，无法对大量数据交互的主从服务提供缓冲功能，建议往大调至128mb。

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com