Apusic 金蝶天燕

生成时间：2026-06-04

用户手册

金蝶Apusic分布式缓存v2.0.3

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 修订说明

本文根据实际情况进行更新，最新版本包含历史修改记录。

日期	手册版本	适用产品	更新说明
2025年7月	V2E02F01	AMDC v2.0.3c	AMDC c版本用户手册

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；并兼容Redis协议与持久化数据文件，实现简单快捷平稳替换Redis。

3 功能清单

3.1 缓存核心

功能	功能说明
多数据类型缓存	提供string、list、hash、set、sortedset、GEO、hyperloglog、stream的数据缓存类型。
发布订阅	实现了发布订阅功能，增强系统功能性
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。
IP白名单	支持IP、网段的白名单过滤，提高安全性。
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。
持久化	为缓存核心提高可用性，防止在宕机时丢失数据。
Lua脚本支持	支持使用lua脚本操作缓存核心。
多线程模式	支持多线程并发请求处理，提升系统吞吐量。
主从模式	支持主从备份。
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。
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。

4 相关概念

名词	含义	使用说明
主节点	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
从节点	从主节点复制数据，负责数据读写，不参与数据同步	从节点不参与数据读写，只负责数据同步，当主节点挂掉时，从节点自动切换为主节点，当主节点恢复时，从节点自动切换为从节点。
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

5 产品安装

5.1 环境支持

平台类型	系统类型
芯片类型	华为鲲鹏、海思、飞腾、兆芯等X86/ARM架构芯片
操作系统	银河麒麟系列、统信UOS、中标麒麟等
其他Linux系列	RedHat系列、CentOS系列、Ubuntu系列等

5.2 介质支持

介质类型	是否支持
主机介质	√
docker镜像	√
kubernetes(k8s)	√

5.3 推荐配置

部署模式	操作系统	安装内容	硬件规格 (CPU/内存/硬盘)	服务器台数
<!--	单机	Linux	AMDC控制台、AMDC服务	8核/16G/100G
主从	Linux	AMDC控制台、AMDC服务	8核/16G/100G	2
哨兵	Linux	AMDC控制台、AMDC服务	8核/16G/100G	3
集群	Linux	AMDC控制台、AMDC服务	8核/16G/100G	3
单机	Linux	AMDC服务	8核/16G/100G	1

| 主从 | Linux | AMDC服务| 8核/16G/100G | 2 |

| 哨兵 | Linux | AMDC哨兵服务 | 8核/16G/100G | 3 |

| 代理 | Linux | AMDC代理 | 8核/16G/100G | 3 |

5.4 部署前准备

本部分将主要介绍AMDC缓存核心、AMDC代理、AMDC哨兵的安装部署

5.4.1 获取安装包

从金蝶天燕官方网站下载金蝶Apusic分布式缓存软件安装包，或从金蝶Apusic分布式缓存软件产品光盘中获得相应的安装包文件。

5.4.2 关于License

使用官方授权的有效期内license文件。

5.4.2.1 获取License文件

License文件需要使用特征码来生成，不同的机器有不同的特征码，所以并不通用。 在终端使用命令启动时，会在终端打印特征码信息，特征码为SZTY开头，如下图所示，红色线部分的内容为特征码内容：

```
[INFO][Mon, 23 Dec 2024 10:13:29 CST] AMDC KBC authorization code: 'SZTY-1893184753' for IP: 172.24.6.110
```

使用授权码到金蝶KBC系统进行license申请（可联系销售或技术支持人员进行申请）。

其他方式请参考产品license使用

5.4.3 模式选择

- 单机：对缓存依赖程度低，仅提升系统响应速度作用的场景下使用。
- 主从：对缓存依赖程度低，提升系统响应速度作用，但数据比较重要或需要做快速替换的场景下使用。
- 哨兵：对缓存依赖程度高，需要实现快速故障转移的场景下使用。

5.5 主机部署

通过命令行方式部署AMDC缓存核心步骤主要有：

5.5.1 缓存核心

5.5.1.1 单机模式

步骤：

1. 上传AMDC缓存核心安装包(amdc_amd64.tar.gz)至目标服务器的安装目录下（如：/opt目录下）
2. 解压安装包：`tar -zxf AMDC-Core-[version]-[date]-[arch].tar.gz`
3. 进入解压后的文件夹：`cd amdc`
4. 执行启动命令启动AMDC缓存核心：`./amdc-server amdc.yaml`

相关配置项：

参数名称	解析	使用
bind	可以指定客户端访问缓存的ip地址，除绑定地址外无法连接。如：127.0.0.1，仅本机可访问；0.0.0.0表示所有可通过本机所有IP皆可访问	监听的ip地址，默认值0.0.0.0时不需要也不可以绑定其他地址；此外可以绑定多个地址，建议添加本地访问IP及远程访问IP，如： bind: - "127.0.0.1" - "192.168.0.190"
port	缓存核心的端口号	port: 6359
requirepass	设置密码	auth密码，在users.acl存在的情况下，server优先使用users.acl中的密码

5.5.1.1.1 测试

```
./amdc-cli -h ip -p 6359 -a password info
```

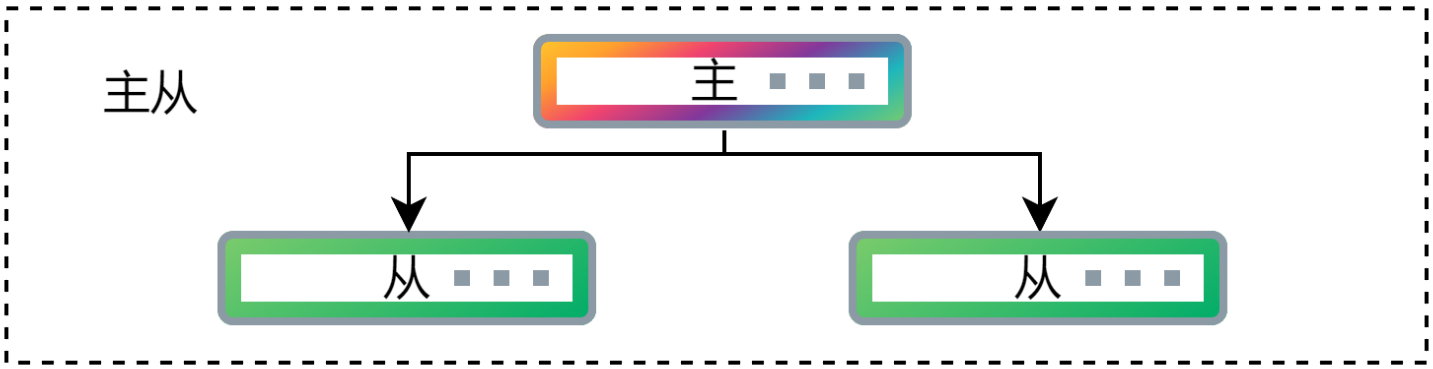
若正常打出info信息，即连接成功。

```
172.24.4.110:9000> info server
# Server
AMDC_version:V2.0
AMDC_mode:standalone
os:linux
arch_bits:amd64
process_id:28892
run_id:9dd323a658869b82607cd9d65d3cd6aa205897ec
tcp_port:9000
uptime_in_seconds:7626439
uptime_in_days:88
hz:10
lru_clock:6288079
executable:/opt/amdc/AMDCServer/amdc/amdc-server
config_file:/opt/amdc/AMDCServer/amdc/9000/conf.yaml
```

5.5.1.2 主从模式

1. 上传AMDC缓存核心安装包(amdc_amd64.tar.gz)至目标服务器的安装目录下(如：/opt目录下)，至少需要2个节点，组成1主1从，也可以部署更多从节点形成1主n从
2. 解压安装包： `tar -zxf AMDC-Core-[version]-[date]-[arch].tar.gz`
3. 进入解压后的文件夹: `cd amdc`
4. 若该节点为主节点，接(5)；若该节点是从节点，在amdc.yaml文件中Replication部分的replicaof配置项中，配置主节点的"ip port"
5. 执行启动命令启动AMDC缓存核心: `./amdc-server amdc.yaml`

主从部署图



相关配置项：

参数名称	解析	使用
bind	可以指定客户端访问缓存的ip地址，除绑定地址外无法连接。如：127.0.0.1，仅本机可访问；0.0.0.0表示所示可通过本机所有IP皆可访问	监听的ip地址，默认值0.0.0.0时不需要也不可以绑定其他地址；此外可以绑定多个地址，建议添加本地访问IP及远程访问IP,如： bind: - "127.0.0.1" - "192.168.0.190"
port	缓存核心的端口号	port: 6359
requirePass	设置密码，可以为空。从节点密码建议与主节点密码保持一直，以保证主从切换之后能够正常连接使用。	auth密码，在users.acl存在的情况下，server优先使用users.acl中的密码，eg: requirePass: "123456"
replicaof	指定主节点的ip和端口	设置启动服务器为指定服务器的从节点，eg: replicaof: "127.0.0.1 6378"
masterauth	主节点的密码，若主节点无密码则不需要填	masterauth: "123456"

5.5.1.2.1 测试

```
./amdc-cli -h ip -p 6359 -a password info replication
```

若信息中的slave（主节点）/replcaof（从节点）指向正确的IP:port，即为正常.

```
172.24.3.110:7020> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=172.24.3.110,port=7021,state=online,offset=42,lag=1
master_replid:d1f3e1118d7dff58c366f7d11b0b38b15ec325f2
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:42
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:42
repl_last_start_time:0
repl_last_repl_time:0
repl_last_master_addr:
172.24.3.110:7020> _
```

```
172.24.3.110:7021> info replication
# Replication
role:slave
master_host:172.24.3.110
master_port:7020
master_link_status:up
master_last_io_seconds_ago:7
master_sync_in_progress:0
slave_repl_offset:84
slave_priority:100
slave_read_only:1
connected_slaves:0
master_replid:d1f3e1118d7dff58c366f7d11b0b38b15ec325f2
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:84
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:84
repl_last_start_time:1734341632966
repl_last_repl_time:157
repl_last_master_addr:
172.24.3.110:7021>
```

5.5.1.3 哨兵模式

1. 搭建一个可用主从模式集群，并将哨兵文件amdc-sentinel与sentinel.yaml放置到目标服务器路径下；
2. 配置哨兵sentinel.yaml文件，bind绑定访问哨兵的IP，port是哨兵应用的端口，其中SentinelItems中的sentinel monitor项必须指定为主节点。

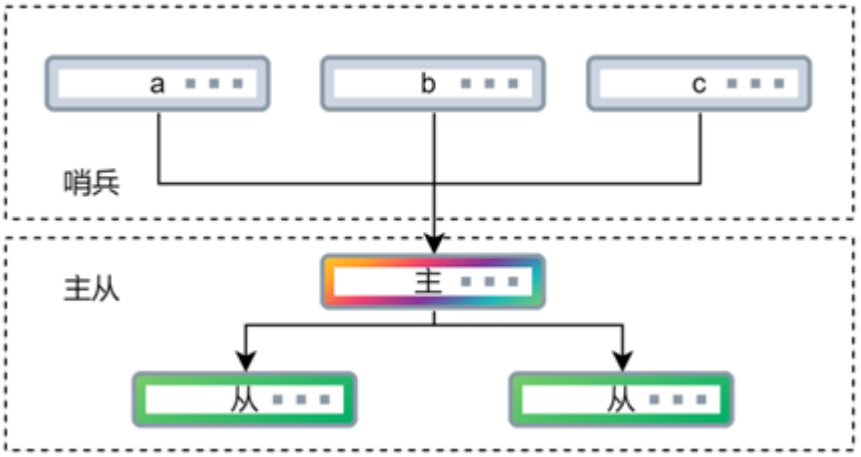
```
Sentinel:
  # 绑定的ip地址，可以绑定多个ip地址，支持ipv4/ipv6地址，eg:
  # Bind:
  #   - "127.0.0.1"
  #   - "::1"
  Bind:
    - "127.0.0.1"
  # 端口号
  Port: 26359
  # 日志等级，用于过滤输出日志，包括debug, info, warn, error, fatal五个等级
  LogLevel: "debug"
  # 日志文件输出目录，当设置为空字符串时，日志文件不会写入磁盘
  LogFile: ""
  # license文件位置
  LicensePath: "./license.xml"
  # sentinel相关配置项
  SentinelItems:
    - "sentinel monitor mymaster 127.0.0.1 6379 1"       # 设置监听的主节点信息，IP地址/端口/判断为主观下线的哨兵数
    - "sentinel down-after-milliseconds mymaster 30000"  # 设置主节点主观下线的超时时间（单位毫秒）
    - "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，如果主节点有设置密码，请求123456为目标密码并取消注释
    # - "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
```

添加机器对外的ip

3. **执行启动命令启动AMDC哨兵**

   启动命令：`./amdc-sentinel sentinel.yaml`

   哨兵模式部署图



   相关配置项：

参数名称	解析	使用
bind	可以指定客户端访问哨兵的ip地址，除绑定地址外无法连接。默认127.0.0.1，仅本机可访问。	监听的ip地址，可以绑定多个地址，建议添加本地访问IP及远程访问IP,如： bind:

		- "127.0.0.1" - "192.168.0.190"
port	缓存核心的端口号	port: 6359
SentinelItems	哨兵的具体配置，可以参考Redis的哨兵配置	其他配置可以按需求改动过，但是monitor一定要设置为主节点的ip和端口 - "sentinel monitor mymaster 127.0.0.1 6379 1" 若主节点有密码，则加上密码校验参数 - "sentinel auth-pass mymaster 123456"

5.5.1.3.1 测试

```
./amdc-cli -h ip -p 26359 -a password info sentinel
```

若信息中显示正常的主节点、从节点以及其他sentinel信息，即为正常,如下图中的master0。

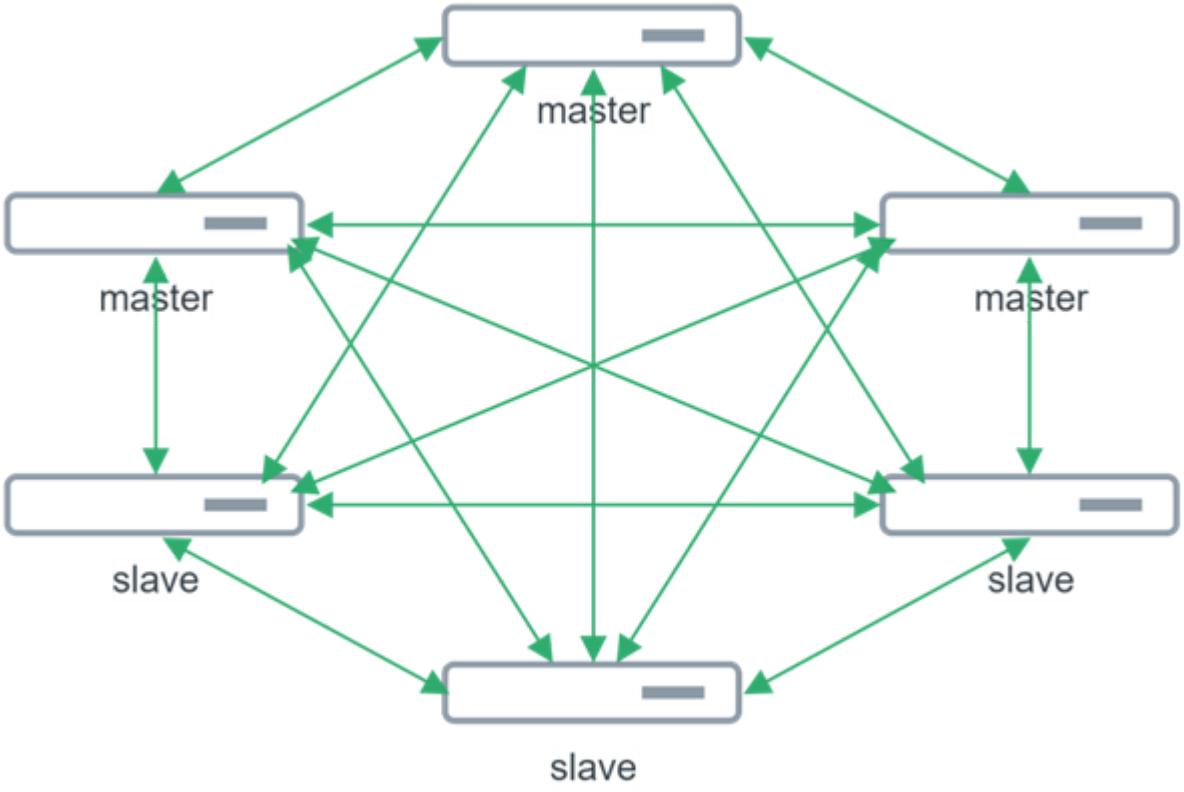
```
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=172.24.3.110:7030,slaves=1,sentinels=1
sentinel_last_master_addr:
sentinel_last_failover_time:0
sentinel_last_failover_s_time:0
172.24.3.110:7032>
```

5.5.1.4 集群模式

步骤：

1. 上传AMDC缓存核心安装包(amdc_amd64.tar.gz)至目标服务器的安装目录下(如：/opt目录下)，需要至少3个主节点来组成集群，一般情况下使用3主3从
2. 解压安装包： `tar -zxf AMDC-Core-[version]-[date]-[arch].tar.gz`
3. 进入解压后的文件夹: `cd amdc`
4. 给所有的节点都配置好bind、ClusterEnabled等参数，参考下方相关配置
5. 执行启动命令启动AMDC缓存核心:; `./amdc-server amdc.yaml`
6. 在随便一个节点中使用amdc-cli发送一次命令： `./amdc-cli --cluster create [ip:port ...(多个节点信息用空格隔开)] --cluster-replicas [number(每个主节点的从节点数)]`

集群部署图



相关配置项：

参数名称	解析	使用
bind	可以指定客户端访问缓存的ip地址，除绑定地址外无法连接。默认127.0.0.1，仅本机可访问。	监听的ip地址，可以绑定多个地址，建议添加本地访问IP及远程访问IP,如： bind: - "127.0.0.1" - "192.168.0.190"
port	缓存核心的端口号	port: 6359
requirepass	设置密码，所有节点的密码必须一致，可以为空。	auth密码，在users.acl存在的情况下，server优先使用users.acl中的密码，eg: requirepass: "123456"
masterauth	主节点的密码，若主节点无密码则不需要填，若有密码，不分主从所有节点都需要填写该参数。	masterauth: "123456"
cluster-enabled	是否开启集群模式	yes / no，eg: cluster-enabled: "yes"

cluster-config-file	每个集群节点配置文件名称，节点自动生成与更新，生成后不可重名，不可手动编辑	cluster-config-file: "./node.conf"

5.5.1.4.1 测试

```
./amdc-cli -h ip -p 26359 -a password cluster info
```

若信息中显示`cluster_state:ok`，即为正常。

```
172.24.4.110:7004> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:3
cluster_my_epoch:3
cluster_stats_messages_ping_sent:15151192
cluster_stats_messages_pong_sent:14662638
cluster_stats_messages_meet_sent:2
cluster_stats_messages_sent:29813832
cluster_stats_messages_ping_received:7331318
cluster_stats_messages_pong_received:7575597
cluster_stats_messages_meet_received:1
cluster_stats_messages_received:14906916
```

5.5.1.5 启动方式与表现

c版本启动AMDC缓存核心目前仅能通过命令行启动。

1. 启动方式

   命令行分为前端启动和后端启动模式，其中前端启动模式启动完成后不能再在终端进行其他命令行操作，如果要操作必须使用Ctrl+C，同时AMDC缓存核心停止。后台启动方式需要在配置文件中将daemonize配合项设置为yes。

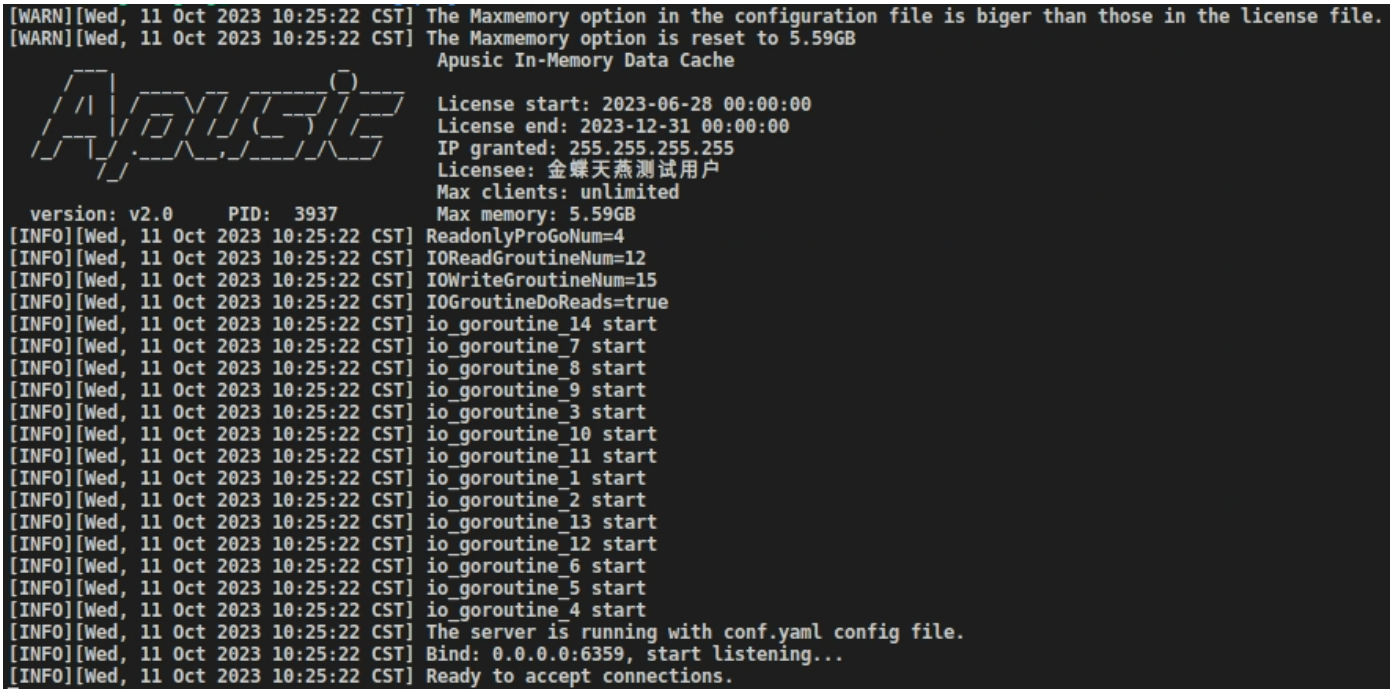
   - 启动AMDC缓存核心命令:`./amdc-server amdc.yaml`

2. 启动表现

   启动截图:

   ```
   ./amdc-server
   ```

启动成功：



5.5.1.6 停止AMDC缓存核心

AMDC缓存核心的停止方式有以下三种方式：

1. 客户端命令方式：通过客户端向AMDC发送shutdown命令
2. 强制结束进程的方式：通过终端命令强行终止AMDC缓存核心进程

考虑到AMDC缓存核心有可能正在将内存中的数据同步到硬盘中，强行终止AMDC缓存核心的进程有可能导致数据丢失，正确停止AMDC缓存核心的方式是通过客户端向AMDC缓存核心发送shutdown命令，关闭AMDC缓存核心。

- 方法1：客户端停止AMDC缓存核心:通过客户端向AMDC缓存核心发送shutdown命令 `./amdc-cli -h ip -p port -a password shutdown` 。
- 方法2：客户端链接正常情况下输入shutdown命令,当AMDC缓存核心收到shutdown命令后，会断开所有的客户端连接，然后根据配置执行持久化，最后退出。
- 方法3：强制结束AMDC缓存核心程序：使用kill -9 进程的pid，强行终止AMDC缓存核心进程。
  - 步骤一： 通过查看端口进程命令找到进程PID： `netstat -nltp | grep 6359`
  - 步骤二：执行kill -9命令，强制杀死进程： `kill -9` **进程**`ID`

5.5.1.7 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）**

5.6 Docker部署

5.6.1 AMDC缓存核心

AMDC缓存核心Docker包可以通过标准包进行构建，或者直接在官网获取对应架构的docker镜像（仅amd64/arm64架构）。

注意：若需要自行构建AMDC缓存核心镜像，则需要编写Dockerfile，跳过步骤1；否者跳过步骤2、3。

1. 加载镜像：`docker load -i amdc.tar`(名称自行修改,官方镜像名称通常为：AMDC-Core-[version]-Docker-[date]-[arch].tar)
2. 将amdc.yaml中的`license: ./license.lic`的修改为配置为`license: /config/license.lic`，方便后续挂载目录启动程序以及更换license、修改配置文件等操作。
3. 编写Dockerfile

   将Dockerfile放置在AMDC根目录。

```
# 基础镜像，from后面的替换为自有镜像名称
FROM base/ubuntu:18.04

# 镜像作者
LABEL maintainer="Apusic"

# 环境变量
ENV PATH=$PATH:/app

# 二进制文件目录
WORKDIR /app

# 添加文件
COPY ./amdc-server /app/amdc-server
COPY ./amdc-cli /app/amdc-cli

# 配置文件目录
WORKDIR /configs
```

```
# 添加文件
COPY ./amdc.yaml /configs/amdc.yaml
COPY certs /configs/certs
# 将license放置到容器中
# COPY license.lic /configs/license.lic

# 开放端口：服务端口、TLS端口、Exporter端口
EXPOSE 6359 6369 8080

# 启动服务
ENTRYPOINT ["amdc-server","amdc.yaml"]
```

4. **构建镜像**

   使用命令构建镜像：`docker build -t amdc:v2 .`

   构建完成后得到镜像名称为amdc:v2的Docker镜像。

5. **运行容器**

   - 运行容器：`docker run -d -p 6359:6359 --name amdc amdc:v2`
   - 挂载本地配置目录
     - 将license文件、配置文件放置到需要挂载的目录下：
     - 启动：`docker run -d -p 6359:6359 -v /local/conf_path:/app/configs --name amdc amdc:v2 ./configs/amdc.yaml`

6. **连接amdc服务**

   使用amdc-cli：`./amdc-cli -h ip -p 6359`

7. **其他**

   - 查看容器运行状态：`docker ps -a | grep amdc`
   - 查看容器日志：`docker logs -f amdc`
   - 停止容器：`docker stop amdc`
   - 删除容器：`docker rm amdc`
   - 删除镜像：`docker rmi amdc:v2`

5.6.2 AMDC哨兵

官网的docker镜像不包含哨兵，需要自行构建。

1. 编写dockerfile

   将dockerfile放置在AMDC根目录。

```
# 基础镜像，from后面的替换为自有镜像名称
FROM base/ubuntu:18.04

# 镜像作者
LABEL maintainer="Apusic"

# 环境变量
ENV PATH=$PATH:/app

# 二进制文件目录
WORKDIR /app

# 添加文件
COPY ./amdc-sentinel /app/amdc-sentinel
COPY ./amdc-cli /app/amdc-cli

# 配置文件目录
WORKDIR /configs

# 添加文件
COPY ./sentinel.yaml /configs/sentinel.yaml
COPY certs /configs/certs

# 开放端口：服务端口、TLS端口
EXPOSE 26359 26369
```

```
# 启动服务
ENTRYPOINT ["amdc-sentinel","sentinel.yaml"]
```

2. **构建镜像**

   使用命令构建镜像：`docker build -t amdc-sentinel:v2 .`

   构建完成后得到镜像名称为amdc-sentinel:v2的docker镜像。

3. **运行容器**

   运行容器：`docker run -d -p 6359:6359 --name amdc-sentinel amdc-sentinel:v2`

   挂载本地配置目录：`docker run -d -p 26359:26359 -v /local/conf_path:/app/configs --name amdc-sentinel amdc-sentinel:v2 ./configs/sentinel.yaml`

4. **连接amdc服务**

   使用amdc-cli：`./amdc-cli -h ip -p 26359`

5. **其他**

   - 查看容器运行状态：`docker ps -a | grep amdc-sentinel`
   - 查看容器日志：`docker logs -f amdc-sentinel`
   - 停止容器：`docker stop amdc-sentinel`
   - 删除容器：`docker rm amdc-sentinel`
   - 删除镜像：`docker rmi amdc-sentinel:v2`

5.6.3 AMDC管控台

目前多租户版本不支持管控台操作

5.6.4 其他说明

如果要实现缓存的不同模式，操作与在主机上部署一样，请参考主机部署。

5.7 测试验证

5.7.1 缓存核心连接测试

连接命令：`amdc-cli -h ip -p port [-a password] [-c]`

1. **使用amdc-cli连接缓存核心**

```
./amdc-cli -h ip -p 6359
```

2. 使用redis-cli连接缓存核心

```
redis-cli -h ip -p 6359
```

3. 使用redis-benchmark测试缓存核心性能

```
amdc-benchmark -h ip -p 6359
```

4. 使用redis-cli连接哨兵

```
amdc-cli -h ip -p 26359
```

5. 使用redis-cli连接集群

```
amdc-cli -h ip -p 26359 -c
```

5.8 AMDC如何平稳替换Redis

AMDC兼容Redis协议以及各语言客户端，实现AMDC平稳替换Redis有两种方案：从节点转正、RDB/AOF文件继承。

5.8.1 从节点转正

利用主从模式的数据同步，来获取Redis中数据，待同步数据成功后,将Redis节点关闭，手动或等待哨兵或者集群的自动故障切换功能将AMDC切换为主节点。

手动将从节点转正，以下方法二选一:

1. 使用 `amdc-cli -h <ip> -p <port> slaveof no one`
2. 连接到从节点，使用命令 `replicaof no one`

哨兵将从节点转正:

1. 修改AMDC配置文件将Redis主节点加入到AMDC配置文件中
   `vim /安装目录/amdc/amdc.yaml`
   修改replicaof字段为： `replicaof: [Redis主服务ip] [主服务端口]`

2. 待同步数据成功后,AMDC设置为哨兵主节点并启动AMDC哨兵，哨兵启动后即可将Redis节点关闭 同步成功：

哨兵配置：

配置项	含义
bind	哨兵绑定的IP地址，可以绑定多个IP地址
port	哨兵监听的端口
sentinel monitor	主服务ip 端口 选举个数（少于从节点个数，多于从节点个数的一半）

5.8.2 RDB/AOF文件继承

步骤一：获取原Redis的数据持久化文件RDB/AOF,将该持久化文件放到AMDC配置文件中指定的位置：`cat /安装目录/amdc/amdc.yaml`，通过查看DbFileName、Dir字段确定文件位置。

配置项	默认值	含义
dbfilename	"dump.rdb"	rdb文件名，不包括路径
dir	"./"	rdb和aof文件存储在安装目录下

步骤二：重启AMDC节点。

6 快速开始

6.1 缓存核心快速开始

6.1.1 启动缓存核心

步骤如下：

1. 解压安装包：`tar -zxvf amdc-x.x.x.tar.gz`
2. 上传License
3. 启动AMDC服务：`./amdc-server /安装目录/amdc/amdc.yaml`

6.1.2 使用shell客户端amdc-cli连接

步骤如下：

1. 使用`./amdc-cli -h [ip] -p [port] [-a passsword]`。
2. 使用`set [key_name] [key_value]`可以设置一个key到缓存中。
3. 使用`get [key_name]`可以获取这个key的值。

7 产品使用介绍

7.1 产品license使用说明

7.1.1 授权文件类型支持

AMDC支持金蝶天燕认证、金蝶KBC认证、金蝶统一授权三种模式，默认为金蝶KBC授权验证。

7.1.2 不同授权类型及其配置说明

1. xml文件: 表示金蝶天燕本地授权，配置到`amdc.yaml`中即可。
2. lic文件: 表示金蝶KBC授权，配置到`amdc.yaml`中即可。
    - KBC特征码获取：执行命令`./amdc-server`，即显示授权码信息。
    - 授权码为SZTY开头的内容,如上为：`SZTY2500879438`
    - 使用授权码在KBC系统中申请授权文件。
    - 获取授权文件后，放入安装目录，并更新license.conf配置，重启AMDC即可。

    AMDC授权文件的配置文件为：`安装目录/amdc.yaml`

    ```
    license: "license.lic"
    ```

3. 授权中心: 表示金蝶天燕统一授权中心，使用`acls.properties`文件进行配置，根据注释填写即可，也可以参考文档中授权认证中心配置。
    - 如果租户名称不确定，可以填写为public。

注意，AMDC支持license热更新，使用新license文件内容覆盖原license内容即可。

7.1.3 通过环境变量配置统一授权

AMDC支持环境变量中设置统一授权的配置，如下关键项：

```
export apusic_acls_enable=                  # 开启变量统一授权
export apusic_acls_authUrls=                # 统一授权中心地址
export apusic_acls_ns=                      # 命名空间
export apusic_acls_tenant=                  # 租户名称
```

7.2 管控台使用介绍

目前多租户版本不支持管控台操作

7.3 缓存核心使用介绍

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。

7.3.1 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

7.3.2 AMDC集群

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、集群模式（目前多租户版本不支持该类型集群操作）。

7.3.2.1 AMDC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。

7.3.2.1.1 主从命令

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

7.3.2.2 AMDC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。

7.3.2.2.1 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：s
   entinel 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

7.3.2.3 AMDC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

7.3.2.3.1 集群命令

1. 为接收节点分配新的哈希插槽：
   CLUSTER ADDSLOTS slot [slot ...]
2. 为接收节点分配新的哈希插槽：
   CLUSTER ADDSLOTSRANGE start-slot end-slot [start-slot end-slot ...]
3. 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
4. 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
5. 返回指定哈希插槽中的本地键数：
   CLUSTER COUNTKEYSINSLOT slot
6. 将哈希插槽设置为接收节点中的未绑定：
   CLUSTER DELSLOTS slot [slot ...]
7. 将哈希插槽设置为接收节点中的未绑定：
   CLUSTER DELSLOTSRANGE start-slot end-slot [start-slot end-slot ...]
8. 强制副本执行其主站的手动故障转移：
   CLUSTER FAILOVER [FORCE|TAKEOVER]
9. 删除节点自己的插槽信息：
   CLUSTER FLUSHSLOTS
10. 从节点表中删除节点：
    CLUSTER FORGET node-id
11. 在指定的哈希插槽中返回本地键名称：
    CLUSTER GETKEYSINSLOT slot count
12. 提供有关AMDC Cluster节点状态的信息：
    CLUSTER INFO
13. 返回指定键的哈希槽：
    CLUSTER KEYSLOT key
14. 强制一个节点集群与另一个节点握手：
    CLUSTER MEET ip port
15. 返回节点 id：
    CLUSTER MYID

16. 获取节点的集群配置：
    CLUSTER NODES

7.3.3 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：

7.3.3.1 SSL配置项

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-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-auth-clients: "./certs/ssl_tls_cert/client.crt"
tls-client-key-flle	客户端SSL证书的密钥，为集群/主从模式使用	tls-auth-clients: "./certs/ssl_tls_cert/client.key"
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

7.3.3.2 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. ## 8 创建服务器私钥与证书 - `openssl genrsa -out server.key 2048`

```
openssl req -new -sha256 -subj "/O=APUISC /CN=AMD.com" -key server.key | openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -days 365 -extfile openssl.cnf -extensions server_cert -out server.crt
```

5. # 9 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`

9.0.0.1 服务器SSL启动

1. 修改amdc.yaml（关键段落）

```
port 0
tls-port 6369

# 证书与CA
tls-cert-file ./cert/amdc.crt
tls-key-file ./cert/amdc.key
tls-ca-cert-file ./cert/ca.crt

# 强制客户端证书验证（可选）
tls-auth-clients yes
```

2. 启动amdc `./amdc-server amdc.yaml`

9.0.0.2 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())
}
```

9.0.1 数据持久化

amdc为用户提供了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。

9.0.1.1 RDB

RDB持久化方式，会将amdc中的数据以RDB格式存储到硬盘中。

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。

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。

9.0.1.2 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。

1. 生成AOF文件的方式

在配置文件中将AppendOnly改为"yes"即可。

2. AOF数据恢复方式

   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。

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： `./amdc-aof-check --fix [filename.aof|filename.mainfest]`

9.0.2 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`

9.1 shell客户端(amdc-cli)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。

9.1.1 AMDC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

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

9.1.2 AMDC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。

9.1.2.1 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]#
```


```
[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

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --rdb 1.rdb
SYNC sent to master, writing 249 bytes to '1.rdb'
Transfer finished with success.
[root@linux-4-190 looper]# cat 1.rdb
REDIS0009◆ctime¶◆2cused-mem°◆repl-stream-db◆◆repl-id(dd399ab99571c81dce95f06f6f9167d9f87d944c◆
                                                                                repl-offset◆◆◆k6◆

amdc v2.0
◆◆◆◆◆◆◆◆1◆1◆2◆2◆3◆Qh◆4◆◆◆◆◆◆◆◆◆◆◆◆◆7amdc-cli v2.0
```

- 从机模式，将客户端当作服务端的从节点 --slave

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359
127.0.0.1:6359>
127.0.0.1:6359>
127.0.0.1:6359>
127.0.0.1:6359> set k10 v10
OK
127.0.0.1:6359>
```

```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --slave
SYNC with master, discarding 278 bytes of bulk transfer...
SYNC done. Logging commands from master.
"PING",
"PING",
"PING",
"PING",
"PING",
"PING",
"PING",
"PING",
"SELECT","0"
"set","k10","v10",
"PING",
```

- 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>
```

9.1.2.2 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

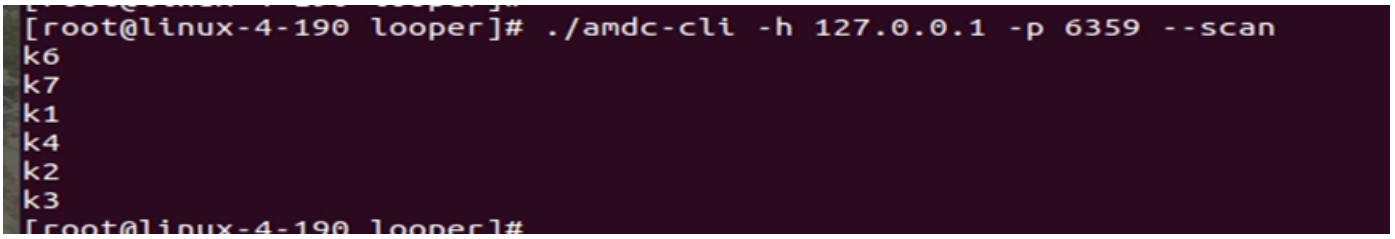
```
[root@linux-4-190 looper]# ./amdc-cli -h 127.0.0.1 -p 6359 --bigkeys

# 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

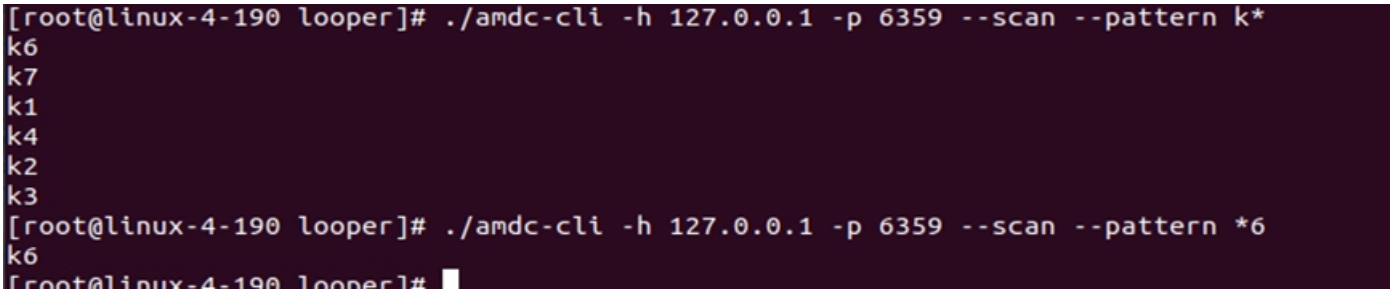
```
# Scanning the entire keyspace to find biggest keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).

[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]#
```

9.1.2.3 查询操作

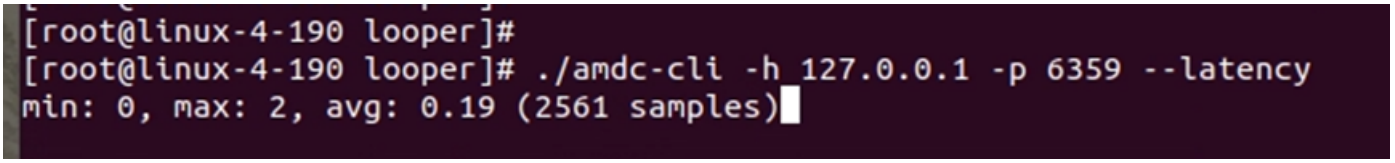
- 获取所有键列表 --scan

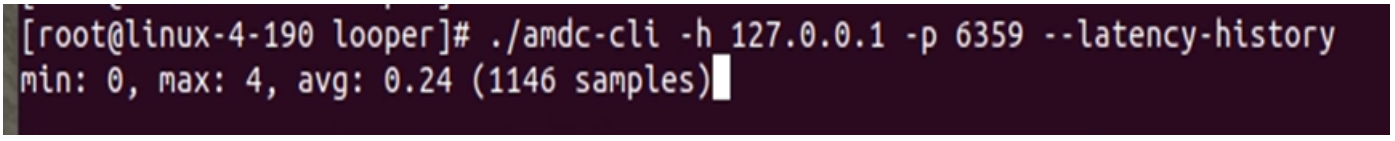


- 扫描指定的key --pattern （正则表达式）



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





9.1.2.4 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"
```

9.1.2.5 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>
```

9.1.2.6 集群操作

目前多租户版本不支持集群操作

9.2 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

9.3 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。

9.3.1 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
```

10 产品安全及调优配置

AMDC以最有性能作为模式进行设计，以减少服务端对客户端的资源消耗，所以无需额外配置参数进行调优。

1. 配置文件(amdc.yaml)

11 产品所有配置说明

11.1 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）。

11.1.1 缓存配置（amdc.yaml）

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"

11.1.2 哨兵配置（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
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
port	26379	哨兵进程监听的 TCP 端口
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
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

11.1.3 授权认证中心配置(acls.properties)

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_namespace	""	命名空间
apusic_acls_tenant	""	租户名称

12 常见问题解决

12.1 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。

12.2 问题二：AMDC缓存核心端口修改

修改配置文件:vim amdc.yaml，port字段

> *更多配置文件请参考部分I第三章配置文件。*

12.3 问题三：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。

12.4 问题四：如何解决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。

12.5 问题五：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。

12.6 问题六：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
```

12.7 问题七：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）。

12.8 问题八：系统CPU持续占用过高

与问题八的2相似，频繁触发淘汰算法会极大提高CPU占用率，并且长期保持高占用状态，解决办法同问题八2.

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